DataGeneral

Addendum

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

086–000195–00

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

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

Notice

Data General Corporation
4400 Computer Drive
Westboro, MA 01580

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

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

The addendum number appears on all pages in this addendum.

Updating Your Manual

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

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

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

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

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

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

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

093-000542-02

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

Ordering No. 093-000542

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

Notice

Data General Corporation
4400 Computer Drive
Westboro, MA 01580

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

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

Effective with:

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

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

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 {required$_1$ / required$_2$} which means you must enter one of the arguments. Do not type the braces; they only set off the choices.
[optional]	You have the option of entering this argument. Do not type the brackets; they only identify the argument as an option.
...	You may repeat the preceding entry.

Standard Symbols

Additionally, we use the following symbols:

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

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

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

Finally, in examples we use

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

Contacting Data General

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

Manuals

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

Telephone Assistance

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

Joining Our Users Group

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

End of Preface

Contents

Chapter 1 — Introducing the System Calls

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

?A Through ?Q

086–000195 updates
093–000542

?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

086–000195 updates
093–000542

Figures

Figure **?A Through ?Q**

?A Through ?Q

?R Through ?Z

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

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

(continued)

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

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

(continued)

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

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

(continued)

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

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

(concluded)

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

Input

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

Output

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

Error Codes in AC0

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

Why Use It?

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

Who Can Use It?

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

What It Does

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

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

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

?ALLOCATE Continued

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

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

Notes

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

?ASSIGN — Assigns a character device to a process.

AOS/VS

?ASSIGN
error return
normal return

Input

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

Output

AC0	Unchanged
AC1	Undefined
AC2	Undefined

Error Codes in AC0

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

Why Use It?

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

Who Can Use It?

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

What It Does

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

Notes

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

?AWIRE — Changes the wiring characteristics of the Agent.

AOS/VS

?AWIRE
error return
normal return

Input

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

Output

AC0	Unchanged
AC1	Unchanged
AC2	Unchanged

Error Codes in AC0

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

Why Use It?

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

Who Can Use It?

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

What It Does

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

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

Operating System Differences

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

Input

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

Output

AC0	Undefined
AC1	Undefined
AC2	Address of the packet.

Error Codes in AC0

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

Why Use It?

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

Who Can Use It?

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

What It Does

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

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

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

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

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

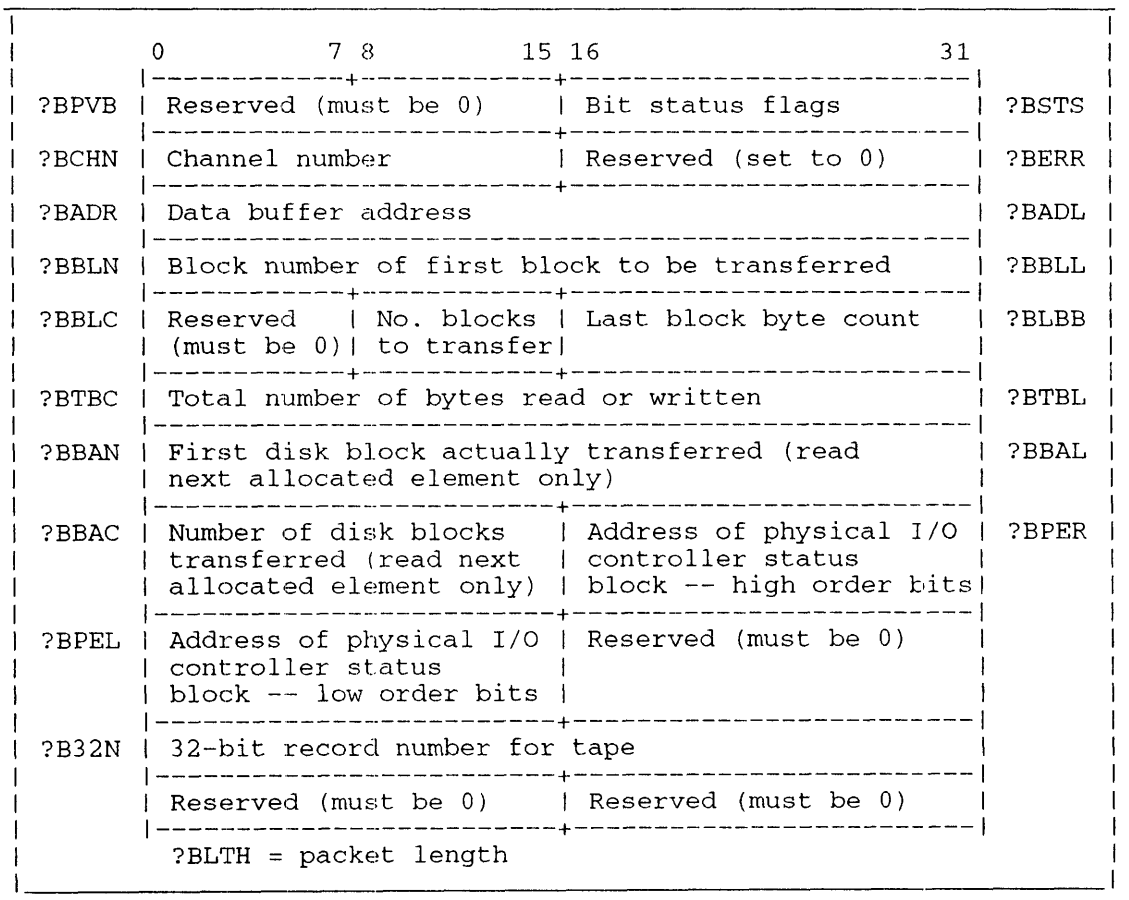


Figure 2–1. Structure of ?BLKIO Packet

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

Notes

- See the description of ?UBLPR in this chapter.

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

?BNAME
error return
normal return

Input

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

Output

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

Error Codes in AC0

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

Why Use It?

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

Who Can Use It?

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

What It Does

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

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

Who Can Use It?

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

What It Does

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

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

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

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

You cannot connect segment images within the same process.

Notes

- See the description of ?DCON in this chapter.

?CONFIG — Display or reset current MRC routes.

AOS/VS II

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

Input

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

Output

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

Error codes returned in AC0

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

Why Use It?

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

Who Can Use it?

Any user who has the access devices privilege.

What It Does

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

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

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

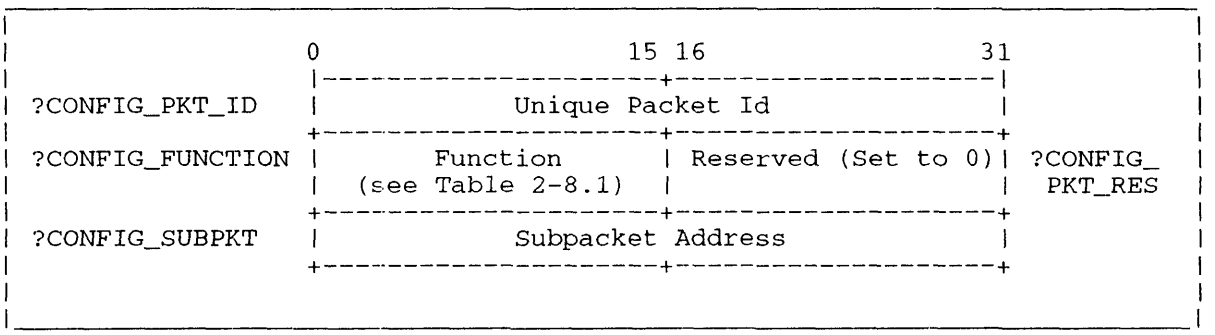


Figure 2–11.1. Structure of ?CONFIG Main Packet

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

Table 2–8.1. Valid ?CONFIG_FUNCTION Function Codes

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

What ?CONFIG_GET_CURRENT_ROUTE Does

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

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

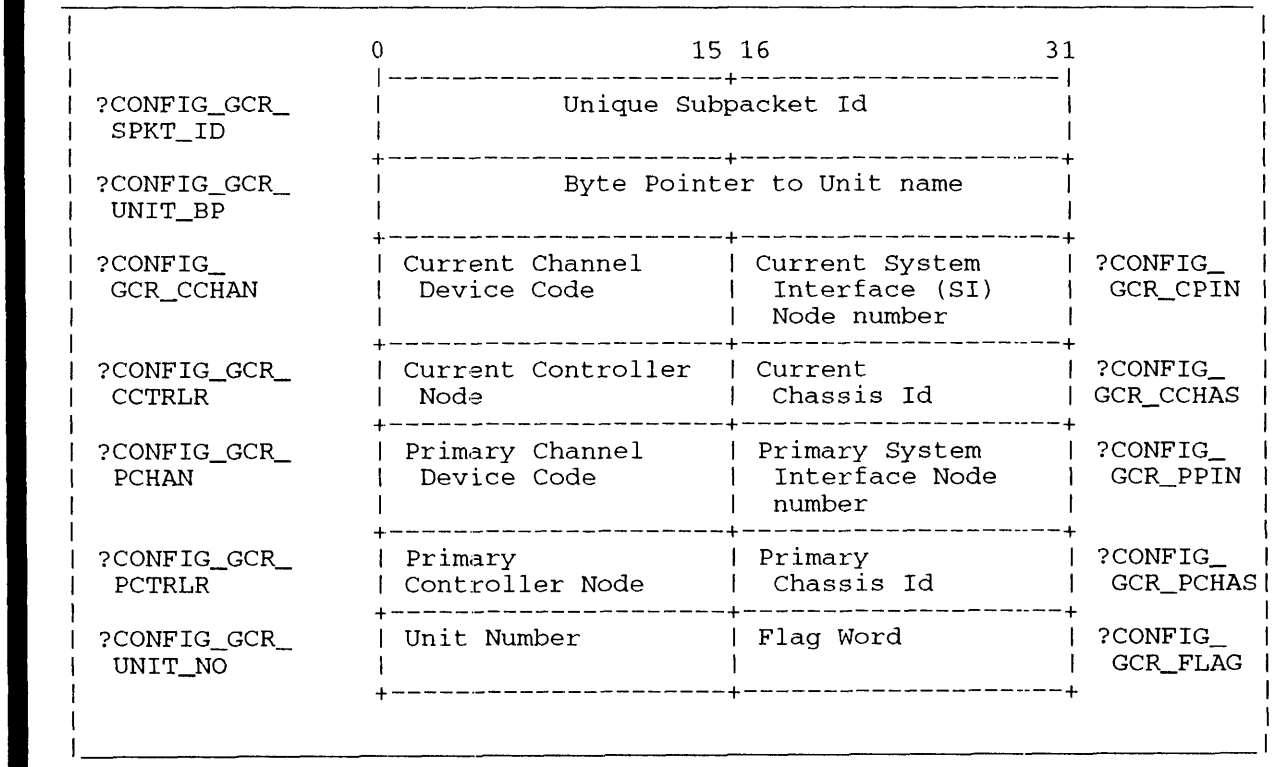


Figure 2–11.2. Structure of the ?CONFIG_GET_CURRENT_ROUTE Function Subpacket

Table 2–8.2. Contents of ?CONFIG_GET_CURRENT_ROUTE Function Subpacket

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

What ?CONFIG_RESET_MRC_CHAN Does

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

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

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

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

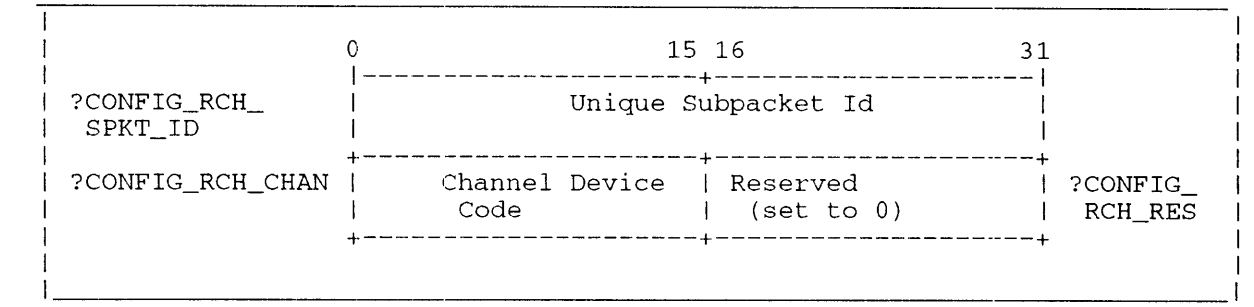


Figure 2–11.3. Structure of the ?CONFIG_RESET_MRC_CHANNEL Function Subpacket

Table 2–8.3 ?CONFIG_RESET_MRD_CHANNEL Function Subpacket Contents

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

What ?CONFIG_RESET_MRC_CTRLR does

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

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

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

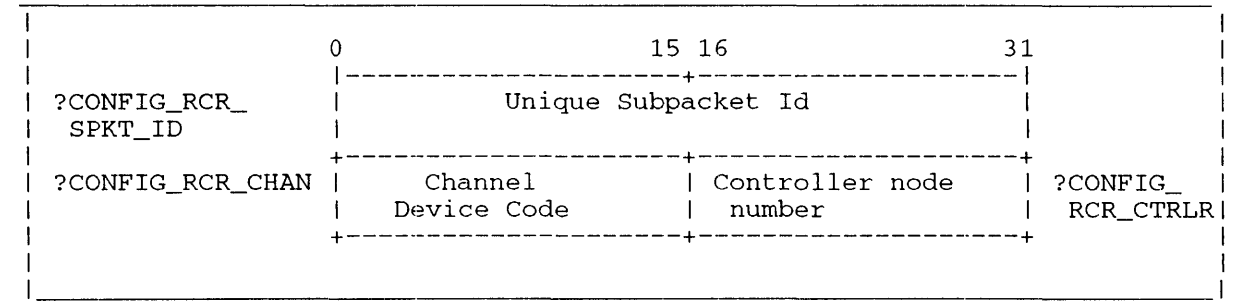


Figure 2–11.4. Structure of the ?CONFIG_RESET_MRC_CTRLR Function Subpacket

Table 2–8.4 ?CONFIG_RESET_MRC_CONTROLLER Function Subpacket Contents

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

Notes

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

?CONINFO

Request for addressing information on a terminal or console.

AOS/VS II

?CONINFO [packet address]
error return
normal return

Input

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

Output:

AC0 Unchanged or error code
AC1 Unchanged
AC2 Unchanged

Error codes returned in AC0

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

Why Use It?

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

Who Can Use It?

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

What It Does

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

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

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

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

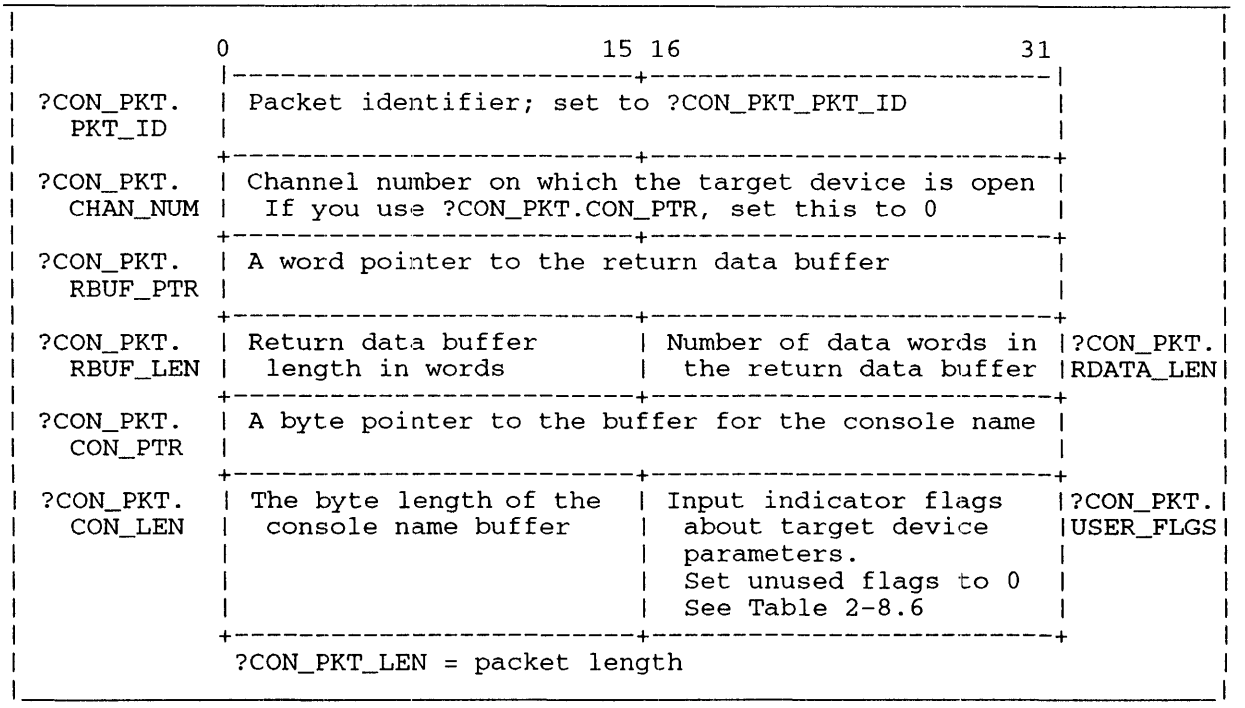


Figure 2–11.5. Structure of ?CONINFO Main Packet

Table 2–8.5. Contents of the ?CONINFO Packet

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

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

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

Return Packet Types

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

Return Packets and Line Numbers

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

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

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

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

?CON_TCP_RET_TYPE Return Type = 1

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

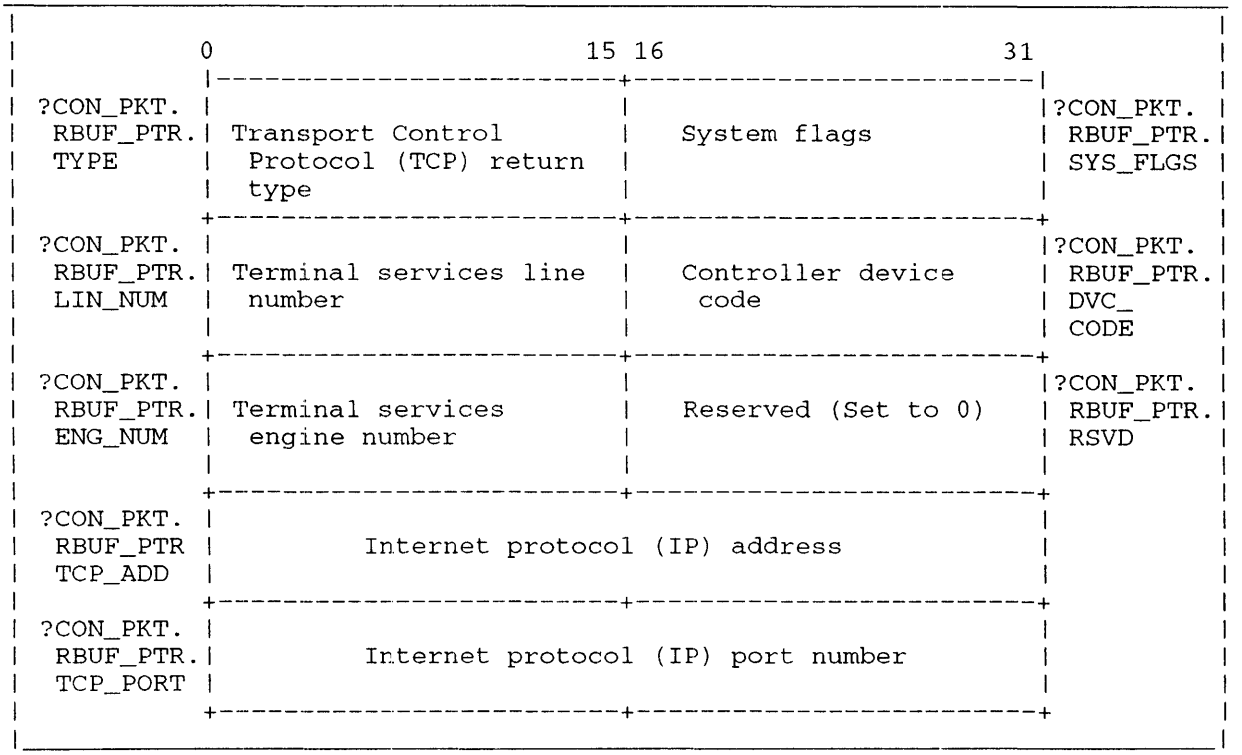


Figure 2–11.6. Structure of ?CON_TCP_RET_TYPE Return Packet

Table 2–8.8. Contents of ?CON_TCP_RET_TYPE Packet

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

?CON_XNS_RET_TYPE Return Type = 2

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

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

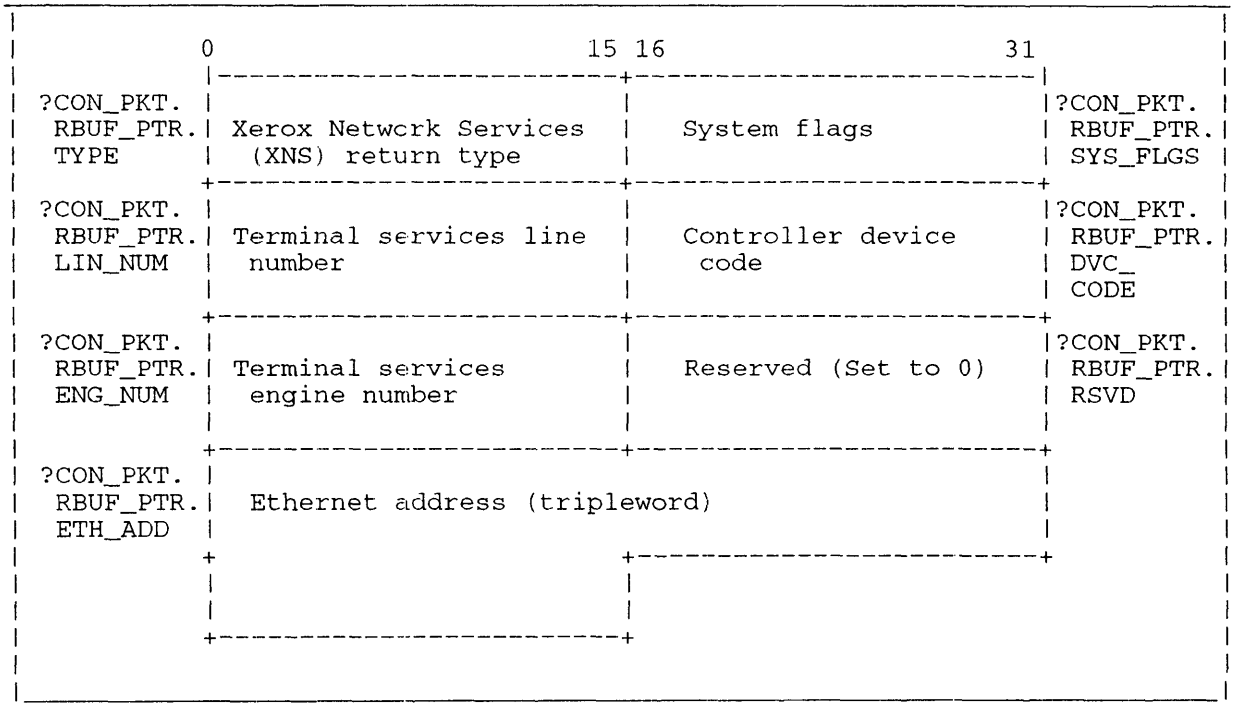


Figure 2–11.7. Structure of ?CON_XNS_RET_TYPE Return Packet

Table 2–8.9. Contents of ?CON_XNS_RET_TYPE Packet

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

?CON_CON_RET_TYPE Return Type = 6

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

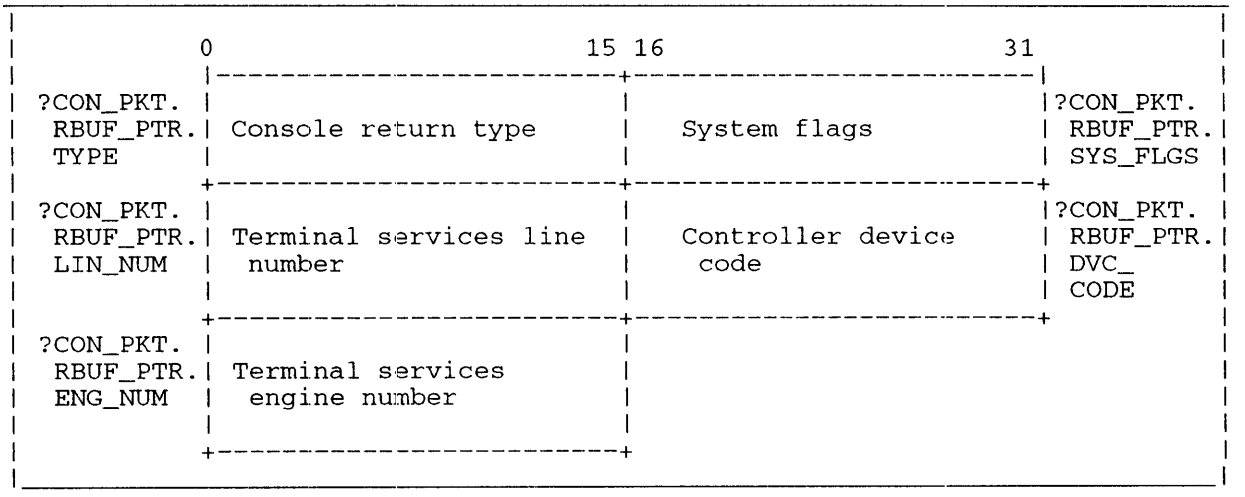


Figure 2–11.8. Structure of ?CON_CON_RET_TYPE Return Packet

Table 2–8.10. Contents of ?CON_CON_RET_TYPE Packet

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

?CON_TNET_RET_TYPE Return Type = 11

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

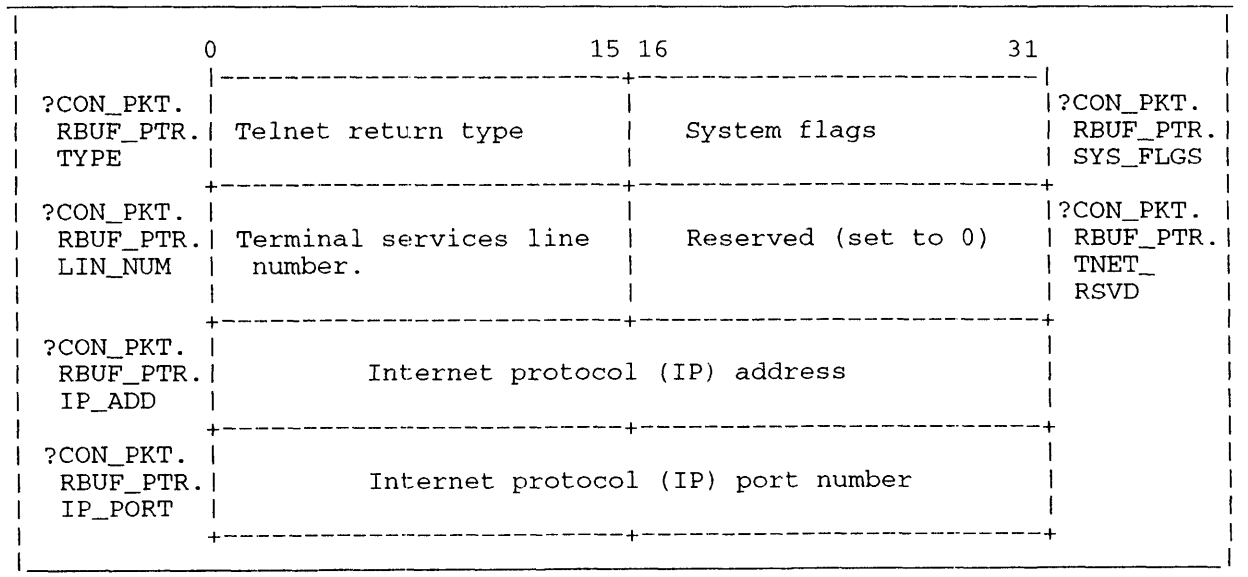


Figure 2–11.9. Structure of ?CON_TNET_RET_TYPE Return Packet

Table 2–8.11. Contents of ?CON_TNET_RET_TYPE Return Packet

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

?CON_ITC_MIN_DATA Return Type = 14

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

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

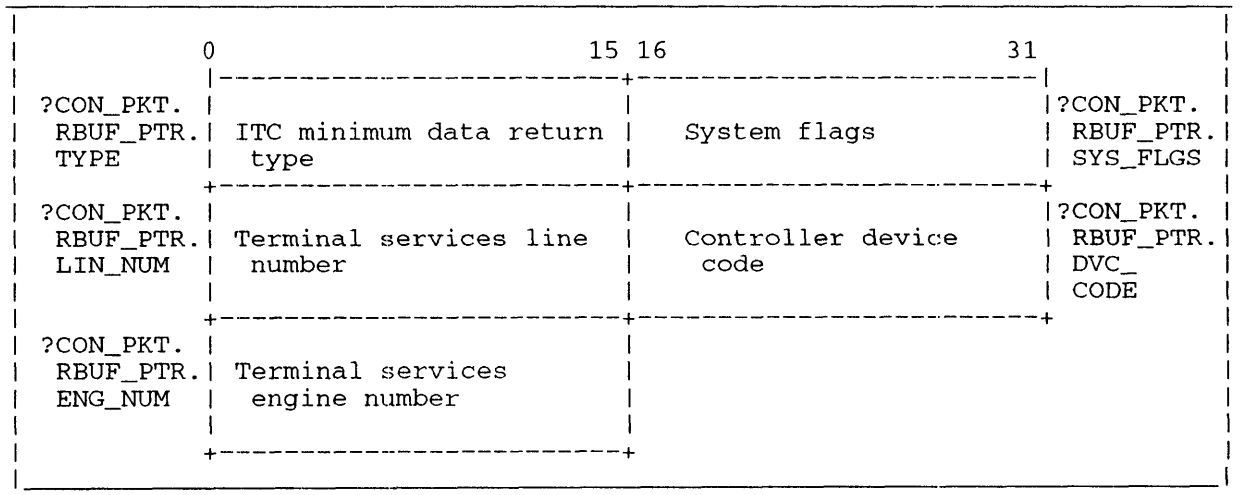


Figure 2–11.10. Structure of ?CON_ITC_MIN_DATA Return Packet

Table 2–8.12. Contents of ?CON_ITC_MIN_DATA Packet

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

?CON_TSC_MIN_DATA Return Type = 15

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

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

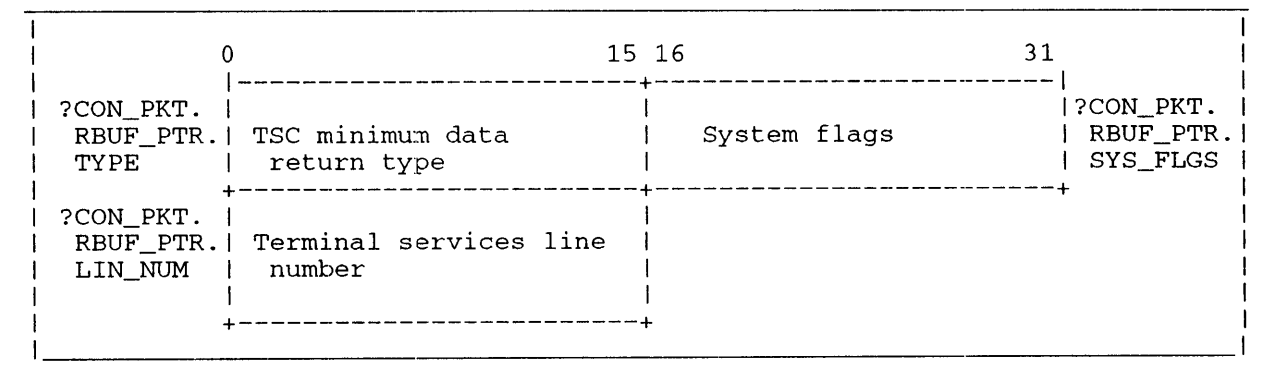


Figure 2–11.11. Structure of ?CON_TSC_MIN_DATA Return Packet

Table 2–8.13. Contents of ?CON_TSC_MIN_DATA Packet

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

?CON_PVC_RET_TYPE Return Type = 16

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

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

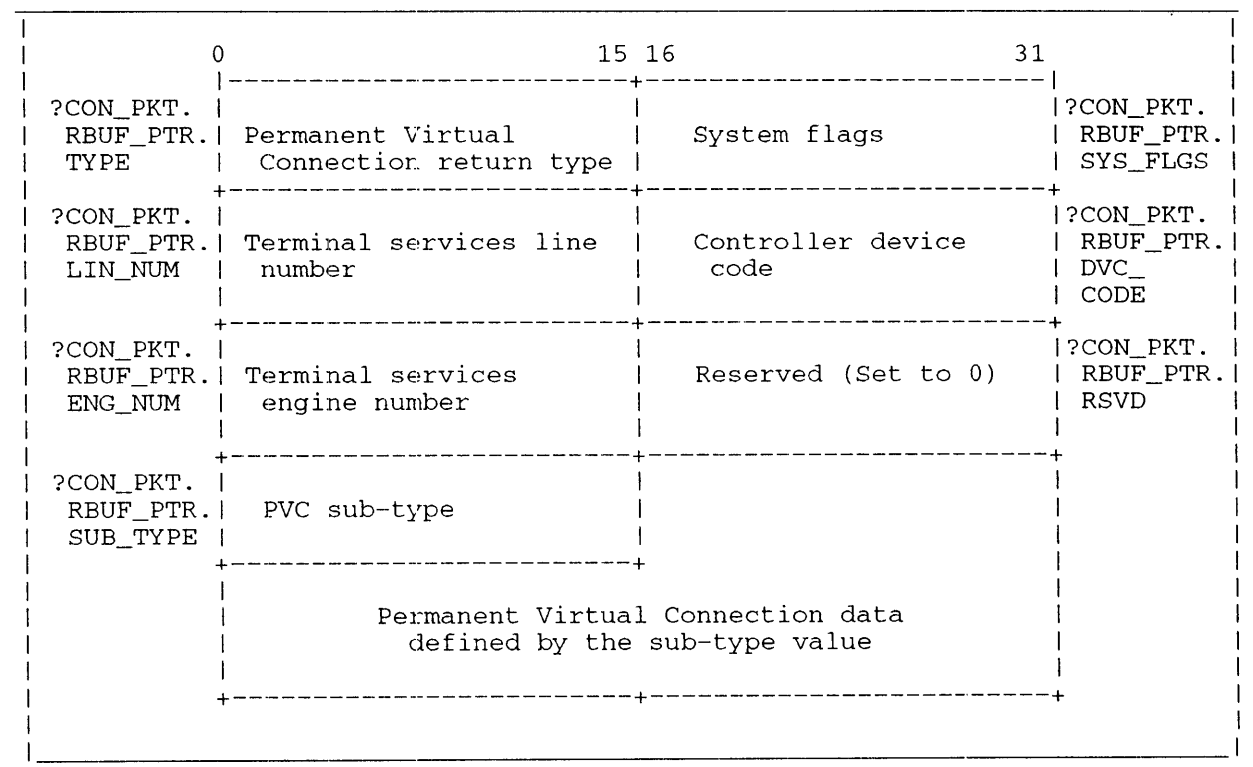


Figure 2–11.12. Structure of ?CON_PVC_RET_TYPE Return Packet

Table 2–8.14. Contents of ?CON_PVC_RET_TYPE Packet

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

?CON_PVC_RET_TYPE Subpackets

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

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

Table 2–8.15. ?CON_PVC_RET_TYPE Subpacket Types and Definitions

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

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

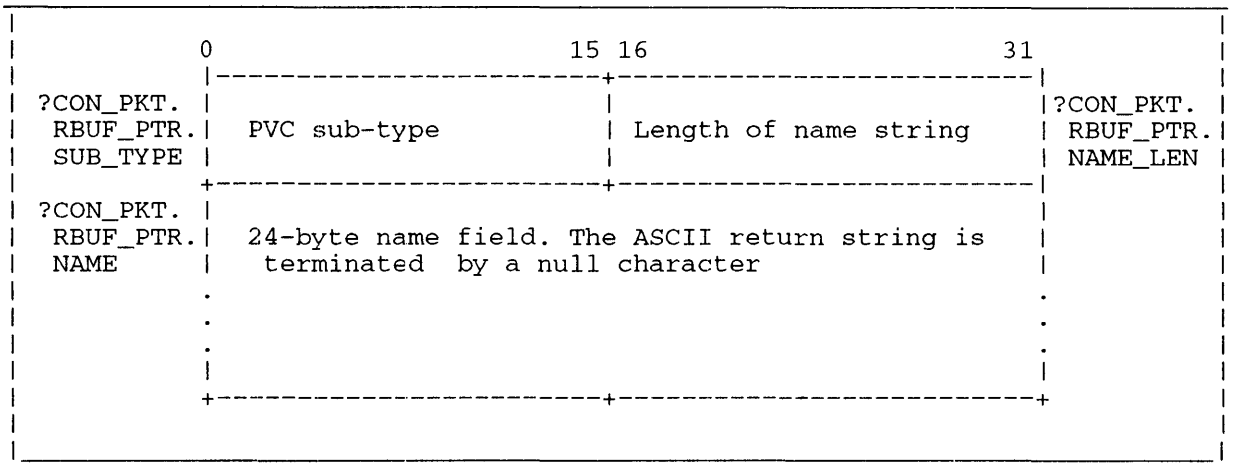


Figure 2–11.13. Structure of ?CON_PVC_NAME Return Packet

Table 2–8.16. Contents of ?CON_PVC_NAME Subpacket

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

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

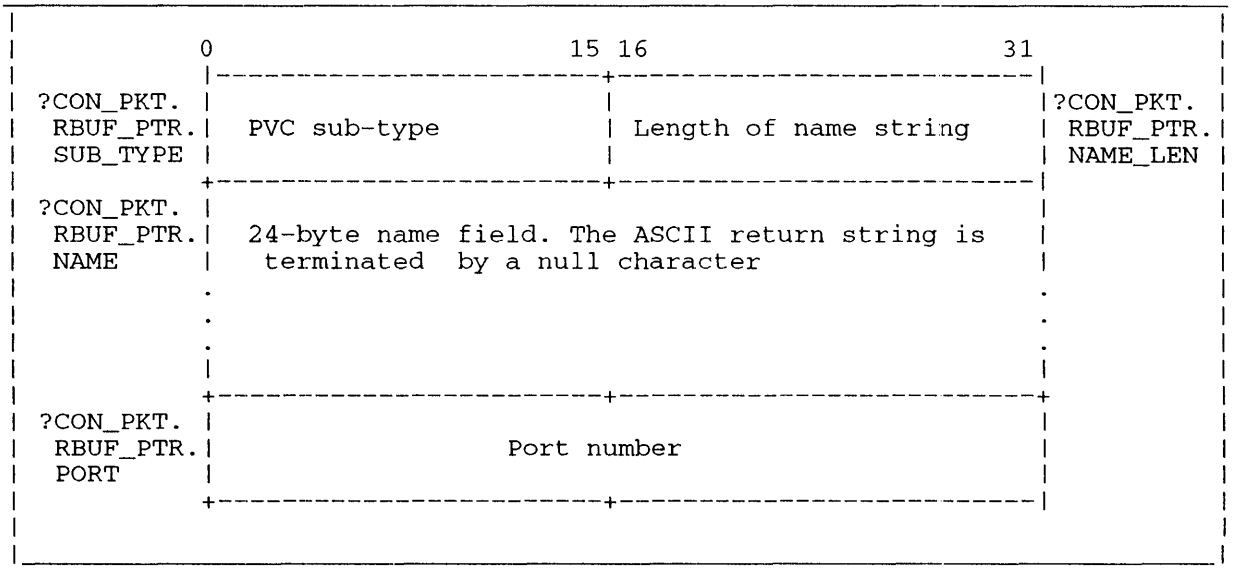


Figure 2–11.14 Structure of ?CON_PVC_NAME_PORT return packet

Table 2–8.17. Contents of ?CON_PVC_NAME_PORT Subpacket

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

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

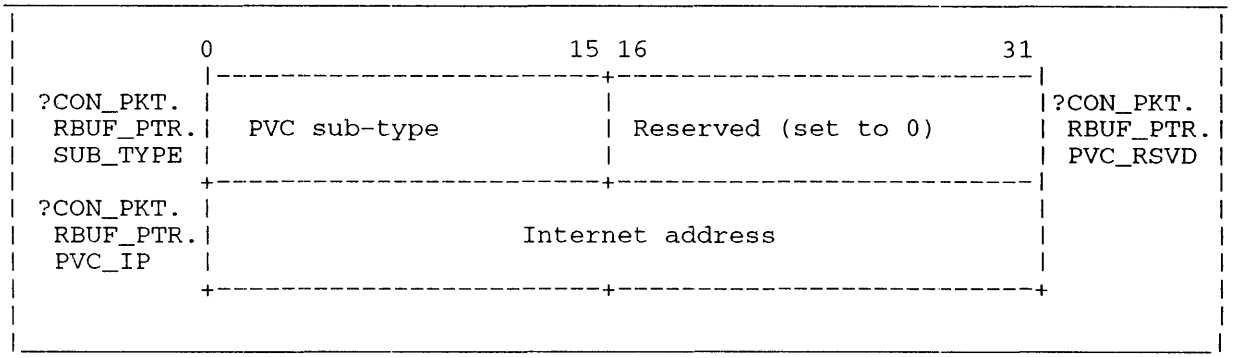


Figure 2–11.15. Structure of ?CON_PVC_IP return packet

Table 2–8.18. Contents of ?CON_PVC_IP Subpacket

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

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

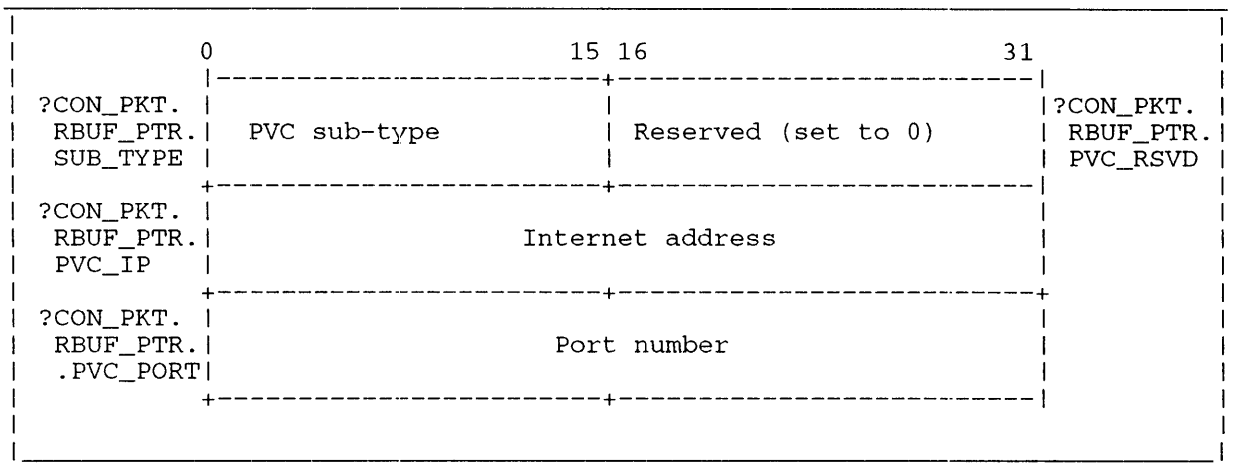


Figure 2–11.16 Structure of ?CON_PVC_IP_PORT return packet

Table 2–8.19. Contents of ?CON_PVC_IP_PORT Subpacket

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

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

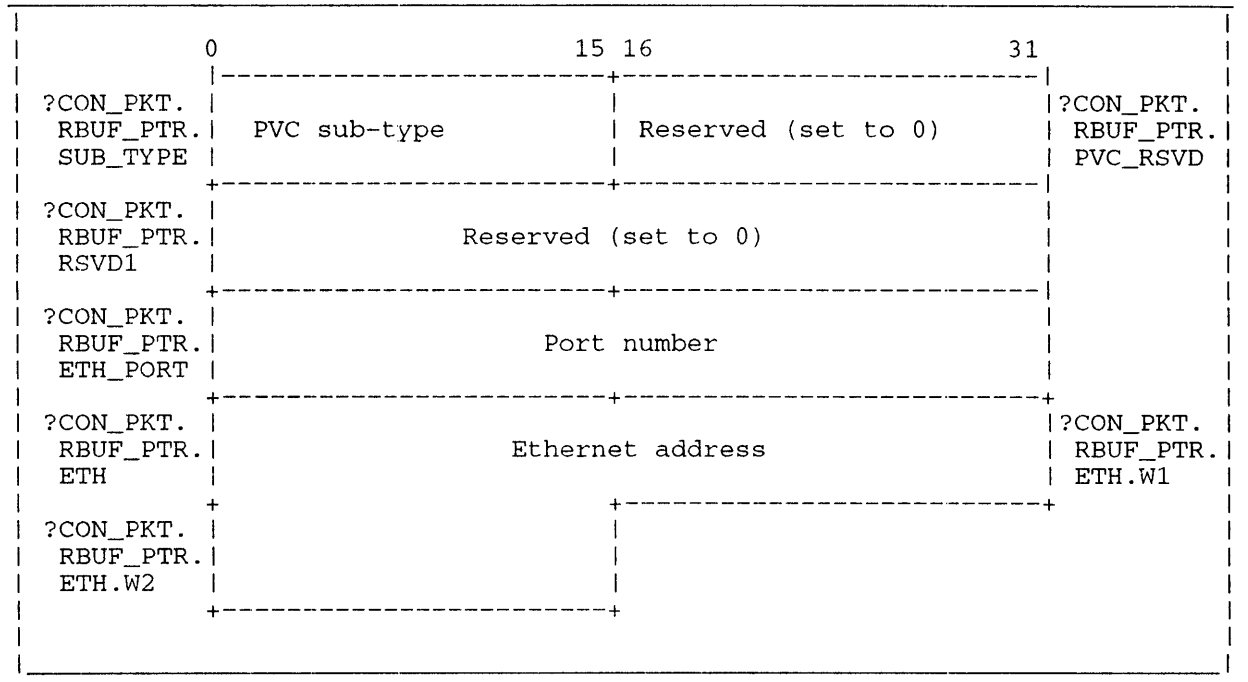


Figure 2–11.17 Structure of ?CON_PVC_ETH return packet

Table 2–8.20. Contents of ?CON_PVC_ETH Subpacket

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

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

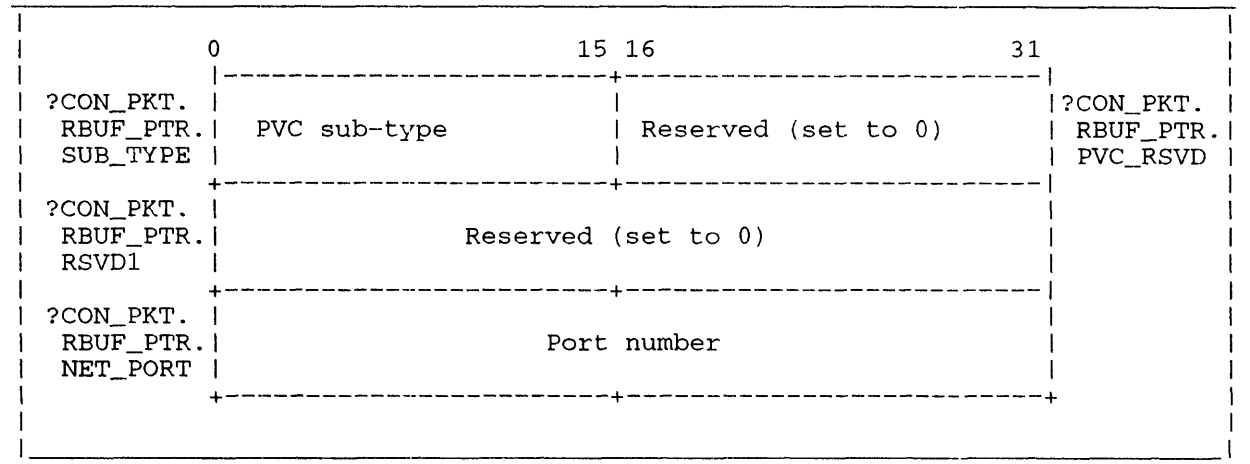


Figure 2–11.18 Structure of ?CON_PVC_PORT return packet

Table 2–8.21. Contents of ?CON_PVC_PORT Subpacket

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

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

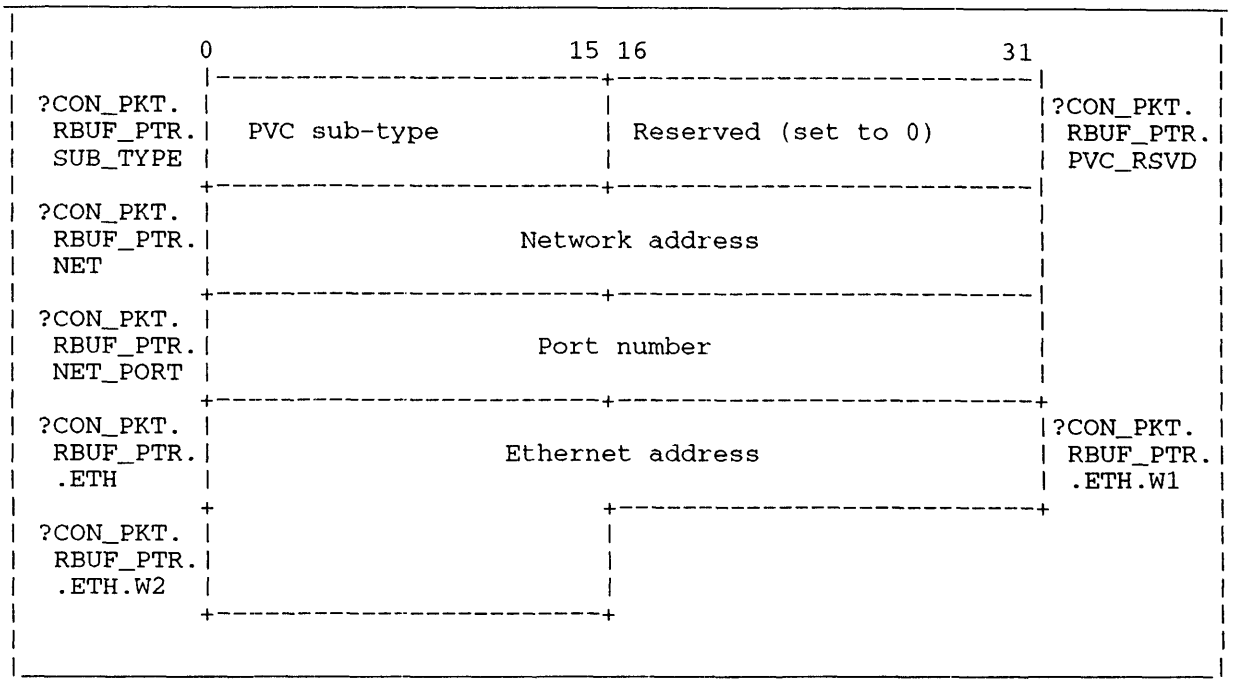


Figure 2–11.19 Structure of ?CON_PVC_NET return packet

Table 2–8.22. Contents of ?CON_PVC_NET Subpacket

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

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

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

Input

AC0 One of the following:

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

AC1 One of the following:

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

AC2 One of the following:

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

Output

AC0 Unchanged

AC1 Unchanged

AC2 Unchanged

Error Codes in AC0

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

Why Use It?

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

Who Can Use It?

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

What It Does

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

Number of Primary Elements = 1.

Index Element Size = 1.

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

  Primary Data Element Size = 1.

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

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

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

Input

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

Output

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

Error Codes in AC0

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

Why Use It?

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

Who Can Use It?

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

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

?DELAY
error return
normal return

Input

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

AC2 Reserved (Set to 0.)

Output

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

AC2 Undefined

Error Codes in AC0

No error codes are currently defined.

Why Use It?

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

Who Can Use It?

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

What It Does

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

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

Notes

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

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

Input

AC0 One of the following:

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

AC1 Reserved (Set to 0.)

AC2 One of the following:

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

Output

AC0 Unchanged

AC1 Undefined

AC2 Undefined or address of ?DELETE packet

Error Codes in AC0

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

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

ERFDE File does not exist

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

ERVBP Invalid byte pointer passed as a system call argument

ERWAD Write access denied

ERCDR Can't delete the root directory

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

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

Why Use It?

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

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

086-000195 updates
093-000542

?DIR
error return
normal return

Input

AC0 One of the following:

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

AC1 Reserved (Set to 0.)

AC2 Reserved (Set to 0.)

Output

AC0 Unchanged

AC1 Undefined

AC2 Undefined

Error Codes in AC0

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

Why Use It?

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

Who Can Use It?

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

What It Does

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

?DQTSK

Removes from the queue one or more previously queued tasks.

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

Input

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

Output

AC0	Undefined
AC1	Undefined
AC2	Address of the task definition packet

Error Codes in AC0

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

Why Use It?

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

Who Can Use It?

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

What It Does

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

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

?ESFF
error return
normal return

Input

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

Output

AC0	Unchanged
AC1	Unchanged
AC2	Undefined

Error Codes in AC0

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

Why Use It?

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

Who Can Use It?

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

What It Does

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

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

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

Notes

- See the description of ?UPDATE in this chapter.

AOS/VS

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

Input

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

Output

AC0	Undefined
AC1	Undefined
AC2	Address of the ?EXEC packet

Error Codes in AC0

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

Why Use It?

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

Who Can Use It?

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

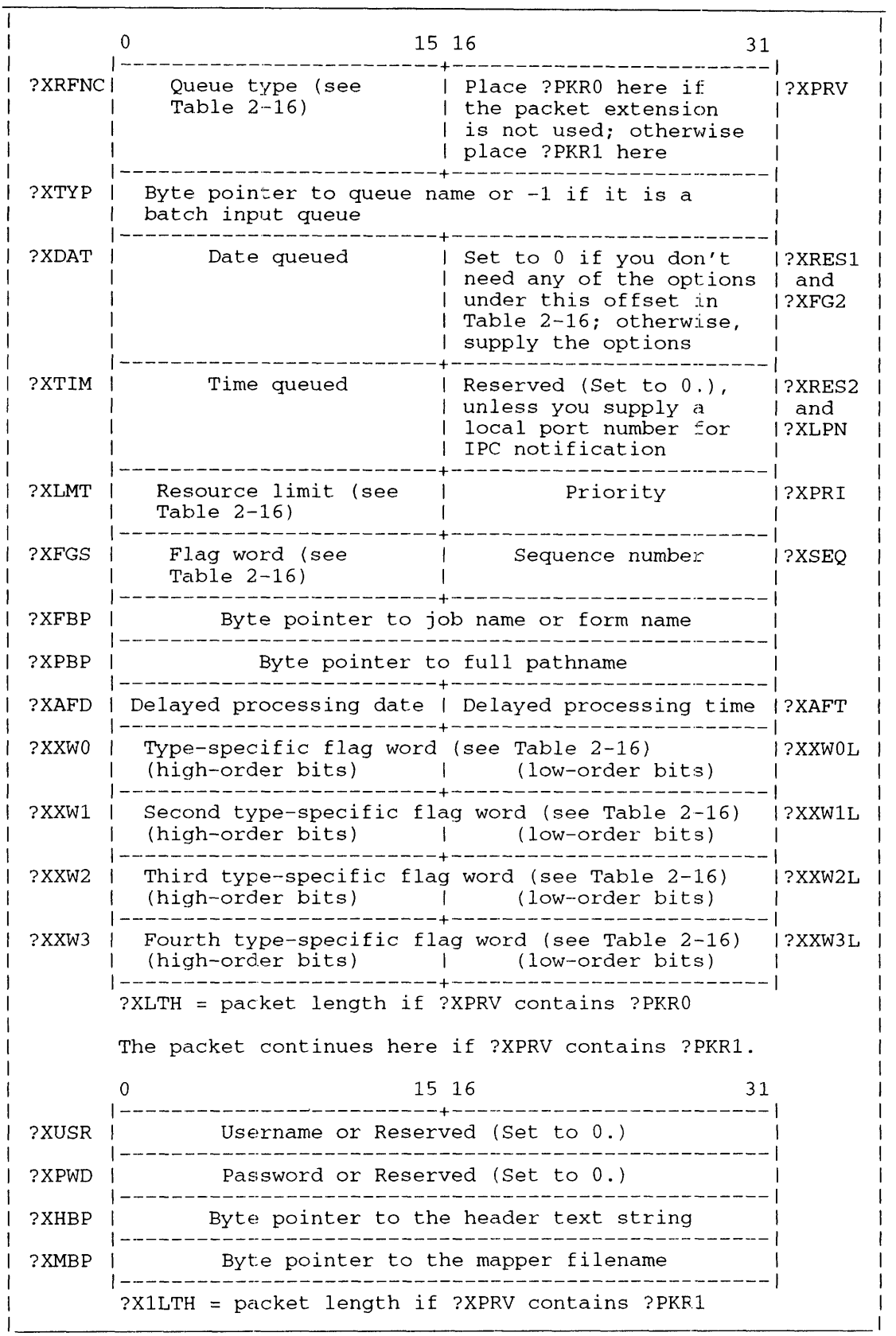


Figure 2–28. Structure of ?EXEC Packet for Queue Requests

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

?FLOCK — Locks an object.

AOS/VS

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

Input

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

Output

AC0	Unchanged
AC1	Unchanged
AC2	Address of the ?FLOCK packet

Error Codes in AC0

ERLCK	Object already locked
ERLNG	Lock not granted — pended request canceled
ERLLE	Lock limit exceeded — more than 1023 locks for one object
ERFNO	Channel not open
ERICN	Illegal channel number
ERVWP	Invalid address passed as system call argument
ER_FS_DIRECTORY_NOT_AVAILABLE	Directory not available because the LDU was force released (AOS/VS II only)

Why Use It?

Use this system call to restrict access to objects. You can use it to prevent cooperating processes from gaining access to the same object at the same time. An object is any entity whose nature requires processes to cooperate so they can properly gain access to it. For example, an object may be a component of a file. On the other hand, an object may be a critical shared code path. Cooperating processes must agree on the meaning of the object numbers.

To achieve the desired cooperation, all processes wanting to gain access to an object must first attempt to lock the object by issuing ?FLOCK. After the operating system grants a lock on an object to a process the process may then gain access to the object. Note that the operating system does not prevent access to a locked object — it's up to the cooperating processes to make sure they don't refer to an object until the operating system has granted them a lock on the object. When a process has completed its access to an object it should release its lock by issuing system call ?FUNLOCK.

?FLOCK offers three types of lock requests: shared, exclusive, and whole file. With shared locks several processes may use an object simultaneously (e.g., reading records), but no process may gain exclusive access (perhaps for writing). Whole file locking can be either shared or exclusive. With whole–file shared locking, only whole–file shared locks are subsequently permitted. With whole–file exclusive locking, no other locks are permitted on the same file or object.

Who Can Use It?

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

What It Does

?FLOCK locks an object for use by cooperating processes. An object is locked for EXCLUSIVE or SHARED use based on the type of lock you request. To lock an object, place the number that represents the object in offset ?FLSEL of the parameter packet.

Cooperating processes use a file to represent a group of related objects that the processes want to gain access to. Objects may represent components (such as records) of the file, but this is not necessary. The file can be empty and the objects may represent entities that are separate from the file. Your process must open the file before it can lock one of its objects. Place the channel number that the open request (i.e., the ?OPEN system call) returned, along with the type of lock, in the ?FLOCK parameter packet.

Cooperating processes must have access to the same file so they can use the locking mechanism that ?FLOCK provides. Note that the operating system does *not* prevent access to a locked object — it is the responsibility of the cooperating processes not to attempt access to an object until the operating system grants them a lock on the object. You can prevent noncooperating processes from gaining access to the objects by setting the file's access control list accordingly.

For more complex situations, we suggest using a SHARED PROTECTED file (see ?SOPPF). A global server process that first opens the file controls access to a shared protected file. To prevent unauthorized access to the file when the global server process is not running, set the file's access control list to null. The global server process must then turn on its Superuser privilege so it can open the file.

To gain access to objects that the file represents, cooperating processes can issue ?RINGLD against a local server routine that then becomes a customer (see ?CON) of the global server. Upon request from a local server the global server can grant permission (see ?PMTPF) for the local server to become a subsequent opener of the file. This permission supersedes the file's access control list. The local server can then issue ?SOPPF against the file, and next use the file to represent objects by issuing ?FLOCK on the channel number that ?SOPPF returns. If these objects represent components of the file's contents, the local server can read the file by issuing ?SPAGE to gain access to an object.

Figure 2–47 shows the structure of ?FLOCK's parameter packet, and Table 2–26 describes its contents.

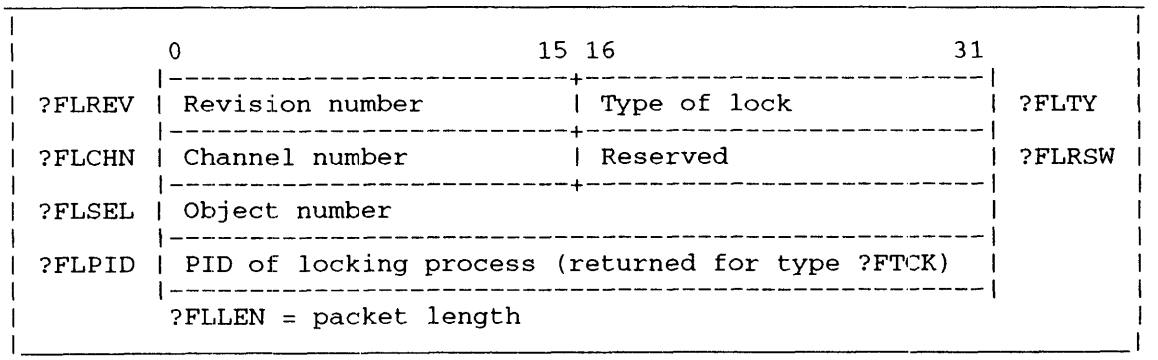


Figure 2–47. Structure of ?FLOCK Packet

Table 2–26. Contents of ?FLOCK Packet*

Offset	Contents
?FLREV	Packet revision number. Place ?PKR0 here.
?FLTY	Type of lock. It specifies the operation you want. Select from the following values. ?FTCK -- Check for a lock. If selected, AOS/VS does no locking but it returns the status of the object specified. If it is unlocked, then ?FLTY contains 0; if it is locked, then ?FLTY contains the lock type (?FTSH, ?FTEX, or ?FWFL). ?FTEX -- Create an exclusive lock. ?FTSH -- Create a shared lock. ?FWFL -- Lock the whole file. If you lock the whole file, you must also set either ?FTEX or ?FTSH. If you supplied ?FTEX, ?FTSH, or ?FWFL, then you should also specify one of the following pending actions. ?FTER -- Take an error return if AOS/VS is unable to lock the file element. ?FTPN -- Pend if not able to lock, and wait for an unlock to occur. AOS/VS doesn't return an error.
?FLCHN	Supply the channel number for the file.
?FLRSW	Reserved. (Set to 0.)
?FLSEL (double-word)	Object number. Your process and cooperating processes must agree on the meaning of this number.
?FLPID (double-word)	PID of the locking process. If the value of ?FTCK is set in offset ?FLTY, AOS/VS returns the PID. Otherwise, supply 0.

* There is no default unless otherwise specified.

?FLUSH — Flushes the contents of a shared page to disk.

?FLUSH
error return
normal return

Input

AC0	Any address in the page you want to flush to disk (The OS derives the logical page number by stripping off Bits 22 through 31.)
AC1	Reserved (Set to 0.)
AC2	Reserved (Set to 0.)

Output

AC0	Unchanged
AC1	Undefined
AC2	Undefined

Error Codes in AC0

ERNSA Shared I/O request not to shared area
ER_FS_DIRECTORY_NOT_AVAILABLE
Directory not available because the LDU was force released (AOS/VS II only)

Why Use It?

?FLUSH directs the operating system to write a shared page out to disk immediately. Therefore, ?FLUSH is useful if you have modified a shared page and you want to make sure that the operating system updates it immediately.

Note that before you can use ?FLUSH, you must use ?SOPEN to open the target file for shared access.

Who Can Use It?

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

What It Does

?FLUSH copies the contents of a shared page from memory to disk, without actually releasing the shared page from memory. The calling process regains control after the operating system has performed the flush.

Notes

- See the description of ?RPAGE and ?ESFF in this chapter.
- See the descriptions of ?SOPEN and ?UPDATE in this chapter for information on simulating ?FLUSH.

?FSTAT *[file status packet address]*
error return
normal return

Input

AC0	may contain one of the following: • Byte pointer to the pathname of the target file • Channel number of the target file
AC1	Flags: Bit 0 = 0 if AC0 contains a byte pointer Bit 0 = 1 if AC0 contains a channel number Bit 1 = 0 to resolve links in the pathname Bit 1 = 1 to ignore links in the pathname
AC2	Address of the file status packet, unless you specify the address as an argument to ?FSTAT

Output

AC0	Unchanged
AC1	Unchanged
AC2	Address of file status packet

Error Codes in AC0

ERCIU	Channel in use
ERFAD	File access denied
ERFNO	Channel not open
ERICN	Illegal channel
ERMPR	System call parameter address error
ERVBP	Invalid byte pointer passed as a system call argument
ERVWP	Invalid word pointer passed as a system call argument
ER_FS_DIRECTORY_NOT_AVAILABLE	Directory not available because the LDU was force released (AOS/VS II only)

Why Use It?

?FSTAT lets you determine the specifications set for a file or a directory when it was created. ?FSTAT also returns the date and time the file was last accessed or modified. (You may need this information before you open or reopen a file.)

Who Can Use It?

There are no special process privileges needed to issue this call. See the following section "What It Does" for an explanation of file access requirements.

What It Does

?FSTAT returns the status parameters of the target file. You can specify the packet address as an argument to ?FSTAT or you can load the address into AC2 before you issue ?FSTAT. Before the calling process can issue ?FSTAT, it must meet these rules:

- If it has no access rights to a file, it needs Read and Execute access to the file's directory.
- If it has any access right to a file, it needs Execute access to the file's directory.

Before you issue ?FSTAT, load AC0 with either the channel number of the target file or a byte pointer to its name.

If you specify a pathname that ends in a link and Bit 1 of AC1 equals 1, the operating system returns status information about the link without resolving it. If the pathname ends in a link and you do not set Bit 1 of AC1, the operating system resolves the link entry and returns status information about the file to which the link refers.

The format of the information ?FSTAT returns varies, depending on the type of file. The operating system returns different information packets for IPC files, directories, unit files, and other file types. Unit files are devices that have been opened as a unit (for example, MCAs, MTBs, disks, etc.). Figure 2–48, Figure 2–49, Figure 2–50, and Figure 2–51 show the structure of the various ?FSTAT packets.

?FTOD
error return
normal return

Input

AC0	Seconds from 0 through 59
AC1	Minutes from 0 through 59
AC2	Hour from 0 (midnight) through 23 (11 p.m.)

Output

AC0	Number of biseconds (seconds/2) since midnight
AC1	Unchanged
AC2	Unchanged

Error Codes in AC0

ERPRE	Invalid system call parameter

Why Use It?

You can use ?FTOD to obtain input parameters for the ?CREATE system call's time block.

Who Can Use It?

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

What It Does

?FTOD takes time in seconds, minutes, and hours and converts it to a scalar value; that is, the number of biseconds (half the number of seconds) since midnight.

Before you issue ?FTOD, load AC0, AC1, and AC2 with the appropriate octal values for seconds, minutes, and hours, respectively. The legal input parameters for ?FTOD range from 0:00:00 (midnight) to 23:59:59 (11:59 p.m.).

The operating system returns the scalar time value to AC0. If the scalar conversion results in an odd number of seconds, the operating system rounds the value to the next number before division; for example, 7 seconds would result in (8/2) = 4 biseconds.

AOS/VS

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

Input

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

Output

AC0	Unchanged
AC1	Unchanged
AC2	Address of the ?FUNLOCK packet

Error Codes in AC0

ERNLK Object not locked
ERVWP Invalid address passed as system call argument
ER_FS_DIRECTORY_NOT_AVAILABLE
Directory not available because the LDU was force released (AOS/VS II only)

Why Use It?

Use this system call to unlock an object when you no longer need it.

Who Can Use It?

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

What It Does

?FUNLOCK releases a lock that your process has placed.

?FUNLOCK lets other processes gain a lock on the same object. It can also unlock all locks that your process holds on the specified file. You must have issued ?OPEN for a file before you can unlock one of its objects. Place the channel number ?OPEN returns in the ?FUNLOCK packet.

Cooperating processes must have access to the same file in order to use the unlocking mechanism that ?FUNLOCK provides. The processes must also agree on the meaning of the object numbers so they can guarantee successful unlocking.

Figure 2–53 shows the structure of ?FUNLOCK's parameter packet, and Table 2–28 describes its contents.

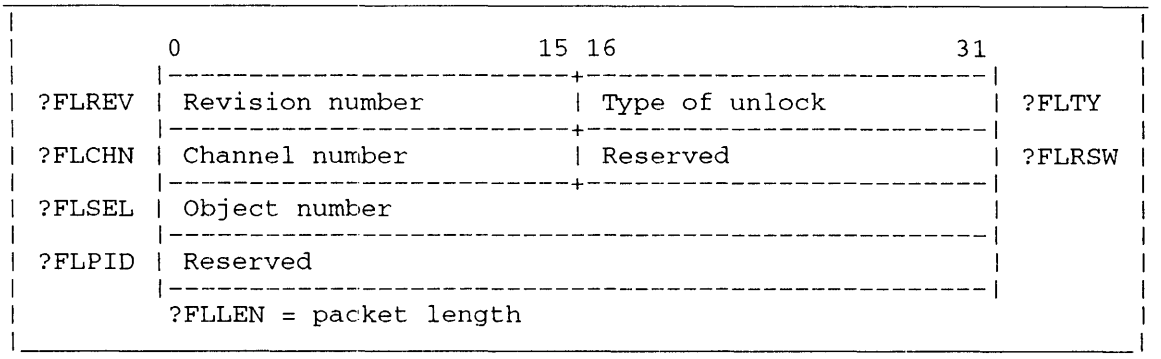


Figure 2–53. Structure of ?FUNLOCK Packet

Table 2–28. Contents of ?FUNLOCK Packet*

Offset	Contents
?FLREV	Packet revision number. Place ?PKR0 here.
?FLTY	Type of unlock. It specifies the operation you want. Select from the following values. ?FULA -- Unlocks all locks that the process holds on this channel. If you don't select ?FULA, AOS/VS unlocks only the lock specified by offset ?FLSEL. ?FWFL -- Unlocks the whole file.
?FLCHN	Supply the channel number for the file.
?FLRSW	Reserved. (Set to 0.)
?FLSEL (double-word)	Object number. Your process and cooperating processes agree on the meaning of this number.
?FLPID (double-word)	Reserved. (Set to 0.) ?FLOCK uses this offset.

* There is no default unless otherwise specified.

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

Operating System Differences

AOS/RT32 always returns "+,OWARE" as the ACL.

Input

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

Output

AC0	Unchanged
AC1	Unchanged
AC2	Unchanged

Error Codes in AC0

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

Why Use It?

Because ?GACL returns a file or directory's current ACL, you can use it along with ?SACL, which sets the current ACL, or with ?DACL, which sets the default ACL.

Who Can Use It?

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

What It Does

?GACL returns the access control list (ACL) associated with a file or directory. AOS/RT32 always returns "+,OWARE" as the ACL.

?GLIST
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Byte pointer to a receive buffer for the search list
AC2	Byte length of the receive buffer

Output

AC0	Undefined
AC1	Unchanged
AC2	Unchanged

Error Codes in AC0

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

Why Use It?

?GLIST lets you examine the current contents of your search list. Thus, you can use ?GLIST as a preliminary system call to ?SLIST, which changes the contents of the search list.

Who Can Use It?

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

What It Does

?GLIST returns a copy of the calling process's search list to the buffer that you specify in AC1.

Before you issue ?GLIST, load AC1 with a byte pointer to the receive buffer and load AC2 with the length of the buffer. (The symbol ?MXPL represents the maximum length of the search list buffer.) Then, after you issue ?GLIST, the operating system returns the search list to the buffer in the following format:

pathname<0>...pathname<0><0>

where

pathname is a complete pathname string.

<0> is the null terminator.

Be sure to allocate enough buffer space to accommodate the two trailing nulls that the operating system appends to the search list.

Notes

- See the description of ?SLIST in this chapter.

?GMEM Returns the number of undedicated memory pages.

?GMEM
error return
normal return

Input

None

Output

AC0 Undefined

AC1 Current number of undedicated memory pages

AC2 Undefined

Error Codes in AC0

No error codes are currently defined.

Why Use It?

?GMEM gives you information about overall memory contention. In general, the smaller the value returned by ?GMEM, the more likely memory contention becomes. Issuing ?GMEM before you wire pages to a resident process's working set helps you to correctly adjust the size of the resident process to your memory configuration. Do not wire more pages than the current number of undedicated memory pages, because it can cause a system deadlock.

Who Can Use It?

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

What It Does

?GMEM returns the current number of undedicated pages available to the calling process. Undedicated pages are unused pages that the operating system can allocate to the calling process.

Notes

- See the description of ?WIRE in this chapter.

?GNAME
error return
normal return

Input

AC0	Byte pointer to a filename or pathname fragment
AC1	Byte pointer to a receive buffer for the pathname
AC2	Byte length of the receive buffer

Output

AC0	Unchanged
AC1	Unchanged
AC2	Byte length of the complete pathname, excluding the null terminator

Error Codes in AC0

ERFAD	File access denied
ERFDE	File does not exist
ERIRB	Insufficient room in buffer
ERMPR	System call parameter address error
ERVBP	Invalid byte pointer passed as a system call argument

Why Use It?

?GNAME can be useful as a preliminary system call for system calls that require a complete pathname as input.

Who Can Use It?

Required file access privileges depend on whether your program runs under the old file system (before AOS/VS II Release 1.00 and with any revision of AOS/RT32 or of AOS/VS) or under the new file system (effective with AOS/VS II Release 1.00).

Old File System

There are no special process privileges needed to issue this call. The calling process must have access (of any type) to each file in the pathname fragment or have Read access to their parent directories. The calling process must also have Execute access to the files' parent directories. However, the operating system does not check for access privileges if the pathname fragment consists of only a prefix (for example, =).

New File System

There are no special process privileges needed to issue this call. The calling process must have access (of any type) to each file in the pathname fragment or have Read access to their parent directories. The calling process must also have Execute access to the files' parent directories. However, the operating system does check for access privileges in all cases, including when the pathname fragment consists of only a prefix (for example, =). In this case, the calling process still must have access (of any type) to each file in the pathname fragment or have Read access to their parent directories.

?GNAME Continued

An Important Difference

One important difference between issuing ?GNAME under the old and new file systems occurs in the following sequence of events:

- The CLI issues ?PROC to create a new child CLI process.
- The newly created child CLI process issues ?SUSER and ?SUPROC to explicitly disable Superuser and Superprocess mode in case the parent process has either privilege.
- The child CLI process issues ?GNAME against "=" to determine the full pathname of the initial directory. Furthermore, suppose the child process does not have access to its working directory, "=".
  - Under the old file system ?GNAME succeeds because AOS/VS performs no access checking; however, the child CLI process has no access to its initial directory. If this process changes its working directory and tries to reenter the initial directory (via, say, a **DIR/I** command), ?GNAME returns the error ERFAD.
  - Under the new file system ?GNAME fails right away with error ERFAD. The child CLI process terminates.

What It Does

?GNAME returns the complete pathname, starting with the root, when you specify a pathname fragment in AC0. ?GNAME functions with either an open or a closed file. If you load AC0 with the byte pointer to the prefix =, the operating system returns the pathname of the current working directory.

You cannot use ?GNAME to obtain the "true" pathname associated with a generic file. For example, ?GNAME would return a complete pathname of :PER:DATA for the input pathname @DATA, even though the "true" complete pathname is really :UDD:USER:DATA. Therefore, to obtain the "true" pathname, use ?GRNAME.

Notes

- See the description of ?GRNAME in this chapter.

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

Input

AC0	Reserved (Set to 0.)
AC1	Channel number of the target directory
AC2	Address of the ?GNFN packet, unless you specify the address as an argument to ?GNFN

Output

AC0	Undefined
AC1	Unchanged
AC2	Address of the ?GNFN packet

Error Codes in AC0

ERCIU	Channel in use
EREOF	End of file (There are no more files in the target directory.)
ERFNO	Channel not open
ERFTL	Filename too long (The template exceeds 63 characters.)
ERIFC	Illegal filename character
ERNDR	Not a directory (The channel that you specified is not opened on a directory.)
ERVBP	Invalid byte pointer passed as a system call argument
ERVWP	Invalid word pointer passed as a system call argument
ER_FS_DIRECTORY_NOT_AVAILABLE	Directory not available because the LDU was force released (AOS/VS II only)

Why Use It?

?GNFN allows you to obtain a specific directory's filenames. The CLI, for example, uses ?GNFN to implement its FILESTATUS command.

Who Can Use It?

There are no special process privileges needed to issue this call. However, your process must be able to open (via ?OPEN or ?GOPEN) the directory. It must also have Read and Execute access to the directory at the time it issues ?OPEN or ?GOPEN.

What It Does

?GNFN returns a filename in the target directory (or all filenames that match the template that you specify) to the buffer that you specify in the ?GNFN packet. The first time you issue ?GNFN, the operating system returns the name of the first file in the directory; the second time, it returns the second filename, and so on.

To get all the filenames, use the + template. You can also obtain every filename by issuing repeated ?GNFN system calls until ?GNFN fails on error code EREOF. (EREOF is an end–of–file condition that means you have reached the last file entry in the directory.)

?GNFN Continued

Before you issue ?GNFN, you must open the target directory. After you open the directory with ?OPEN (or ?GOPEN), load AC1 with the correct channel number, which returns in the ?OPEN (or ?GOPEN) packet. You can specify the packet address as an argument to ?GNFN, or you can load it into AC2 before you issue ?GNFN. Figure 2–55 shows the structure of the ?GNFN packet, and Table 2–34 describes its contents.

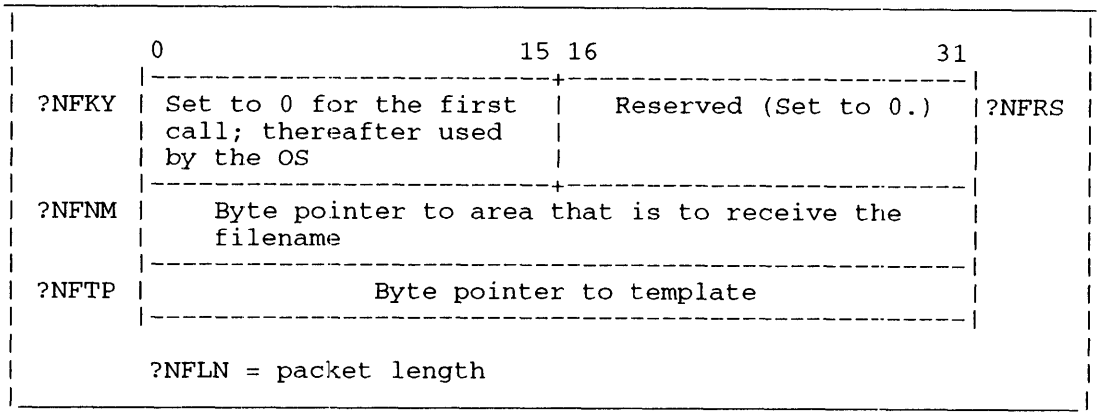


Figure 2–55. Structure of ?GNFN Packet

Table 2–34. Contents of ?GNFN Packet*

Offset	Contents
?NFKY	OS internal value. Set to 0 the first time you issue ?GNFN; ignore thereafter. This value is a key to where the list should continue. So do not set ?NFKY to zero between calls.
?NFRS	Reserved. (Set to 0.)
?NFNM (doubleword)	Byte pointer to area that is to receive the filename.
?NFTP (doubleword)	Byte pointer to filename template, if present. DEFAULT = -1 (no template). If you omit the template, the operating system searches only the open directory. Table 2-35 shows the template characters.

* There is no default unless otherwise specified.

Table 2–35. Filename Template Characters

Character	Description
* (asterisk)	Matches any single filename character except a period.
- (hyphen)	Matches any series of characters not containing a period.
+ (plus sign)	Matches any series of characters.

The first time you issue ?GNFN, set the contents of offset ?NFKY to 0. Do not modify the contents of ?NFKY on subsequent system calls, because the operating system uses this offset as an internal pointer. There is no default value for offset ?NFNM. Therefore, you must use offset ?NFNM as a byte pointer to a receiving area for the filename. If you do not want to use a template, choose the default value (–1) of offset ?NFTP.

Sample Packet

```
PKT:    .BLK    ?NFLN           ;Allocate enough space for the packet
                                ;(packet length is ?NFLN).
        .LOC    PKT+?NFKY       ;Set to 0 for first call. The OS
        .WORD   0               ;uses this value for second and
                                ;subsequent calls.
        .LOC    PKT+?NFNM       ;Byte pointer to an area to receive
                                ;the filename.
        .DWORD  BUFF*2          ;Byte pointer to BUFF.
        .LOC    PKT+?NFTP       ;Byte pointer to template.
        .DWORD  -1              ;No template (default is -1).
        .LOC    PKT+?NFLN       ;End of packet.
```

Notes

- See the description of ?OPEN in this chapter.

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

Operating System Differences

Only AOS/VS II supports ?ODTL in offset ?ODF1 of the packet extension.

Input

AC0	Byte pointer to the target file's pathname
AC1	Target file's channel number, or −1, to direct the OS to assign a channel number
AC2	Address of the ?GOPEN packet, unless you specify the address as an argument to ?GOPEN

Output

AC0	Contains the following: • Bits 0 through 25 are reserved (Set to 0.) • Bits 26 through 31 contain one or more of the following file access privilege masks: ?FACO Owner access ?FACW Write access ?FACA Append access ?FACR Read access ?FACE Execute access
AC1	Unchanged
AC2	Address of the ?GOPEN packet

Error Codes in AC0

ERDMO Disk marked as owned by another system
ERDRS Device reserved by another port
ERFTM File/tape mismatch
ERITD Indecipherable tape density
ERVBP Invalid byte pointer passed as a system call argument
ERVWP Invalid word pointer passed as a system call argument
ER_FS_NO_TLA_PRIVILEGE
Attempt to issue ?GOPEN with ?ODTL value supplied, but without proper privilege
ER_FS_OBSOLETE_IPC_FILE_DETECTED
Obsolete IPC file type has been detected; file has been deleted (AOS/VS II only)

Why Use It?

You must use ?GOPEN to open a file for block I/O or physical block I/O. Also, you must issue ?GOPEN to open an MCA as a unit or to open a specific link. Then, if you use the CLI MOUNT command to mount a tape, you must ?GOPEN the file using its link name.

Who Can Use It?

To issue this call, you need a special privilege for only one situation. Your process must be running under the new file system (AOS/VS II) and in Superuser mode in order to issue ?GOPEN with value ?ODTL supplied in offset ?ODF1 of the packet extension.

You must have Read and Execute access to the file's parent directory. In addition, you must have Read access to the file to read from it and Write access to the file to write to it.

What It Does

?GOPEN opens a file for block I/O and assigns the file a channel number. You can specify the channel number in AC1 or force the operating system to assign the channel number by loading AC1 with –1. You can open any of the following devices for block I/O: disk files and units, tape file, MTB tape units, MTD tape units, and MCA units.

The ?GOPEN packets are different for IPC files (see Figure 2–56 and Table 2–36) and for all other file types (see Figure 2–57 and Table 2–38). Although you must reserve sufficient space in your logical address space for the packets, you need to provide input specifications for offset ?OPFL (for the standard packet) or offset ?OPCH (for the IPC packet) only.

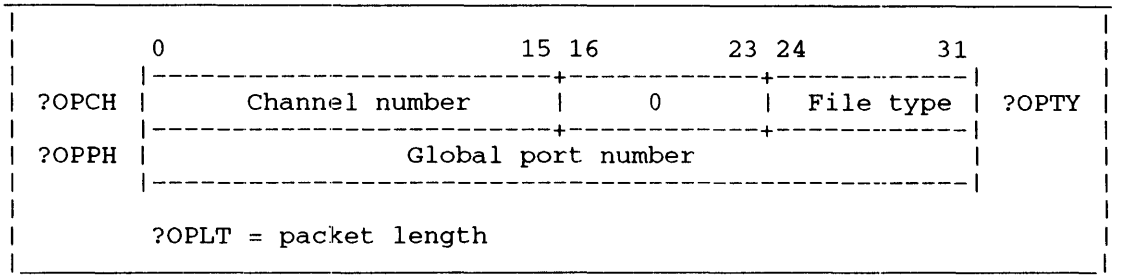


Figure 2-56. Structure of ?GOPEN Packet for IPC Files

Table 2–36. Contents of ?GOPEN Packet for IPC Files*

Offset	Input Value	Output Value
?OPCH	None.	Channel number.
?OPTY	None.	Left byte: 0 Right byte: File entry type (?FIPC)
?OPPH (doubleword)	None.	Global port number.

* There is no default unless otherwise specified.

Table 2–37. Contents of Standard ?GOPEN Packet*

Offset	Input Value	Output Value
?OPFL	Flag bits ?OPME--Exclusive open. ?OPMD--Inhibit initial form feed. ?OPXP--Packet has extension. If tape density mode, specify one of the following: ?OPDL--800 bpi. ?OPDM--1600 bpi. ?OPDH--6250 bpi. ?OPAM--automatic density mode matching selected ?OPD5--Low density ?OPD6--Medium density ?OPD7--High density Transfer mode (for Model 6352 magnetic tape units only): ?OPMBF--Use buffered mode when performing tape I/O ?OPMST--Use streaming mode when performing tape I/O DEFAULT--0 for non-buffered and non-streaming data transfer	Bits 0-15: Channel number.
?OPTY	None.	Left byte: Record format Right byte: File entry type
?OPFC	None.	File control parameters. See the description of offset ?SCPS in the explanation of ?FSTAT.
?OPEW	None. (Set to 0.)	Unchanged.
?OPEH (doubleword)	None	File size in bytes.

* There is no default unless otherwise specified.

The operating system returns all the other parameters, based on the specifications you set when you created the file. Although the operating system returns file type and format information, the block I/O and physical block I/O system calls ignore record formatting information, because they operate in terms of blocks only.

Notice that offset ?OPFL/?OPCH in both packets contains different input and output values. On input, you can use ?OPFL/?OPCH to indicate a tape–density mode if you are opening a magnetic tape file on a type MTB or MTD controller. On output, ?OPFL/?OPCH contains the target file's channel number.

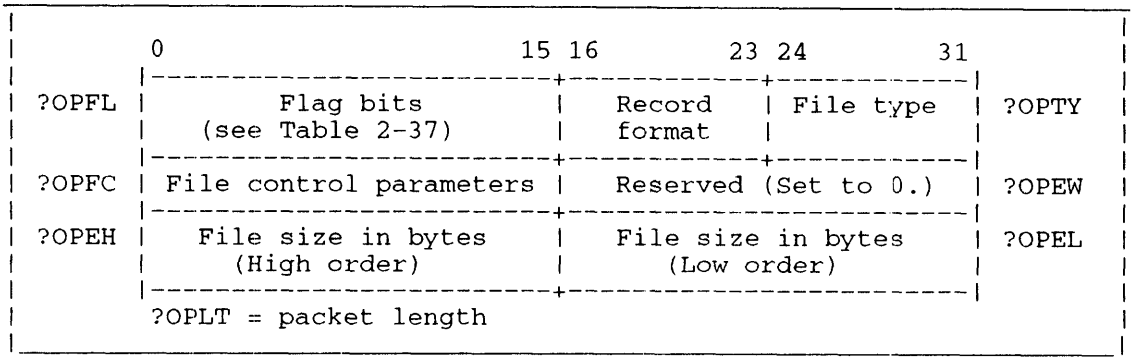


Figure 2–57. Structure of Standard ?GOPEN Packet

Opening System Areas

You can issue ?GOPEN against a system area. To do this, simply place a byte pointer to the desired unit name and system area number in AC0. An example of a system area name is @DPJ0:1082. You must have previously created, via system utility Disk Jockey, the system area. You can open a system area only under the New File System.

Issuing ?GOPEN against a system area is much the same as issuing ?GOPEN against a physical disk alone, but in the first case you can gain access to only part of the disk. Bad block remapping does not occur for system areas that you issue ?GOPEN against.

?GOPEN Options

To open the file to the calling process exclusively, supply value ?OPME in offsets ?OPFL/?OPCH at input. Note that this option is available only within the standard packet (for non–IPC files), because it makes no sense to exclusively open an IPC file.

If you are reading or writing a magnetic tape file on an MTB or MTD tape controller, you can select a density mode for the tape controller by supplying one of the density–mode values in offsets ?OPFL/?OPCH.

Supply ?OPDL to specify a density of 800 bpi (bytes per inch); supply ?OPDM to set the density to 1600 bpi; supply ?OPDH to specify 6250 bpi; or supply ?OPAM if you want the operating system to select the density mode automatically.

Supply ?OPD5, ?OPD6, and ?OPD7 for low, medium, or high densities respectively. When you specify low or high density, the OS selects the lowest or highest density supported by the drive. When you select medium density: if the drive supports three levels, the OS selects the middle density value; when the drive supports two densities, the OS selects the lower density.

If you have issued ?GOPEN with value ?ODTL supplied, you cannot issue the following system calls to the file:

- ?ALLOCATE
- ?BLKIO (write block only)
- ?CPMAX
- ?CRUDA
- ?DELETE
- ?ESFF
- ?GTRUNCATE
- ?PWRB
- ?SACL
- ?UPDATE
- ?WRB
- ?WRUDA

An attempt to issue one of these system calls with value ?ODTL supplied in the ?GOPEN packet results in error code ER_FS_TLA_MODIFY_VIOLATION.

When you default the density–mode parameter (that is, omit the values), the operating system uses the density mode you chose for the MTB or MTD controller during the system–generation procedure. Make sure the default density mode matches that of your tape; otherwise, the operating system fails on error code ERFTM (file tape density mismatch). However, density matching always occurs for reads on MTD drives unless you specify a density.

If you select ?OPAM (automatic density selection) and the tape unit is damaged or the operating system cannot set the density, ?GOPEN fails on error code ERITD (tape density indecipherable).

In buffered mode (Model 6352 magnetic tape units only) the tape controller indicates that a tape transfer is complete after data has been read from memory, but before it has been written to tape. In this mode the system might not report error conditions for a request that fails. Therefore your program must check for errors whenever it issues a ?GCLOSE call. (The system reports all errors when it writes a file mark. Since ?GCLOSE writes a file mark, any program using buffered mode must explicitly close each tape file.)

The streaming mode (Model 6352 magnetic tape units only) allows your program to open a tape unit for high–speed backup purposes. In this mode the tape controller expects data transfer with the tape to occur quite rapidly. If data is not transferred fast enough, the performance of the tape unit degrades. So, a program that cannot maintain a high data transfer rate should not select this mode.

?GOPEN Extension

The ?GOPEN extension is used to allow access to options that exist for some Data General disk subsystems. These options (described below) may not be supported by all disk subsystems. An error will be returned (typically ERIOD or ERMNS) if the option selected is not available for the specified device.

The extension allows access to the following hardware features:

- Format selection for 4514 (DPM) disks.
- Modified sector I/O for DPJ disks (only large capacity models).
- Trespass option for dual ported disks.
- Model option for some MTJ tapes (Model 6352 only).
- Tape drives supporting data compression and the SCSI–2 protocol.

Supply value ?ODTL in offset ?ODF1 of the packet extension when you do not want the operating system to change the file's date/time last accessed values. These values are offsets ?STAH and ?STAL of the parameter packet for system call ?FSTAT and are also the values that the /TLA switch of the CLI FILESTATUS command returns. This change of date/time values occurs by default (i.e., ?ODTL not supplied) when your process later reads or closes the file; it will not occur if you have supplied ?ODTL.

Data compression is on (the default condition) for tape drives supporting compression. You must switch it off if you do not want to use it. Set ?ODCOF to 1 in offset ?ODF1 of the packet extension when you want to turn compression off and operate the drive in native mode.

To use the extension, you must set the extension bit (?OPXP) in the flag word (?OPFL) of the ?GOPEN packet. Figure 2–58 shows the structure of the ?GOPEN packet extension and Table 2–38 contains the option flags for offset ?ODF1 of the ?GOPEN packet extension. The number of words in the combined ?GOPEN main packet and its extension is ?OPLT + ?OPXL.

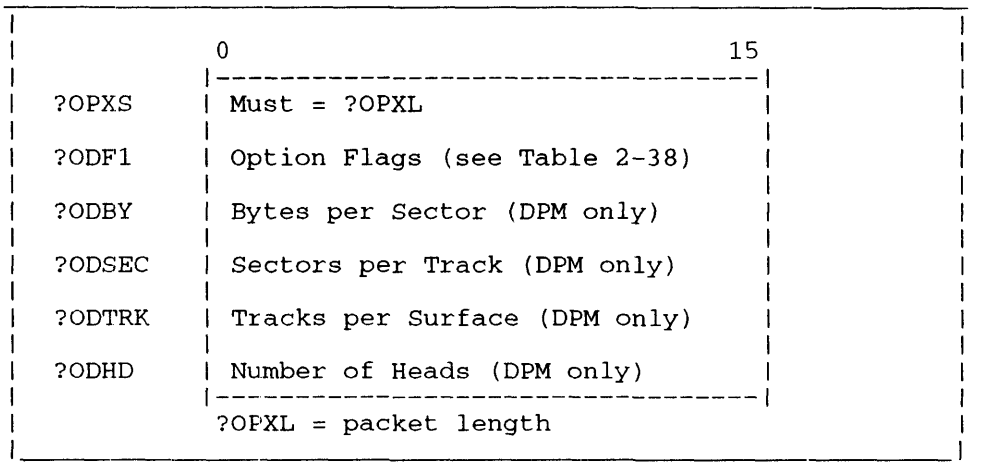


Figure 2–58. Structure of ?GOPEN Packet Extension

Table 2–38. Option Flags for Offset ?ODF1

Flag	Description
?ODND	Write without Modified bits (DPJ only)
?ODMB	Allow Modified Sector I/O (DPJ only)
?ODHS	Swap Head Option (DPM only)
?ODBS	Swap Byte Option (DPM only)
?ODP0	First Physical Sector is 0 (DPM only) (Used for Enterprise/MPT format only)
?ODST	Choose streaming mode for tape (MTJ only)
?ODTEO	9-track tape emulation override
?ODCOF	Data compression off If ?ODCOF = 0 the tape operates in data compression mode. If ?ODCOF = 1, then data compression is turned off, and the drive operates in native mode.
?ODTP	Old file system -- trespass if reserved by another port (DPJ only); new file system -- trespass if reserved by another port or if marked as owned by another system
?ODTL	Do not change the time last accessed value when the file is read from or closed.

Table 2–39 contains the specific format selectors that Figure 2–58 introduces.

Table 2–39. Valid Format Options (DPM disks)

Offset	8 Sector Format	9 Sector Format (Standard AOS)	10 Sector Format (Enterprise/MPT)
?OPXS	?OPXL	?OPXL	?OPXL
?ODF1	0	0	?ODHS+?ODBS+?ODP0
?ODBY	512.	512.	512.
?ODSEC	8.	9.	10.
?ODTRK	40.	40.	35.
?ODHD	1 or 2	1 or 2	2

In order to perform modified sector I/O (DPJ disks) using the ?BLKIO call, bit ?ODMB must be set. If this bit is not set, an error (ERCPO) will be returned. Use the ?ODND option to clear modified bits in conjunction with the ?BLKIO call.

Sample Packet

The following sample packet shows the standard ?GOPEN packet:

```
PKT:    .BLK    ?OPLT           ;Allocate enough space for the
                                ;packet.  Packet length = ?OPLT
        .LOC    PKT+?OPFL       ;Channel number.
        .WORD   0               ;The OS returns this value.
        .LOC    PKT+?OPTY       ;Record format and file type.
        .WORD   0               ;The OS returns this value.
        .LOC    PKT+?OPFC       ;Record length (if fixed format).
        .WORD   0               ;The OS returns this value.
        .LOC    PKT+?OPEW       ;Reserved.
        .DWORD  0               ;You must set this value to 0.
        .LOC    PKT+?OPEH       ;File size (in bytes).
        .DWORD  0               ;The OS returns this value.
        .LOC    PKT+?OPLT       ;End of packet.
```

Notes

- See the description of ?GCLOSE in this chapter.
- Refer to the chapter about VSGEN in the manual *Installing, Starting, and Stopping AOS/VS II* for information on the system--generation procedure.

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

Input

AC0	Reserved (Set to 0.)
AC1	Channel number of the disk file
AC2	Address of the ?GTRUNCATE packet, unless you specify the address as an argument to ?GTRUNCATE.

Output

AC0	Undefined
AC1	Unchanged
AC2	Address of the ?GTRUNCATE packet

Error Codes in AC0

EREOF End of file (You tried to set the EOF (end–of–file) past the current EOF.)
ERFNO Channel not open
ERMUS Multiple users of file; cannot truncate
ERSHR Shared file; cannot truncate
ERSIM Simultaneous requests on same channel
ERVWP Invalid word pointer passed as a system call argument
ER_FS_DIRECTORY_NOT_AVAILABLE
Directory not available because the LDU was force released (AOS/VS II only)
ER_FS_TLA_MODIFY_VIOLATION
Attempt to modify an AOS/VS II file with ?ODTL value supplied in ?GOPEN packet

Why Use It?

You can use ?GTRUNCATE to reduce the size of a disk file that is currently open for block I/O. Thus, ?GTRUNCATE allows you to reclaim disk space that you no longer need.

Who Can Use It?

There are no special process privileges needed to issue this call. You must have access to the file's channel number (via ?OPEN or ?GOPEN) and you must have Write access to the file.

What It Does

?GTRUNCATE truncates (shortens) disk files. The caller passes a channel number and a pointer to a new ?GTRUNCATE packet. The packet contains the new size of the file in bytes. This new size must be less than or equal to the current size of the file or the operating system returns error code EREOF (end of file).

If you need to truncate a magnetic tape file, use ?TRUNCATE.

The following restrictions apply to files that you want to truncate:

- The file must have a use count of +1.

?GTRUNCATE Continued

- The file must not be open for shared I/O.
- The file should have been opened with ?GOPEN instead of ?OPEN.

Figure 2–75 shows the structure of the ?GTRUNCATE packet, and Table 2–55 describes its contents.

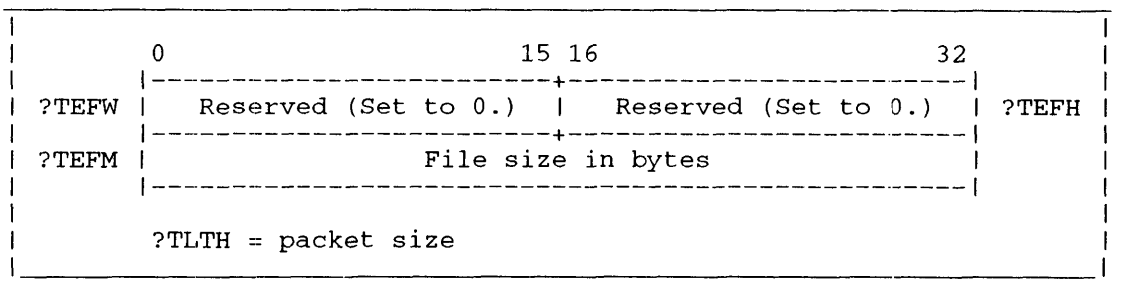


Figure 2–75. Structure of ?GTRUNCATE Packet

Table 2–55. Contents of ?GTRUNCATE Packet*

Offset	Contents
?TEFW	Reserved. (Set to 0.)
?TEFH	Reserved. (Set to 0.)
?TEFM (doubleword)	New file size in bytes. The new file size must be less than or equal to the current (old) file size.

* There is no default unless otherwise specified.

When you use ?GTRUNCATE, be sure to reset the values of offset ?PRNH and ?PRCL in the block I/O packet.

Notes

- See the descriptions of ?OPEN and ?GOPEN in this chapter.
- See the descriptions of ?PRDB/?PWRB and ?BLKIO in this chapter for information on the block I/O packet.

?IDEF
error return
normal return

Input

AC0 Contains the following:

- Bit 1 is a flag bit:

  Bit 1 = 0 if AC2 specifies the number of data channel map slots needed (no map definition table)

  Bit 1 = 1 if AC2 points to a data channel map definition table

- Bit 2 is a flag bit for 16 bit processes:

  Bit 2 = 1 if you want to use the extended packet

- Bits 3 through 31 contain the device code for the user–defined device in the range from 1 through 191 (= 277 octal)

AC1 Bits 1 through 31 contain the address of the device's DCT; set Bit 0 to 1 if the device is a data channel (DCH) or burst multiplexor channel (BMC) device

AC2 One of the following:

- Address of the map definition table
- Number of map slots needed (no map definition table) (each map slot accesses 1K words)

Output

AC0 Unchanged

AC1 Unchanged

AC2 Unchanged

If your program executes as a 16–bit process, then the bit numbers in AC0 and AC1 change from above as follows:

Bit 0	would read Bit 16
Bit 1	would read Bit 17
Bits 1 through 31	would read Bits 17 through 31
Bits 2 through 31	would read Bits 18 through 31

Error Codes in AC0

ERDCH	Data channel map full
ERDNM	Illegal device code (The device code is outside the legal range (1 through 191).)
ERIBS	Device already in use
ERPRE	Invalid system call parameter
ERPRV	Caller is not privileged for this action
ERPTY	Illegal process type

Why Use It?

?IDEF lets you establish an interface between the operating system and a device it does not support. ?IDEF and the other user–device system calls are particularly useful if you have applications–specific peripheral devices for which you have written special device–driver routines.

Who Can Use It?

The caller must be a resident process and must have privilege ?PVDV to use ?IDEF. There are no restrictions concerning file access.

What It Does

?IDEF defines a user device and its device control table (DCT). The operating system builds an internal DCT based on your DCT specifications, and enters this into its interrupt vector table.

Before you issue ?IDEF, set up a device control table in your logical address space and load its address into Bits 1 through 31 of AC1. (See Figure 2–77 for the DCT format if you want to issue ?IDEF from a 32–bit process. See Figure 2–78 if you want to issue ?IDEF from a 16–bit process.) Be sure to set offset ?UDVIS in the DCT to the address of the interrupt service routine for the new device, and define the interrupt service mask in offset ?UDVMS. In addition, you must set offset ?UDVBX (the mailbox) to 0.

For devices that reside on the secondary IOCs (i.e., device codes 64.–191.), you must use PIO instructions to communicate with your device. NOVA I/O is limited to the first IOC.

You may extend the DCT to include a word pointer to a device termination routine. In this case control passes to the routine if your process traps or terminates. This transfer of control prevents a runaway ?IDEF–specified device from altering system databases. Also, the environment would be the same at process termination time as if the device had just requested an interrupt. You might specify the extended DCT if you are writing an interrupt service routine (ISR) for a device such as an intelligent asynchronous controller (IAC) or an intelligent synchronous controller (ISC).

To extend the DCT, place ?UDLX in offset ?UDRS and place a word pointer to your device termination routine in offset ?UDDTR. The length of the extended packet is ?UDLE (= ?UDLN+?UDLX) words.

To specify the standard DCT, place 0 in offset ?UDRS and allow ?UDLN words for the unextended packet. In other words, programs written before you could extend the DCT (effective with AOS/VS Revision 7.00) will not have to change.

If you issue ?IDEF against a device that AOS/VS is using, you receive error code ERIBS. AOS/VS uses devices that were identified to it during the system generation process. An example is an MTD magnetic tape controller. AOS/VS also uses the following devices.

Device Mnemonic	Code	Description
BMC	–	Burst Multiplexor Channel
CPU	77	Central Processor
DCH	–	Data Channel
PIT43	Programmable Interval Timer	
RTC	14	Real–Time Clock
SCP	45	System Control Processor
TTI10	Primary Asynchronous Line Input	
TTO	11	Primary Asynchronous Line Output
UPSC	4	Universal Power Supply Controller

In addition, AOS/VS uses a DRT (Dual Receive/Transmitter, device code 34) on MV/1400 DC and MV/2000 DC computers. Another name for this device is DUART.

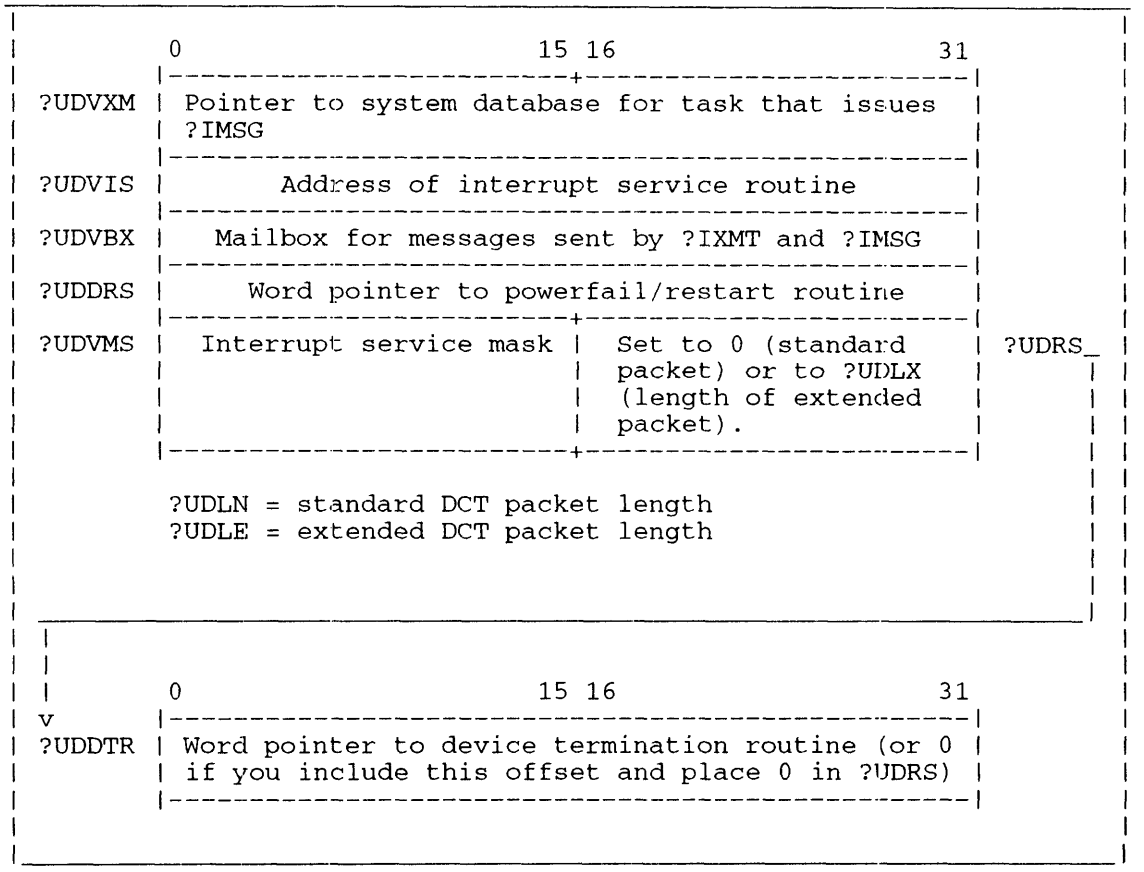


Figure 2–77. Structure of Device Control Table (DCT) for 32–Bit Processes

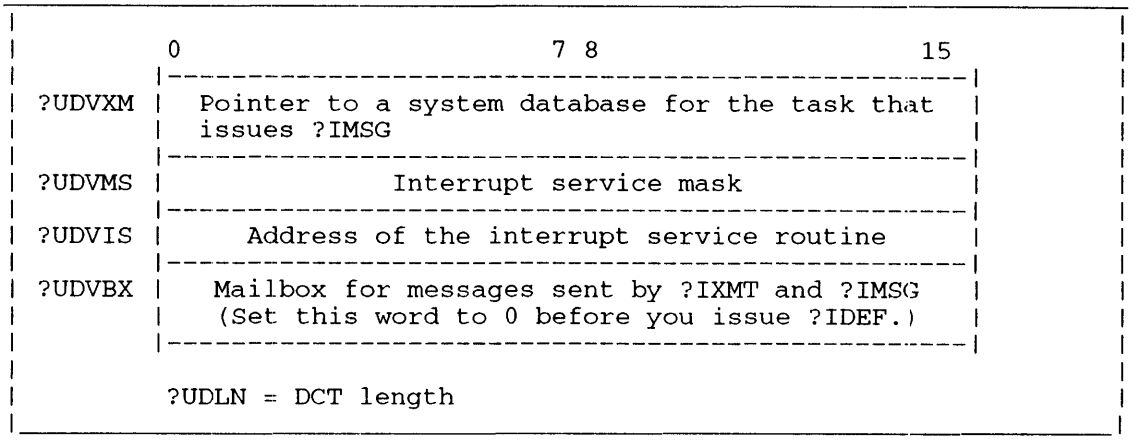


Figure 2–78. Structure of Device Control Table (DCT) for 16–Bit Processes

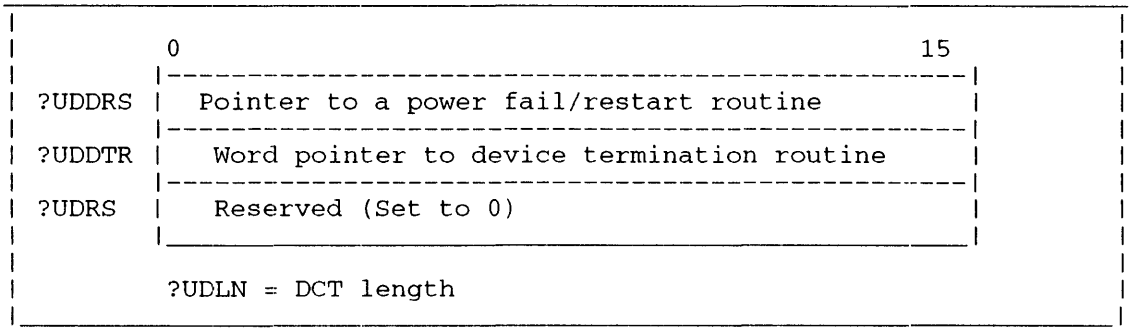


Figure 2–78.1. Structure of Extended Packet for 16–Bit Processes

Options

Set Bit 0 of AC1 if you want to use either the data channel (DCH) or the burst multiplexor channel (BMC) for I/O transfers to and from the new device. If you choose the DCH or BMC option, you must also define the number of map slots the device will need. You can load the map slot value into AC2 before you issue ?IDEF or you can set up a map definition table in your address space. If you set up a map definition table, load its address into AC2 before you issue ?IDEF. (If you use AC2, the operating system will allocate map slots in DCH map A.)

Set Bit 2 of AC0, if you want to use the extended packet to clear LBUS interrupts. Figure 2–78.1 shows the structure of the extended packet.

The map definition table specifies the first acceptable map slot for BMC or DCH transfers, and optionally, selects a particular DCH map (maps A through P). The map definition table can contain as many as eight entries. Each entry is ?UDELTH words long. The entire table (with eight entries) is ?UDLTH words long. (See Figure 2–79 for the structure of a map definition table entry and see Table 2–57 for a description of its contents.)

You must also set up a histogram packet of ?HWLTH words in your logical address space, and reserve a buffer to receive the histogram statistics. Load the packet address into AC2. (An offset within the packet points to the buffer address.) Figure 2–80 shows the structure of the histogram packet for 16–bit processes.

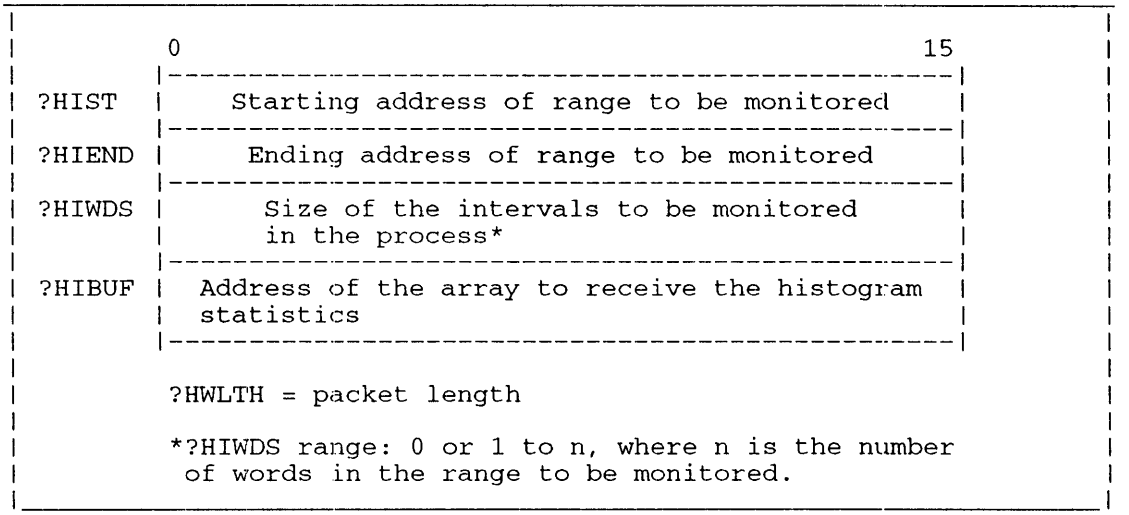


Figure 2–80. Structure of ?IHIST Packet

To start a simple histogram, which merely records how often the target process gained CPU control, set offset ?HIWDS to 0. This directs the operating system to ignore the contents of offsets ?HIST and ?HIEND and prevents the operating system from gathering range statistics. Set offset ?HIBUF to the address of the array you have reserved in your logical address space for the histogram statistics.

Each histogram array has two parts: a fixed–length header followed by double–precision array elements, which correspond to each interval that you want to monitor. Table 2–58 shows the structure of the histogram array for 16–bit processes.

Table 2–58. Structure of ?IHIST Array

Array Offset		Contents
?HTTH	?HTTL	Total number of real-time clock pulses (ticks) counted in this histogram.
?HPRH	?HPRL	Total number of ticks when the program counter (PC) was within the target process, but outside the specified range.
?HAPH	?HAPL	Total number of ticks in other processes.
?HSBH	?HSBL	Total number of ticks in the OS, except those recorded when it was in an idle loop.
?HSIH	?HSIL	Total number of ticks in a system idle loop.
?HARAY	?HARAY+1	Total number of ticks in the first interval.
...		
?HARAY+n*2-2	?HARAY+n*2-1	Total number of ticks in the nth interval.

?ILKUP
error return
normal return

Input

AC0	Byte pointer to the pathname of the target IPC entry
AC1	Reserved (Set to 0.)
AC2	Reserved (Set to 0.)

Output

AC0	Unchanged
AC1	Global port number associated with the IPC entry
AC2	File/device type of the IPC file entry

Error Codes in AC0

ERFDE File does not exist
ERIFT Illegal file type (The file that you specified in AC1 is not an IPC file.)
ERVBP Invalid byte pointer passed as a system call argument
ER_FS_OBSOLETE_IPC_FILE_DETECTED
Obsolete IPC file type has been detected; file has been deleted (AOS/VS II only)
ER_FS_DIRECTORY_NOT_AVAILABLE
Directory not available because the LDU was force released (AOS/VS II only)

Why Use It?

?ILKUP returns the global port number of the IPC file that the target process previously created. Because a process must have the correct global port number to send an IPC message to another process, ?ILKUP can be a useful preliminary step to an ?ISEND or an ?IREC.

Who Can Use It?

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

What It Does

?ILKUP returns the global port number (which includes the PID, the ring number, and the local port number of the creator) associated with an IPC file entry that you specify. If the file does not exist or is not an IPC file, ?ILKUP fails and the operating system returns error code ERFDE or ERIFT in AC0.

Note that if you specify an IPC file whose local port number is 0, the number returned in AC1 is the port number of your terminal, not the port number of the file that you specified.

Notes

- See the descriptions of ?IREC and ?ISEND in this chapter.
- See PARU.32.SR or PARU.16.SR, for the IPC file and device types.

?IREC *[header address]*
error return
normal return

Input

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

Output

AC0	Undefined
AC1	Undefined
AC2	Address of the IPC header

Error Codes in AC0

ERMPR	System call parameter address error (The receive buffer is not in the unshared area of your address space.)
ERVBP	Invalid byte pointer passed as a system call argument
ERVWP	Invalid word pointer passed as a system call argument

Why Use It?

You would issue ?IREC either to receive an IPC message that another process sent via ?ISEND or to receive a process termination or connection management message.

Who Can Use It?

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

What It Does

?IREC opens a receiving port for the calling process, which allows the caller to receive an IPC message from a sending process. ?IREC requires a header, which lists, among other information, the origin and destination ports for the message.

Before you issue ?IREC, set up the header and reserve a receive buffer in your logical address space. You can cite the header address as an argument to ?IREC or load it into AC2 before you issue ?IREC. The ?IREC header consists of ?IPLTH words. Figure 2–81 shows the header's structure, and Table 2–60 describes its contents. Table 2–60 also describes the optional contents of ?ISFL in the ?IREC header.

Offset ?IOPH contains the global port number of the calling process (issue ?ILKUP to obtain this). If you set offset ?IOPH to 0, the calling process can accept messages from any sender. Similarly, if you set offset ?IDPN (the destination port number) to 0, the calling process can accept the message on any of its ports. During message transmission, the operating system writes the actual origin port number into offset ?IOPH, and the actual destination port number into offset ?IDPN.

Offset ?IPTR points to the receive buffer, which must be in the unshared area of your logical address space. (See Table 2–60 which describes the system flag words and user flag words.)

By default, the operating system suspends the receiver if there is no outstanding message for its ?IREC. You can avoid this by setting flag ?IFNBK in offset ?ISFL of the receive header. This flag signals the operating system to return an error to the receiver if there is no message.

An ?IREC from a particular ring can only receive an IPC message whose destination is that particular ring.

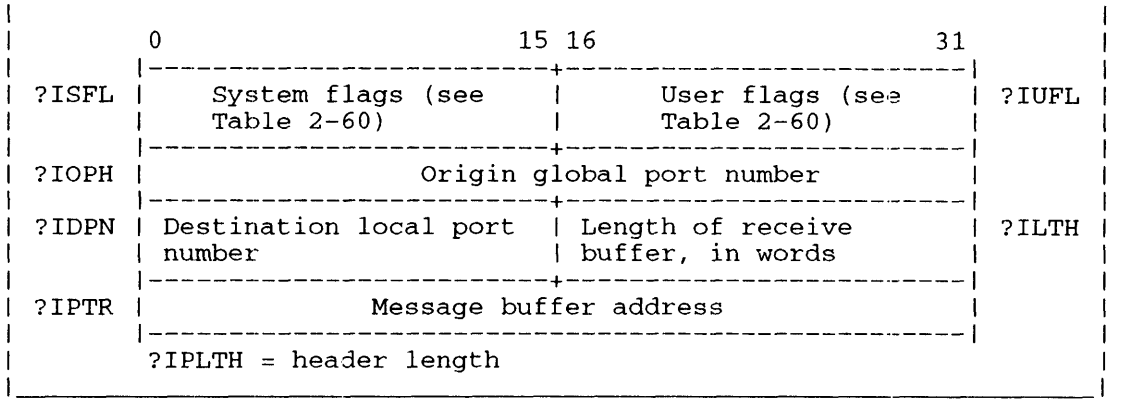


Figure 2-81. Structure of ?IREC Header

Table 2-60. Contents of ?IREC Header*

Offset	Contents
?ISFL	System flag word ?IFRFM--Receive a looped message (sent by this process to itself). Use this to test a process to see if both IPCs are working properly. Also, this is a good way for one program to communicate with another program in a single process (?CHAIN). ?IFSOV--Spool the message if the receive buffer is too small. ?IFNBK--Signal an error if there is no spooled message for this receiver. ?IFRING--Contains the sender's ring field (returned by the OS).(The ring field is a 3-bit field, Bits 13 through 15.) ?IFPR---Indicates .PR file type of sender; 0 if the sender is a 32-bit process, or 1 if the sender is a 16-bit process (returned by the OS).
?IUFL	User flag word. (See Figure 2-82.)
?IOPH (doubleword)	Origin global port number.
?IDPN	Destination local port number.
?ILTH	Size of the receive buffer, in words.
?IPTR (doubleword)	Address of the receive buffer.

* There is no default unless otherwise specified.

After ?IREC has finished you can identify the process that issued the associated ?ISEND or ?IS.R call. To do this, load the origin global port number from offset ?IOPH of the ?IREC packet into AC0 and then issue ?ISPLIT. After ?ISPLIT finishes AC1 will contain the PID of the sending process.

Sample Packet

```
HDR:    .BLK    ?IPLTH          ;Allocate enough space for the
                                ;packet.  Packet length = ?IPLTH.
        .LOC    HDR+?ISFL       ;System flags.
        .WORD   ?IFSOV          ;If buffer is too short, spool.
        .LOC    HDR+?IUFL       ;User flags.
        .WORD   0               ;Set to 0.
        .LOC    HDR+?IOPH       ;Global port number of calling
        .DWORD  0               ;process.  (Set to 0 to accept
                                ;messages from any sender.)
        .LOC    HDR+?IDPN       ;Destination port number.
        .WORD   0               ;Set to 0 to accept messages on any
                                ;of the calling process's ports.
        .LOC    HDR+?ILTH       ;Receive buffer length (words).
        .WORD   80./2           ;Buffer length is 80./2.  Reset
                                ;buffer length after each ?IREC.
        .LOC    HDR+?IPTR       ;Receive buffer address (must be in
        .DWORD  RECBUF          ;unshared area of your logical
                                ;address space).
        .LOC    HDR+?IPLTH      ;End of ?IREC header.
```

Process Termination Messages

When a process terminates, the system sends a process termination IPC message to the father process (who created the terminating process). The system also sends connection management messages to the server (or customer) if the status of the connection in a customer/server relationship is affected. For a process to receive a process termination or connection management message, it must issue an ?IREC system call with offset ?IOPH set to a special system port, ?SPTM.

The location of the PID number, that ?IREC receives as part of a process termination or connection management message, depends on two things: the type of process that the receiver (i.e., the process that issues ?IREC) is, and the type of message that is received, as follows:

- A–type (smallPID) process, process termination message — the PID is in the right byte of offset ?IUFL. If ?ILTH contains 0, then the contents of ?IPTR are undefined. If ?ILTH doesn't contain 0, then ?IPTR contains a word pointer to a process termination message as described in the sections "Termination Messages for A–Type 32–bit Processes," and "Termination Messages for A–Type 16–bit Processes." For more information about process types see *Managing AOS/VS and AOS/VS II* (093–000541), and *AOS/VS System Concepts* (093–000335).

- B– or C–type (hybrid or anyPID programs) process, process termination message — the operating system returns 0 in the right byte of offset ?IUFL. If ?ILTH contains 0, then the contents of ?IPTR are undefined. If ?ILTH doesn't contain 0, then ?IPTR contains a word pointer to a process termination message as described in the section "Termination Messages for B–Type and C–Type Processes." For more information about process types see *Managing AOS/VS and AOS/VS II* (093–000541), and *AOS/VS System Concepts* (093–000335).

- A–type process, connection management (obituary) message — the PID is in the right byte of offset ?IUFL. Regardless of the contents of ?ILTH, the 24 high order bits of ?IPTR are undefined and the low order 8 bits contain a bit map according to the description of connection management in the *AOS/VS System Concepts* manual.

- B– or C–type process, connection management (obituary) message — the PID is in the 16 low order bits of offset ?IPTR (and the 16 high order bits are undefined). The contents of offset ?ILTH are unimportant, but the right byte of offset ?IUFL contains a bit map according to the description of connection management in the *AOS/VS System Concepts* manual.

An A–type process receives a termination message as defined in one of the following two places:

- Table 2–60 (page 2–296) and Table 2–63.1 (page 2–303)
- Table 2–60 (page 2–296) and Table 2–63 (page 2–301)

A B–type or C–type process receives a termination message whose receive buffers are defined in one of the following two places:

- Table 2–63.2 (page 2–304.2) in the section "Termination Messages for B–Type and C–Type Processes," is for 32–bit processes.
- Table 2–64 (page 2–304.6) in the section "Termination Messages for B–Type and C–Type Processes," is for 16–bit processes.

The operating system uses offset ?IUFL of the receive header to describe the reason for the Process Termination Message and to identify the process on whose behalf it is sending the message. Figure 2–82 shows the structure of offset ?IUFL.

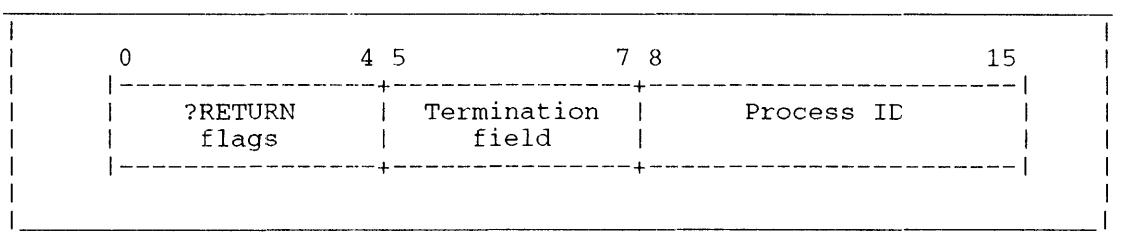


Figure 2–82. Structure of Offset ?IUFL

The termination field (bits 5 through 7) in offset ?IUFL is for the codes that the operating system uses to indicate why the termination message is being sent. The low order byte of offset ?IUFL always contains the PID of the process from which the message originated.

The termination field may contain any of the codes listed in Table 2–62, depending on the reason for the message. Note that when the termination field holds the value ?TEXT, you can find the real cause for the message in the first word of the message itself.

If a ?RETURN system call causes the termination message, the operating system uses the ?RETURN flags (bits 0 through 4) of the ?IUFL offset to provide additional information about the message format. See Table 2–61 for the ?RETURN flags.

If the message is in standard CLI format, the return flag is set to ?RFCF, and the remaining flags (?RFEC, ?RFWA, ?RFER, and ?RFAB) have their conventional meanings. If the father is not the CLI, or if ?RFCF is not set, the father and son must agree about the contents of the message.

Table 2–61. Process Termination Codes in the ?RETURN Portion of ?IUFL

Code	Meaning
?RFCF	The termination message is in CLI format (the CLI is the father).
?RFEC	AC0 contains the error code.
?RFWA	A warning condition caused the termination.
?RFER	An error condition caused the termination.
?RFAB	An abort condition caused the termination.

If a Ctrl–C Ctrl–B (or Ctrl–C Ctrl–E) interrupt sequence caused the process to terminate, ?TCIN appears in the termination field of ?IUFL. If the operating system terminated the process because of an error condition, the system returns ?TAOS as the cause of termination. You can find a further explanation for the termination by looking at the error code returned in the ?IPTR offset of the message header.

If a customer/server relationship caused the message, the system returns codes ?TBCX, ?TCCX, or ?TABR in the termination field.

Table 2–62. Process Termination Codes in Offset ?IUFL for ?IREC and ?ISEND Headers

Code	Meaning
?TSELF	Either a 16-bit process terminated itself with a ?TERM or a ?RETURN system call or a 32-bit process terminated itself with a ?TERM system call.
?TRAP	A user trap terminated a 16-bit process; Word 5 of the IPC message to the father describes the trap.
?TCIN	An abort terminal interrupt (Ctrl-C Ctrl-B sequence) terminated a process.
?TSUP	Terminated by a superior process.
?TAOS	The OS terminated a process because of an error; offset ?IPTR in the IPC header contains the error code.
?TBCX	A process broke a connection that was established via the connection-management system calls.
?TCCX	The connection still exists, but the process chained.
?TEXT	Indicates an extended termination code; the extended code appears in offset 0 (first word) of the IPC message. A termination code of ?TEXT means that the actual termination code is a 16-bit code in the first word of the termination message in the receive buffer. The The following list describes these extended termination codes.

Extended Code	Meaning
?T32T	A 32-bit process terminated itself with a ?TERM or a ?RETURN system call.
?TR32	A 32-bit process terminated because of a user trap; word 10 of the termination message describes the trap.
?TABR	Task abort notification to a server process. This involves customer/server relationship, ?IDGOTO, ?IS.R, and ?IREC, either 16-bit or 32-bit process.

Termination Messages for A–Type 32–Bit Processes

When a 32–bit process terminates because of a ?TERM (without the optional message) or ?RETURN system call, the operating system sets the termination field of ?IUFL to ?TEXT and sets the first word of the termination message to ?T32T. If the message is due to a ?RETURN, the operating system uses the ?RETURN flags field of ?IUFL, and the message has the following format:

Word 0	?T32T (the extended termination code)
Word 1	Byte length of the ?RETURN message text
Words 2 and 3	Error code from ?RETURN AC0
Word 4	Start of message text

If the process terminated with a ?TERM with the optional message, the operating system forwards the message without modification.

If the 32–bit process terminated because of a user trap, the operating system sends a message in the format shown in Table 2–63. The length of this message is ?TPLN words.

Table 2–63. ?TEXT Code Termination Messages Sent on an A–Type 32–Bit Process User Trap

Word	Contents
0	?TR32 (the extended termination code).
1 and 2	AC0 contents.
3 and 4	AC1 contents.
5 and 6	AC2 contents.
7 and 8	AC3 contents.
9	Bit 0, carry; Bits 1 through 15, high-order bits of program counter.
10	Low-order bits of program counter.

(continued)

Table 2–63. ?TEXT Code Termination Messages Sent on an A–Type 32–Bit Process User Trap

Word	Contents
11	The following flag bits, which describe the trap
	Bit 0=0 Trap occurred while control was in the user context.
	Bit 0=1 Trap occurred while control was in the operating system.
	Bit 3=1 A node time-out occurred. (This is a hardware error.)
	Bit 4=1 Process tried to execute a privileged instruction.
	Bit 5=1 Process tried to return to an inner ring from a subroutine call. (This is a violation of the ring structure.)
	Bit 6=1 Process tried to issue a subroutine call to an outer ring. (This is a violation of the ring structure.)
	Bit 7=1 Gate protection error. (This is a violation of the ring structure.)
	Bit 8=1 Process tried to reference an address in an inner ring. (This is a violation of the ring structure.)
	Bit 9=1 Process tried to read a read-protected page.
	Bit 10=1 Process tried to execute data in an execute-protected area.
	Bit 12=1 Process tried to write into a write-protected area.
	Bit 13=1 Memory map validity error. (The process tried to refer to an address outside the user context.)
	Bit 14=1 Defer error. (The process tried to use more than 16 levels of indirection in an address reference.)
	Bit 15=1 Process tried to issue a machine-level I/O instruction without issuing the ?DEBL system call.

(concluded)

Termination Messages for A–Type 16–Bit Processes

When a 16–bit process terminates by issuing a ?RETURN or ?TERM system call, the system returns flag ?TSELF to the termination field in offset ?IUFL.

If the process terminated via the ?RETURN system call, the system sends one or more of the codes in Table 2–61 to the ?RETURN Flags field of ?IUFL. Furthermore, if ?RETURN was used, a 2–word header precedes the text of the message furnished by the ?RETURN:

Word 0 Length of the ?RETURN message in bytes.
Word 1 The error code (if any) found in ?RETURN's AC0.

The text of the message follows this header. If the process supplied no message, only the first two words appear.

If the process terminated itself via the ?TERM system call, the operating system returns either the termination message specified by the system call or returns nothing to the ?IREC receive buffer, if no message was sent.

If the 16–bit process terminated because of a user trap, the operating system sets the termination field to ?TRAP and sends the father a six–word message, as shown in Table 2–63.1.

If the process terminated because of an abort terminal interrupt (Ctrl–C Ctrl–B) or a ?TERM issued by a superior process, the operating system returns ?TCIN or ?TSUP, respectively, to the termination field, but sends no message.

Table 2–63.1. ?TRAP Termination Messages for A–Type 16–Bit Processes

Word	Contents
0	AC0 contents at the time of the trap.
1	AC1 contents at the time of the trap.
2	AC2 contents at the time of the trap.
3	AC3 contents at the time of the trap.
4	Bit 0, carry; Bits 1 through 15, program counter value.
5	The following flag bits, which describe the trap: Bit 0=0 Trap occurred while control was in the user context. Bit 0=1 Trap occurred while control was in the operating system. Bit 12=1 Process tried to write into a write-protected area. Bit 13=1 Memory map validity error. (The process tried to refer to an address outside the user context.) Bit 14=1 Defer error. (The process tried to use more than 16 levels of indirection in an address reference.) Bit 15=1 Process tried to issue a machine-level I/O instruction without issuing the ?DEBL system call.

Termination Messages for B–Type and C–Type Processes

B–type or C–type processes receive the new style termination message described on the next several pages. These formats apply only to issuing processes that are type B or type C.

The calling process is either a 32–bit one or a 16–bit one. Figure 2–82.1 describes the 32–bit termination message format, and Table 2–63.2 describes its contents. Figure 2–82.2 describes the 16–bit termination message format and Table 2–64 describes its contents.

Word ?IUFL of the termination IPC header contains the value of ?TEXT in the termination field, and the value indicates an extended termination message. Also, the PID field of ?IUFL contains 0.

If the ?TERM or ?RETURN call in the terminated process has a user–supplied message, then the values in words ?ISFL and ?IUFL of the user–supplied IPC header go to offsets ?TMR8 and ?TMR9 respectively.

A ?RETURN call has the return flags set in the IPC header for a user–supplied message. The default messages from a ?RETURN call are unchanged and will appear in their entirety in the ?TMMSG area of the termination message. Any user–supplied IPC to a ?TERM call will similarly be appended in this area. If the terminated process passes only an IPC header, then offset ?TMMLG contains zero and the first two words of offset ?TMMSG contain ?IPTR. Note that user–supplied messages are only appended to the default message; they never replace it.

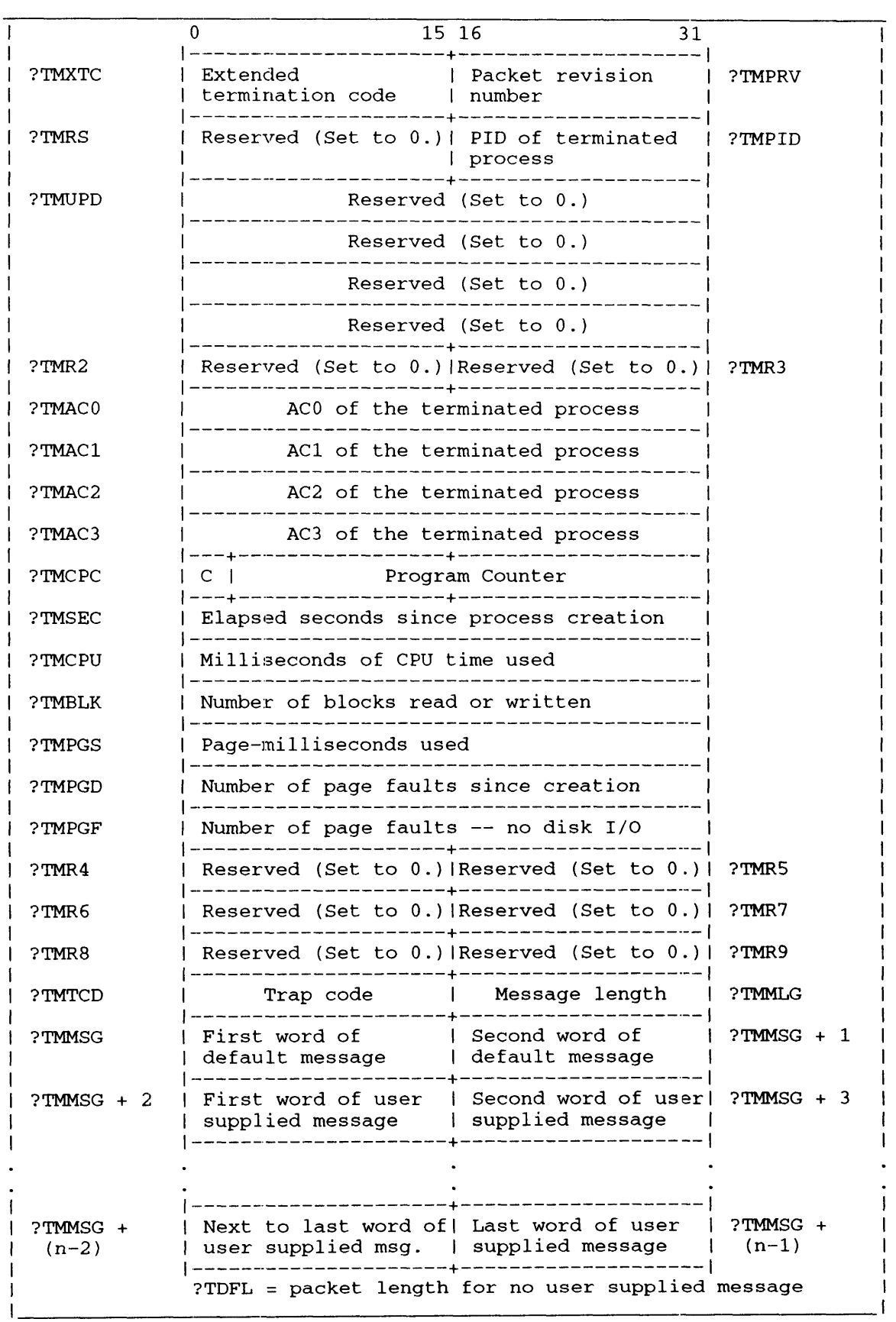


Figure 2–82.1. Structure of Termination Message from a 32–bit B– or C–Type Process

Table 2–63.2. Contents of Termination Message from a 32–bit B– or C–Type Process

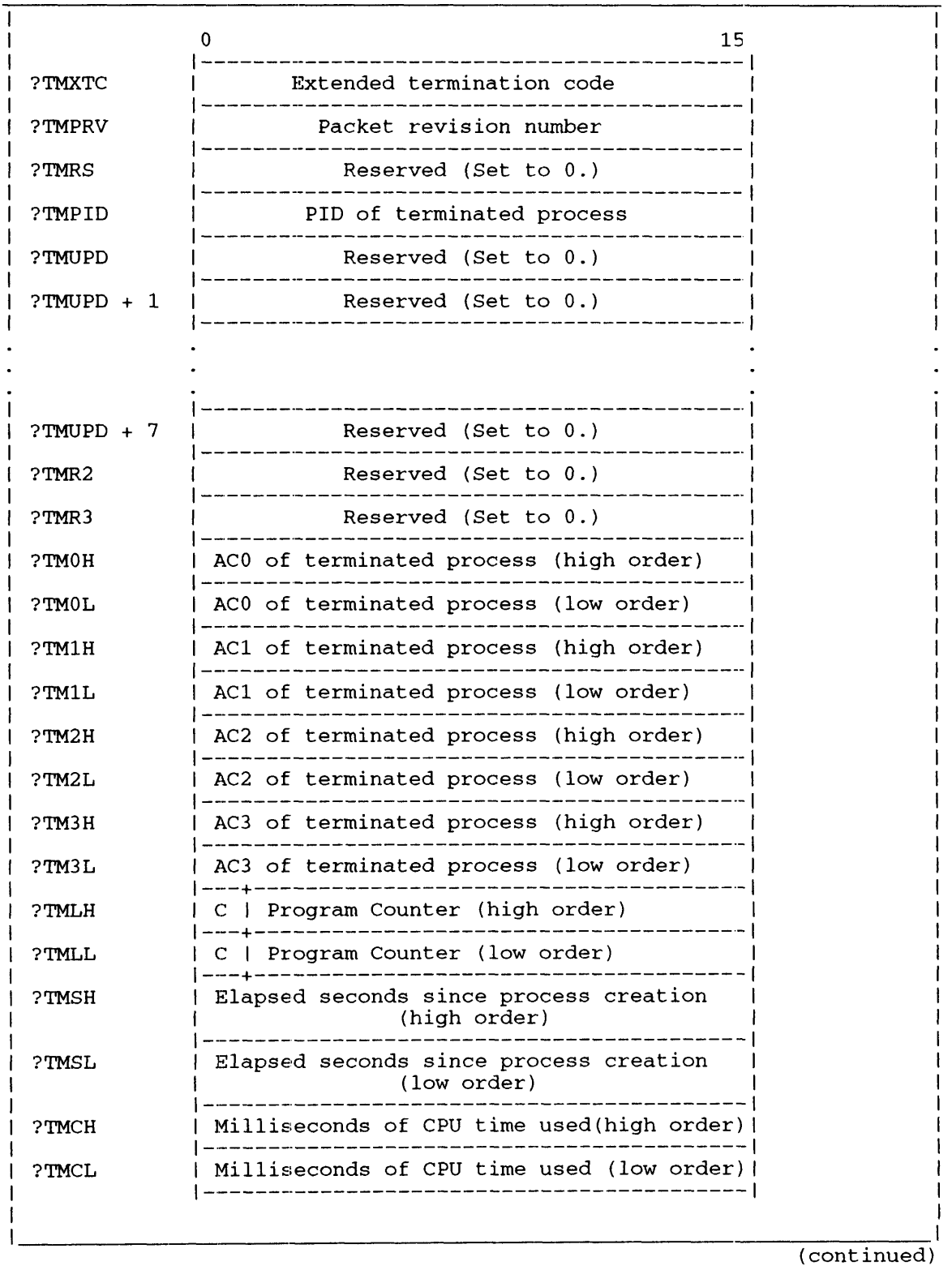
Offset	Contents
?TMXTC	The operating system returns the extended termination code to indicate the type of process termination. The values and meanings of these codes are as follows. ?XT16T -- A 16-bit termination of self occurred. ?XTR16 -- A 16-bit user trap occurred. ?XTCIN -- Termination by a terminal interrupt. ?XTSUP -- Termination by a superior process. ?XTAOS -- Termination by AOS. ?XTBCX -- Customer connection broken. ?XTCCX -- Customer chained. ?XTABR -- Customer did ?TABT. ?XT32T -- A 32-bit termination of self occurred. ?XTR32 -- A 32-bit user trap occurred.
?TMPRV	Packet revision number. Place ?TM6 here.
?TMRS	Reserved. Set to 0.
?TMPID	The operating system returns the PID of the terminated process.
?TMUPD (8. words)	Reserved. Set to 0.
?TMR2	Reserved. Set to 0.
?TMR3	Reserved. Set to 0.
?TMAC0 (double-word)	The operating system returns the contents of AC0 at the time of the termination. This offset is valid only on a trap (?TMXTC contains ?XTR16 or ?XTR32). The high-order word is undefined if a 16-bit process terminated.
?TMAC1 (double-word)	The operating system returns the contents of AC1 at the time of the termination. This offset is valid only on a trap (?TMXTC contains ?XTR16 or ?XTR32). The high-order word is undefined if a 16-bit process terminated.
?TMAC2 (double-word)	The operating system returns the contents of AC2 at the time of the termination. This offset is valid only on a trap (?TMXTC contains ?XTR16 or ?XTR32). The high-order word is undefined if a 16-bit process terminated.
?TMAC3 (double-word)	The operating system returns the contents of AC3 at the time of the termination. This offset is valid only on a trap (?TMXTC contains ?XTR16 or ?XTR32). The high-order word is undefined if a 16-bit process terminated.

(continued)

Table 2–63.2. Contents of Termination Message from a 32–bit B– or C–Type Process

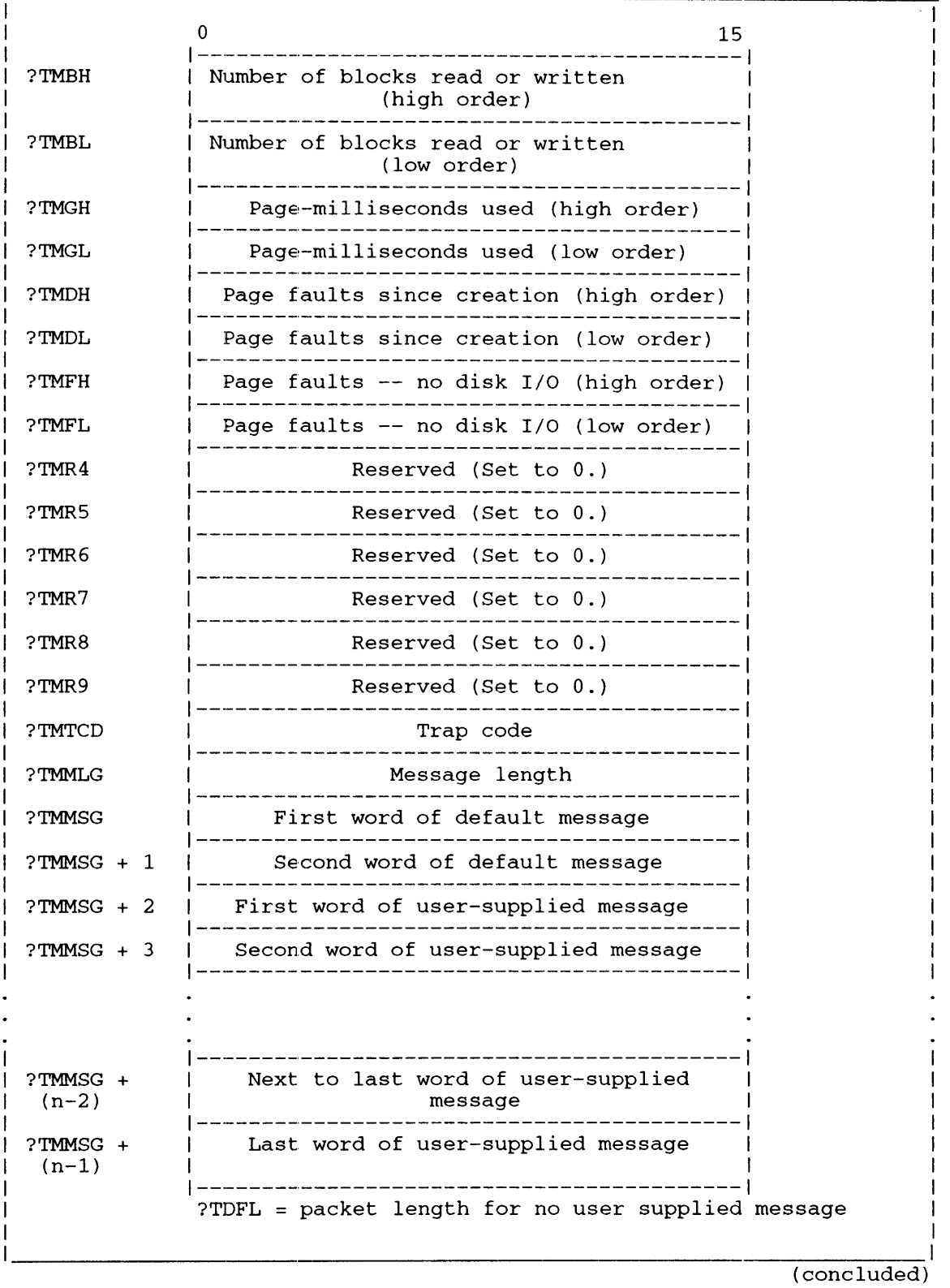
Offset	Contents
?TMCPC (double-word)	The operating system returns the carry bit and program counter contents at the time of the termination. This offset is valid only on a trap (?TMXTC contains ?XTR16 or ?XTR32). The high-order word is undefined if a 16-bit process terminated.
?TMSEC (double-word)	The operating system returns the number of elapsed seconds since the process was created.
?TMCPU (double-word)	The operating system returns the number of milliseconds of CPU time the process used.
?TMBLK (double-word)	The operating system returns the number of blocks read or written.
?TMPGS (double-word)	The operating system returns the page usage over CPU time in pages/second.
?TMPGD (double-word)	The operating system returns the number of page faults since the process was created.
?TMPGF (double-word)	The operating system returns the number of page faults, with no disk I/O, since the process was created.
?TMR4	Reserved. (Set to 0.)
?TMR5	Reserved. (Set to 0.)
?TMR6	Reserved. (Set to 0.)
?TMR7	Reserved. (Set to 0.)
?TMR8	Reserved. (Set to 0.)
?TMR9	Reserved. (Set to 0.)
?TMTCD	The operating system returns the trap code.
?TMMLG	The operating system returns the number of words in any user-supplied message. This number might be zero.
?TMMSG	Beginning of the message area, starting with two words; ending with the zero or more words in any user-supplied message.

(concluded)



(continued)

Figure 2–82.2. Structure of Termination Message from a 16–bit B– or C–Type Process



(concluded)

Figure 2–82.2. Structure of Termination Message from a 16–bit B– or C–Type Process

Table 2–64. Contents of Termination Message from a 16–bit B– or C–Type Process

Offset	Contents
?TMXTC	The operating system returns the extended termination code to indicate the type of process termination. The values and meanings of these codes are as follows. ?XT16T -- a 16-bit termination of self occurred. ?XTR16 -- a 16-bit user trap occurred. ?XTCIN -- termination by a terminal interrupt. ?XTSUP -- termination by a superior process. ?XTAOS -- termination by AOS. ?XTBCX -- customer connection broken. ?XTCCX -- customer chained. ?XTABR -- customer did ?TABT. ?XT32T -- a 32-bit termination of self occurred. ?XTR32 -- a 32-bit user trap occurred.
?TMPRV	Packet revision number. Place ?TM6 here.
?TMRS	Reserved. (Set to 0.)
?TMPID	The operating system returns the PID of the terminated process.
?TMUPD (8 words)	Reserved. (Set to 0.)
?TMR2	Reserved. (Set to 0.)
?TMR3	Reserved. (Set to 0.)
?TM0H	The operating system returns the high-order contents of AC0 at the time of the termination. This offset is valid only on a trap (?TMXTC contains ?XTR16 or ?XTR32). This offset is undefined if a 16-bit process terminated.
?TM0L	The operating system returns the low-order contents of AC0 at the time of the termination. This offset is valid only on a trap (?TMXTC contains ?XTR16 or ?XTR32).
?TM1H	The operating system returns the high-order contents of AC1 at the time of the termination. This offset is valid only on a trap (?TMXTC contains ?XTR16 or ?XTR32). This offset is undefined if a 16-bit process terminated.
?TM1L	The operating system returns the low-order contents of AC1 at the time of the termination. This offset is valid only on a trap (?TMXTC contains ?XTR16 or ?XTR32).
?TM2H	The operating system returns the high-order contents of AC2 at the time of the termination. This offset is valid only on a trap (?TMXTC contains ?XTR16 or ?XTR32). This offset is undefined if a 16-bit process terminated.
?TM2L	The operating system returns the low-order contents of AC2 at the time of the termination. This offset is valid only on a trap (?TMXTC contains ?XTR16 or ?XTR32).

(continued)

Table 2–64. Contents of Termination Message from a 16–bit B– or C–Type Process

Offset	Contents
?TM3H	The operating system returns the high-order contents of AC3 at the time of the termination. This offset is valid only on a trap (?TMXTC contains ?XTR16 or ?XTR32). This offset is undefined if a 16-bit process terminated.
?TM3L	The operating system returns the low-order contents of AC3 at the time of the termination. This offset is valid only on a trap (?TMXTC contains ?XTR16 or ?XTR32).
?TMLH	The operating system returns the carry bit and high-order program counter contents if and when a 32-bit process terminated. This offset is valid only on a trap (?TMXTC contains ?XTR16 or ?XTR32). The offset is undefined if a 16-bit process terminated.
?TMLL	The operating system returns the low-order program counter contents if and when a 32-bit process terminated, or else the carry bit and program counter contents if and when a 16-bit process terminated. The offset is valid only on a trap (?TMXTC contains ?XTR16 or ?XTR32).
?TMSH	The operating system returns the number of elapsed seconds since the process was created (high order).
?TMSL	The operating system returns the number of elapsed seconds since the process was created (low order).
?TMCH	The operating system returns the number of milliseconds of CPU time the process used (high order).
?TMCL	The operating system returns the number of milliseconds of CPU time the process used (low order).
?TMBH	The operating system returns the number of blocks read or written (high order).
?TMBL	The operating system returns the number of blocks read or written (low order).
?TMGH	The operating system returns the page usage over CPU time in pages/second (high order).
?TMGL	The operating system returns the page usage over CPU time in pages/second (low order).
?TMDH	The operating system returns the number of page faults since the process was created (high order).
?TMDL	The operating system returns the number of page faults since the process was created (low order).

(continued)

Table 2–64. Contents of Termination Message from a 16–bit B– or C–Type Process

Offset	Contents
?TMFH	The operating system returns the number of page faults, with no disk I/O, since the process was created (high order).
?TMFL	The operating system returns the number of page faults, with no disk I/O, since the process was created (low order).
?TMR4	Reserved. (Set to 0.)
?TMR5	Reserved. (Set to 0.)
?TMR6	Reserved. (Set to 0.)
?TMR7	Reserved. (Set to 0.)
?TMR8	Reserved. (Set to 0.)
?TMR9	Reserved. (Set to 0.)
?TMTCD	The operating system returns the trap code.
?TMMLG	The operating system returns the number of words in any user-supplied message. This number might be zero.
?TMMSG	Beginning of the message area, starting with two words; ending with the zero or more words in any user-supplied message.

(concluded)

Notes

- See the descriptions of ?ILKUP, ?TERM, ?RETURN, and ?ISEND in this chapter.

?IRMV
error return
normal return

Input

AC0	Device code of the user device that you want to remove
AC1	Reserved (Set to 0.)
AC2	Reserved (Set to 0.)

Output

AC0	Unchanged
AC1	Undefined
AC2	Undefined

Error Codes in AC0

ERDNM	Illegal device code
ERPRE	Invalid system call parameter
ERPRV	Caller not privileged for this action
ERPTY	Illegal process type

Why Use It?

?IRMV revokes a previous ?IDEF; that is, it allows you to remove a device that you defined earlier in your program. ?IRMV removes only a specific user–defined device.

Who Can Use It?

To issue ?IRMV, a process must have privilege ?PVDV. There are no restrictions concerning file access.

What It Does

?IRMV removes a device's DCT entry from the interrupt vector table. After the operating system executes ?IRMV, it ignores the target device and all subsequent interrupts from it.

Before you issue ?IRMV, load AC0 with the device code you specified when you defined the device with ?IDEF.

Notes

- See the description of ?IDEF in this chapter.

Sample Header

```
HDR:    .BLK    ?IPLTH          ;Header length.
        .LOC    HDR+?ISFL       ;System flags.
        .WORD   0               ;Set to 0.
        .LOC    HDR+?IUFL       ;User flags.
        .WORD   0               ;Set to 0.
        .LOC    HDR+?IDPH       ;Global port number of receiving
        .DWORD  5               ;process.  (?ILKUP and ?TPORT return
                                ;this information.)
        .LOC    HDR+?IOPN       ;Origin port no.
        .WORD   7               ;Local port no. to use for this
                                ;?ISEND.
        .LOC    HDR+?ILTH       ;Message buffer length (words).
        .WORD   7               ;Set buffer length before each
                                ;?ISEND.
        .LOC    HDR+?IPTR       ;Message buffer address (must be in
        .DWORD  SBUFF           ;unshared area of your logical
                                ;address space).
        .LOC    HDR+?IPLTH      ;End of ?ISEND header.
```

Notes

- See the descriptions of ?IREC, ?IS.R, ?ILKUP, and ?TPORT in this chapter.
- In the explanation of ?PROC, see the section "Offset ?PIPC." It explains how ?PROC can send a CLI–format command line as the IPC message.
- A global port number contains the PID of the target process. In order to ensure the global port number remains unique, a connection must exist between the calling and the target processes. This connnection will cause the operating system to reserve the PID portion of the global port until the connection is explicitly broken. If an ?ISEND is attempted when the process described by the global port number has terminated, connection management will guarantee the ERIDP error will be returned until the calling process does the ?DCON. See the description of ?CON, ?DCON, ?DRCON, ?RESIGN?, and ?SERVE in this chapter.

?ISPLIT Finds the owner of a port (including its ring number).

?ISPLIT
error return
normal return

Input

AC0 One of the following:

- Global port number (32–bit users only)
- OPH, which indicates high–order bits of global port number (16–bit users only)

AC1 One of the following:

- Reserved (Set to 0.) (32–bit users only)
- OPL, which indicates low–order bits of global port number (16–bit users only)

AC2 Reserved (Set to 0.)

Output

AC0 One of the following:

- Ring field in Bits 29 through 31 (32–bit users only)
- Ring field in Bits 13 through 15 (16–bit users only)

AC1 PID of the process that owns the port

AC2 Local port number

Error Codes in AC0

No error codes are currently defined.

Why Use It?

?ISPLIT, like ?IMERGE, allows both 16– and 32–bit users to manipulate ring fields within global port numbers.

Who Can Use It?

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

What It Does

?ISPLIT returns the ring field, the PID, and the local port number associated with the global port number you specify in AC0.

Notes

- See the description of ?IMERGE in this chapter.

Offset ?LBFG contains status bits that allow you to specify the tape unit's density mode and whether you want the label in IBM format. There are three density mode settings (comparable to those in the ?OPEN packet): ?LB8 sets the tape unit to a density of 800 bytes/inch; ?LB16 sets the unit to 1600 bytes/inch; ?LB62 sets the tape unit to a density of 6,250 bytes/inch; ?LBAM directs the operating system to set the correct density automatically.

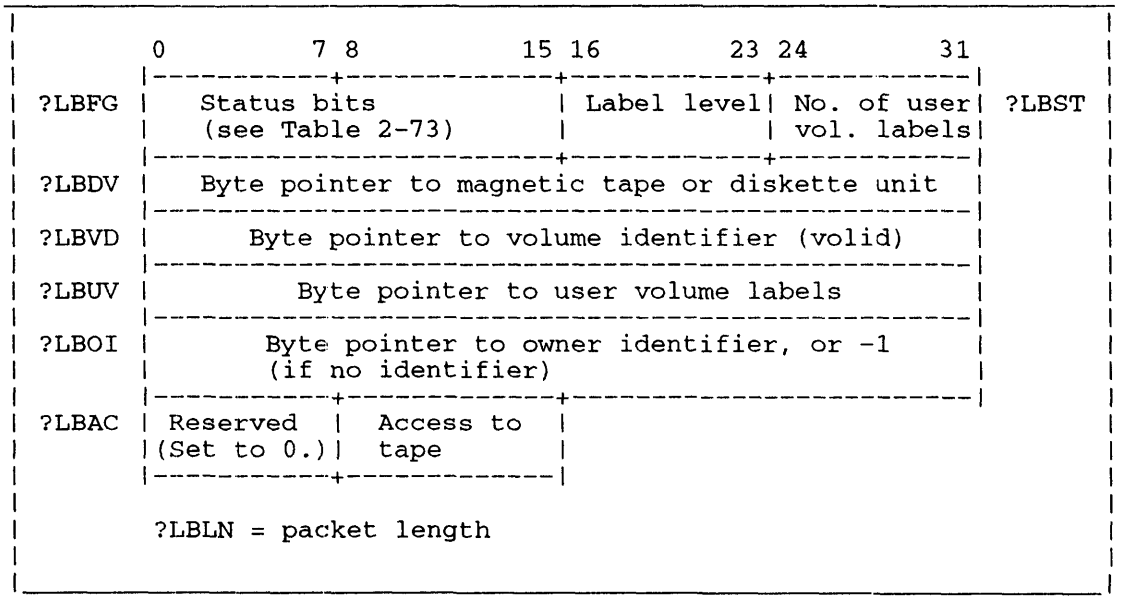


Figure 2–91. Structure of ?LABEL Packet

If you default the density mode, the operating system sets the unit to the density specified in the system–generation procedure. Before you default this parameter, make sure the tape's density matches the density specification set during the system–generation procedure.

Parameter ?LBSC in offset ?LBFG ("scratch this tape") causes the operating system to "scratch" (overwrite) all data on the tape or diskette as it supplies the label.

Use the right byte of offset ?LBAC to specify access to the media. Typically, you supply a space character (ASCII 040) in this byte, which gives all users full access to the media.

Table 2–73. Contents of ?LABEL Packet*

Offset	Contents
?LBFG	Status bits ?LBIM--Label is in IBM format (tape only). ?LBSC--Scratch this tape/diskette. ?LBMF--Use buffered mode (MTJ Model 6352 tape only). ?LBMS--Use streaming mode (MTJ Model 6352 tape only). Density mode (tape only): ?LB8--800 bytes/inch. ?LB16--1600 bytes/inch. ?LB62--6250 bytes/inch. ?LBAM--automatic density matching. ?LB5--Low tape density. ?LB6--Medium tape density. ?LB7--High tape density. Error handling (diskette only): ?LBMR--Check for bad blocks and remap if necessary. ?LBMP--Check for bad blocks and return ERBDK if any are present.
?LBST	Left byte: label level. Supply 1, 2, 3, or 4; the default is 3. Right byte: number of user volume labels (tape only). Supply 1, 2, 3, ..., or 9.
?LBDV (doubleword)	Byte pointer to magnetic tape or diskette unit.
?LBVD (doubleword)	Byte pointer to volume identifier (volid).
?LBUV (doubleword)	Byte pointer to user volume labels (tape only). You can have a maximum of 76 bytes in these labels and you must separate the labels with the null character.
?LBOI (doubleword)	Byte pointer to owner identifier or -1 (if no identifier).
?LBAC	Left byte: reserved. (Set to 0.) Right byte: accessibility (access to tape).

*There is no default unless otherwise specified.

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

Input

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

Output

AC0	Undefined
AC1	Undefined
AC2	Address of the ?OPEN packet

Error Codes in AC0

EREO1	File is open, can't exclusively open
EREO2	File exclusively opened, can't open
ERFAD	File access denied
ERIFL	IAC (Intelligent Asynchronous Controller) failure
ERINP	IPC file not opened by another proc
ERIPS	Illegal pipe size
ERMPR	System call parameter address error
ERP00	Illegal pipe open option
ERVWP	Invalid word pointer passed as a system call argument

Who Can Use It?

There are no special process privileges needed to issue this call. You need Execute access to the file's directory. You also need Write access to the file's directory if issuing ?OPEN will create the file. And, you need Read access to the file and its directory if you are issuing ?OPEN against a file that presently exists.

What It Does

?OPEN opens a file or device for I/O and directs the operating system to assign it a unique channel number. A single process can open as many as 224 channels at the same time. Note that you cannot use ?OPEN to open a file for block I/O, physical block I/O, shared access, or to open a binary synchronous communications line.

You can use ?OPEN to create and then open a file. If you choose this creation option and you choose the default file–type specification, the operating system creates the file as a user data file. (User data files are not executable program files.)

The packets for ?OPEN, ?READ, ?WRITE, and ?CLOSE have the same structure, although not all offsets apply to every system call. You can specify the packet address as an argument to ?OPEN or load AC2 with the packet address before you issue ?OPEN. In both cases, the operating system returns the packet address in AC2.

?OPEN Continued

Figure 2–113 shows the structure of the ?OPEN packet, and Table 2–90 describes each offset and bit position in the packet. The accompanying text explains the options you can exercise in setting the packet specifications.

Figure 2–113 includes byte pointers to extensions for the basic I/O packet. These extensions direct the operating system to perform extended processing on the target file. See the following section "Packet Versions."

The following guidelines apply to the ?OPEN packet:

- You can default some specifications, such as record format, and use the values you set for them when you created the file. You can also alter some of the ?OPEN specifications, such as record format and buffer address, when you read to or write from the file.
- When you close and reopen a file, the operating system overwrites the default values in the packet. Thus, you must reset these values before you use the packet again.
- The parameters in the file specifications word (?ISTI) represent bit masks, not individual bits. To select more than one option for ?ISTI, OR the appropriate masks. For example, the ?ISTI specification ?IEXO!?OFIN opens the file for the exclusive use of the calling task for read purposes only.
- You must set all unused bits in offset ?ISTI to 0.
- The operating system returns the file's channel number to offset ?ICH in the I/O packet. (The operating system always sets the channel number, even if you place a value in ?ICH.)
- You need not set the offsets used by ?READ and ?WRITE to 0, because the operating system ignores them when it executes ?OPEN.

Packet Versions

There are three versions of the ?OPEN parameter packet as follows. Refer to Figure 2–113 as you read the following summaries of these three versions.

Standard — which includes offsets ?ICH through ?IDEL, and is ?IOSZ words long.

Short extended — which includes offsets ?ICH through ?ENET, and is ?IBLT words long. Set bit ?IPKL in offset ?ISTI to indicate the short extended packet.

Long extended — which includes offsets ?ICH through ?ENET and 24 reserved words for a length of ?ETMX words. Set bit ?IPKL in offset ?ISTI and bit ?IMP2 in offset ?ISTO to indicate the long extended packet. You should set the 24 reserved words to zero with one exception. The first doubleword, offset ?ETER, contains zero. The second doubleword, offset ?ETSN, contains 0. The third doubleword (offset ?EPIP, the exception) contains 1S0 and a word pointer to the optional pipe extension packet. (?OPEN uses the third doubleword when working with a pipe file.)

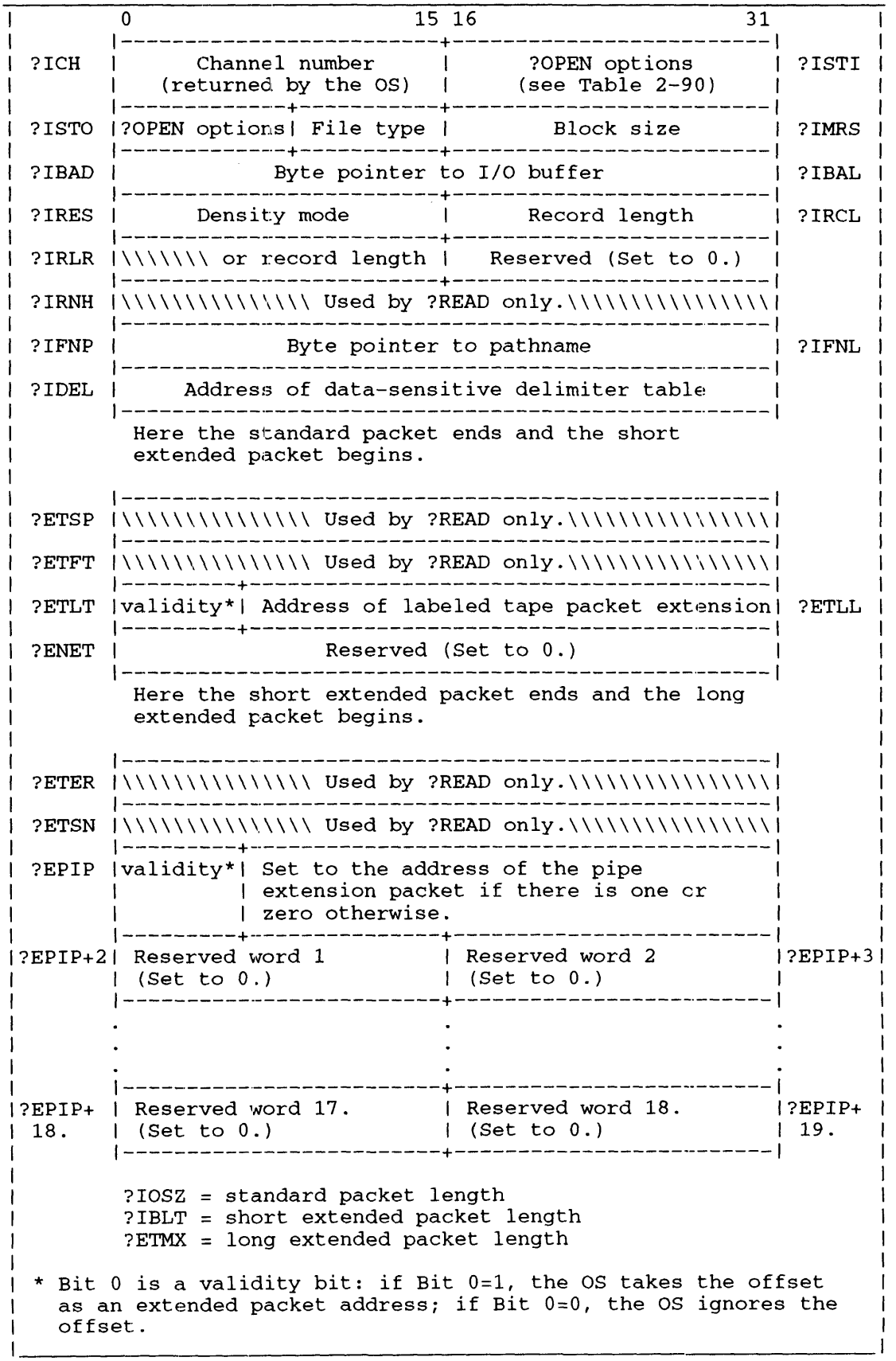


Figure 2–113. Structure of ?OPEN Packet

Table 2–90. Contents of ?OPEN Packet*

Offset	Contents
?ICH	Channel number (assigned by the OS).
?ISTI	File specifications word: Packet type ?IPKL--Extended packet. DEFAULT = 0 (no extended packet) Format select ?ICRF--Change format to that specified in record format field. DEFAULT = 0 (use record format specified in the ?CREATE packet). Absolute file pointer position (?READ/?WRITE only). Append ?APND--Open the file for appending. DEFAULT = 0 (set file pointer to first word in file). Binary I/O. Normally, only ?READ/?WRITE supplies a value here. However, if you issue ?OPEN against a queue type file (e.g., @LPT) and specify ?IBIN here, AOS/VS responds to this bit. Specifically, AOS/VS would do a binary queue submission to EXEC. Force output option (?READ/?WRITE only). Exclusive open ?IEXO--Permit no other task in this or any other process to open file until it has been closed. DEFAULT = 0 (nonexclusive open). Pipe files require a nonexclusive open.

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

Table 2–90. Contents of ?OPEN Packet*

Offset	Contents
?ISTI (continued)	Priority read ?PDEL--Open file for priority read. DEFAULT = 0 (file has normal priority). Creation option ?OFCE--If the file exists, open the file. If the file does not exist, create one and open it. ?OFCR--If the file exists, return an error. If the file does not exist, create one and open it. ?OFCR!?OFCE--If the file exists, then delete it, recreate it, and open it. If the file does not exist, then create it and open it. DEFAULT = 0 (if file does not exist, do not create it; return ?OPEN error). Input/output ?OFIN--Open for input. ?OFOT--Open for output. ?OFIO--Open for input/output. (Not available for pipe files.) Record format ?RTDY--Dynamic-length. ?RTDS--Data-sensitive. ?RTFX--Fixed-length. ?RTVR--Variable-length. ?RTUN--Undefined-length. ?RTVB--Variable block, variable record. IPC no wait option ?IIPC--Setting "IPC No Wait" on IPC file When you select ?IIPC, the process that initially opens the IPC file will not wait for the synchronization message from the partner process. Normally the process waits for the partner process to open the IPC file.
?ISTO	Left byte ?SHOP--Shared ?OPEN request. (Not available for pipe files.) ?IMFF--Inhibit initial form feed. ?IMP2--Long extended packet. ?IMHN--Hold nonpriority reads. DEFAULT = 0 (normal open; write initial form feed when opening line printers). If you specify none of the options ?SHOP through ?IMHN, you must specify 0 here.

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

Table 2–90. Contents of ?OPEN Packet*

Offset	Contents
?ISTO	Right byte File Type. (See Table 2-92) DEFAULT = 0 (if file exists, the OS ignores this parameter and uses the file type specified in the ?CREATE packet). If you are creating the file (you set ?OFCR and/or ?OFCE in ?ISTI), the file type is ?FUDF (user data file).
?IMRS	Physical block size (in Kbytes). Specify the length of the pipe in 2Kbyte (page) multiples, up to a 20 page maximum. 1 page is added to the pipe length at creation. DEFAULT = -1 (block size = 2 Kbytes)
?IBAD (doubleword)	Byte pointer to record I/O buffer. DEFAULT = -1 (deferred until you issue ?READ or ?WRITE).
?IRES	Density mode (for magnetic tapes only). Set this field to 0 for all other file and device types. ?IDAM--Automatic density matching. ?ID8--Density 800 bytes/inch. ?ID16--Density 1600 bytes/inch. ?ID62--Density 6250 bytes/inch. ?ID5--Low tape density. ?ID6--Medium tape density. ?ID7--High tape density. DEFAULT = 0 (use density mode specified during the system-generation procedure). Transfer mode (for Model 6352 magnetic tape units only) ?OMBFM--Use buffered mode when performing tape I/O. ?OMSTR--Use streaming mode when performing tape I/O. ?IMNTE--Use the emulation override. ?IMCOF--Data compression off. ?OIBM--Open as an IBM-labelled tape. ?OANS--Open as an ANSI-labelled tape. DEFAULT = 0 (open as an AOS/VS or AOS/RT32 tape).

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

The ?SHOP parameter causes ?OPEN to behave much like ?SOPEN. However, unlike ?SOPEN, ?OPEN with ?SHOP set involves the Agent. Therefore, to minimize system overhead, you should use ?SOPEN. Using ?SHOP shared I/O consumes more of the system I/O resources (than unshared I/O) and you may not be able to open as many channels.

Normally, the operating system generates a form feed when it opens a line printer. If you do not want this initial form feed, set bit ?IBFF in offset ?ISTO.

Other ?OPEN Offsets

Offset ?IBAD points to the I/O buffer. The I/O buffer is an area that you set up to receive the records you will later read or write. The I/O buffer must be large enough to accommodate the largest record you will read or write; if it is not, the operating system returns error code ERMPR when it tries to perform the I/O system call. If you do not specify an I/O buffer when you open the file, you must do so every time you issue ?READ or ?WRITE.

The meaning of offset ?IRCL varies, depending on the file's record format. If the file consists of fixed–length records, set ?IRCL to the record length in bytes. If the file consists of dynamic–length, variable–length or data–sensitive records, set ?IRCL to the number of bytes to be transferred (read or written) upon each I/O request. If the file consists of fixed–length records and ?IRCL contains –1, ?OPEN places the length of each record in the file in offset ?IRLR.

If the record type is data–sensitive and the operating system encounters a delimiter before it reaches the specified number of bytes, it terminates the I/O transfer. However, if the record type is data–sensitive and the operating system does not find a delimiter within ?IRCL bytes, it returns error code ERLTL ("line too long").

Setting a Delimiter Table

To define alternative delimiters for data–sensitive records, set up a delimiter table in your logical address space, and specify its address in the ?OPEN offset ?IDEL.

The delimiter table must consist of 16 consecutive 16–bit words, to form a table of 256 bits. Each bit in the table represents an ASCII character. Reading from left to right, the first bit (Word 0, Bit 0) represents the null character (0), the second bit (Word 0, Bit 1) represents the Ctrl–A character (octal 001), and so forth. For each bit you set, the operating system recognizes the corresponding character as a delimiter for the file's records.

Figure 2–114 depicts a sample delimiter table with bits set to make the null, carriage return (015 octal), and rubout (177 octal) characters data–sensitive delimiters.

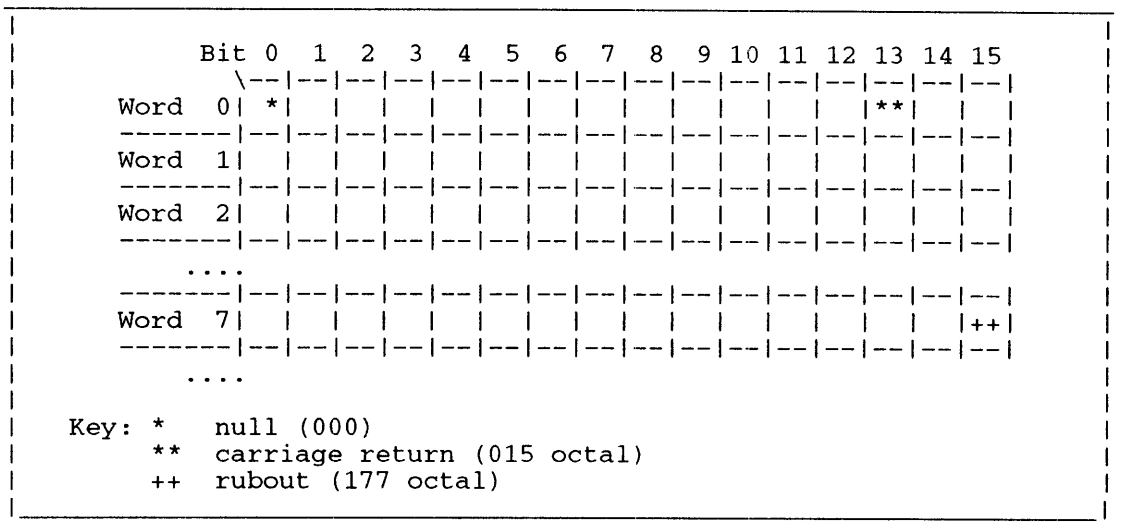


Figure 2–114. Sample Delimiter Table

You can also use the ?SDLM system call to set a delimiter table for an open record or open device. If you issue ?SDLM after you issue ?OPEN, the ?SDLM delimiter specifications override those in the ?OPEN packet.

Extension Packet for Pipes

You use one extension packet for pipes, and only ?OPEN provides the pipe extension packet. You supply the packet as an extension to the packet for a ?OPEN, ?READ, or ?WRITE system call. You must specify the pipe size in 2 Kbyte multiples in offset ?IMRS (see Table 2–90). The maximum pipe size is 20 pages (2 Kbytes equal one page), or you can use the default value of –1.

When the pipe file is created, the operating system adds a 1–page overhead to the size specified in ?IMRS. For a full description of using pipes, see *AOS/VS System Concepts* (093–000335).

Figure 2–115 contains the extension packet. Table 2–93 describes its contents.

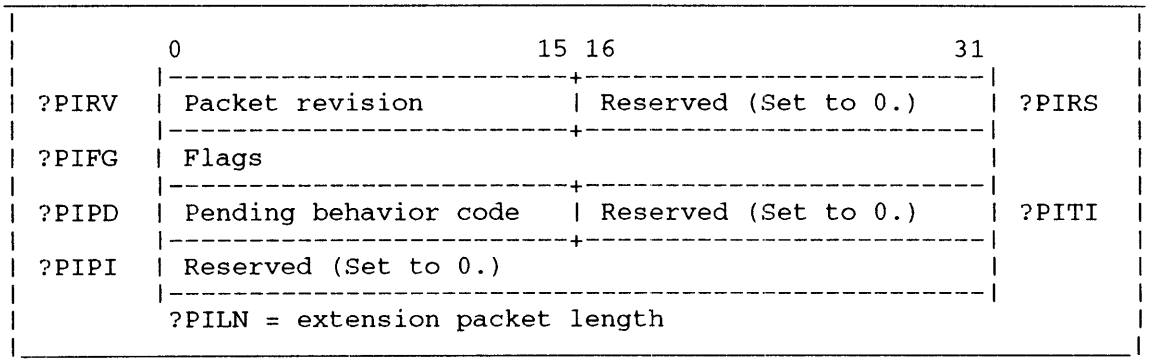


Figure 2–115. Structure of ?OPEN Extension Packet for Pipes

Table 2–93. Contents of ?OPEN Extension Packet for Pipes

Offset	Contents
?PIRV	Packet revision number. Place the value of symbol ?PKR0 here.
?PIRS	Reserved word. (Set to 0.)
?PIFG	Flag word. (Set to 0.)
?PIPD	Use the following symbols to specify the operating system's action. ?PALW -- Always pend on a read from an empty pipe or on a write to a full pipe, regardless of whether the pipe is one-ended or two-ended. ?PTWO -- Pend on a read from an empty pipe or on a write to a full pipe, but only if the pipe is two-ended. If the pipe is one-ended, then return error code EREOF on a read or else ERPFL on a write. ?PTWO is the default value of this offset. ?PNVR -- Never pend on a read from an empty pipe or on a write to a full pipe, regardless of whether the pipe is one-ended or two-ended. Return error code EREOF on a read or else ERPFL on a write.
?PITI	Reserved word. (Set to 0.)
?PIPI (double-word)	Reserved word. (Set to 0.)

Magnetic Tapes

Offset ?IMRS specifies the physical block size of a magnetic tape or disk file. The physical block size is the number of bytes, minus 1, in the largest physical block. The operating system uses this parameter to allocate a buffer ?IMPSTL bytes long for I/O. (The default buffer size is 2 Kbytes (4 blocks) for disk files.)

If you are opening a magnetic tape on a controller that supports multiple tape densities (such as a type MTB controller), you can set offset ?IRES to a specific density mode for the tape. There are seven density mode settings:

- ?IDAM directs the operating system to set the tape unit to the correct density mode automatically.
- ?ID8 sets the tape unit to a density mode of 800 bytes/inch.
- ?ID16 sets the tape unit to a density mode of 1600 bytes/inch.
- ?ID62 sets the tape unit to a density mode of 6250 bytes/inch.
- ?ID5 sets the tape unit to the lowest density it supports.
- ?ID6 sets the tape unit to the middle density on drives supporting three densities, and to the lower density on drives supporting two densities.
- ?ID7 sets the tape unit to the highest density it supports.

If you set offset ?IRES to 0 (the default), the operating system uses the density mode that was set during the system–generation procedure. However, if you set offset ?IRES to 0, you must make sure that the tape's density mode matches the default mode. Otherwise, ?OPEN fails after I/O is attempted.

There are two other possible density mode errors: ERCND (controller does not support this density mode), and ERITD (indecipherable tape density). The operating system returns ERITD when either the tape or the tape unit is damaged.

Labeled Magnetic Tapes

If you want to read or write additional user labels on a labeled tape file, you must open the file with the ?OPEN packet extension for labeled tapes. To do this, perform the following steps:

1. Set Bit ?IPKL in offset ?ISTI of the standard ?OPEN packet.

2. Set Bit 0 of offset ?ETLT to 1 (to indicate that there is a packet extension).

   The first time you use the labeled tape extension, you must set Bit 0 of ?ETLT (the flag bit) to 1 so that the system will recognize Bits 1 through 31 of offset ?ETLT as the address of the packet extension. The operating system sets the flag bit to 0 after it executes the ?OPEN.

   If you want to change one or more of the specifications, and then reuse the extension, be sure to reset the flag bit to 1. If the flag bit is 0, the operating system assumes that the packet extension has not changed and, therefore, does not re–examine it.

   You can set aside space for the labeled tape extension even if you do not supply values for it. In this case, the operating system simply fills the packet extension with the existing label information.

3. Set Bits 1 through 31 of offset ?ETLT to the address of the packet extension.

4. Precede the labeled tape extension with the following pseudo–op to reserve the correct number of words for the packet extension:

   ```
   .LOC  extended–pkt–address+?ELLN
   ```

Figure 2–116 shows the structure of the labeled tape extension.

If you create and open a labeled tape file with the ?OPEN creation option, the operating system treats the extension as follows:

- If you supplied values in the extension, the operating system uses those values.
- If you did not supply values in the extension, the operating system fills the packet extension with the default values listed in Table 2–94.

Mountstatus Command

Offset ?ZXNA of the main ?OPEX packet in Figure 2–124 must contain the byte address of the queue name buffer for the mountstatus function. To obtain all the mount queue names, use the ?XFNQN (get queue names) function of system call ?EXEC.

Figure 2–152 shows the structure of the mountstatus command subpacket, and Table 2–131 describes its contents.

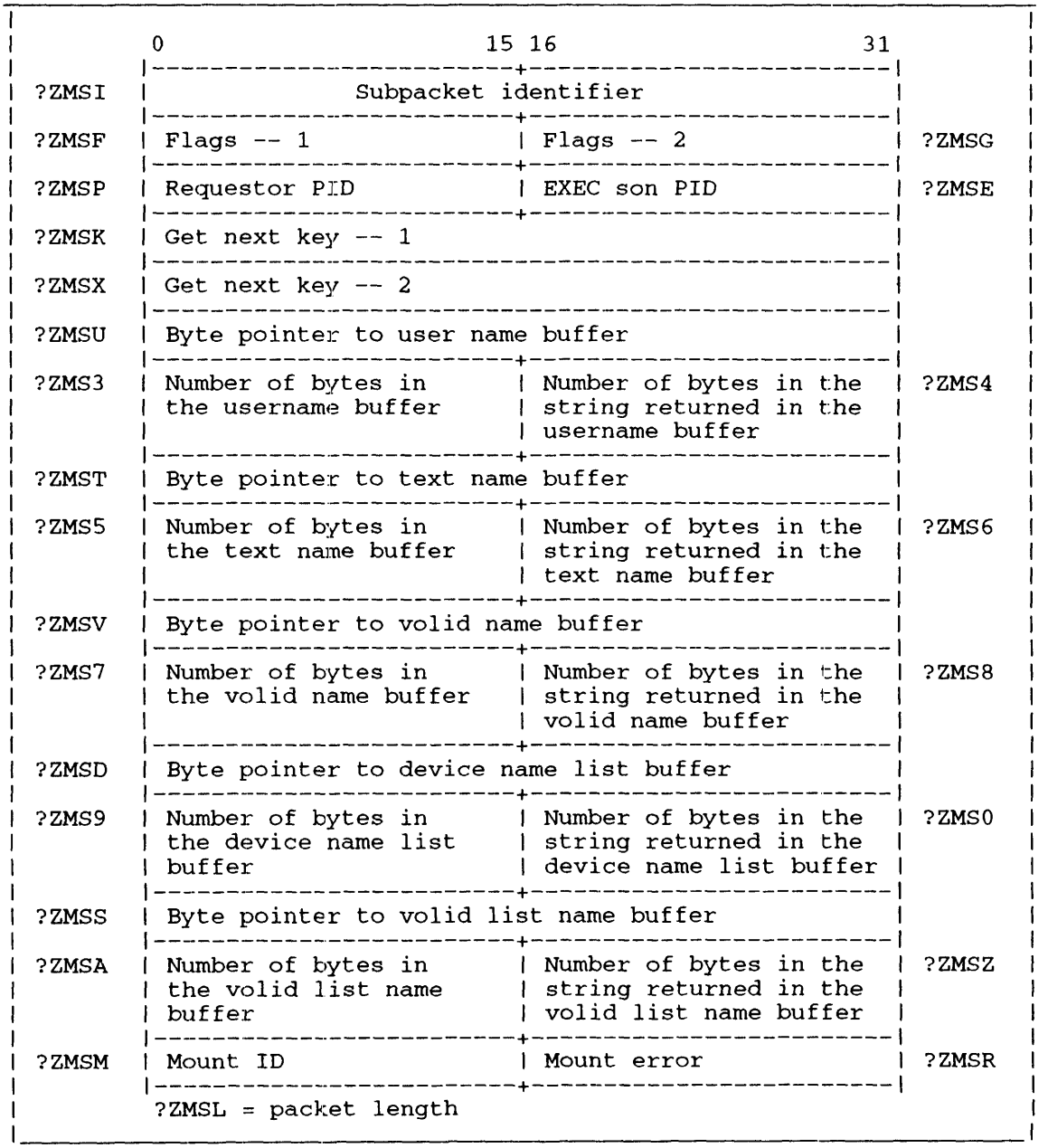


Figure 2–152. Structure of Mountstatus Command Subpacket

Table 2–131. Contents of Mountstatus Command Subpacket

Offset	Contents
?ZMSI (double-word)	Subpacket identifier. Place ?ZMSQ here.
?ZMSF	Flags word. It contains bit flags that indicate default choices to the operating system. You must set either bit ?ZY20 or bit ?ZY21. ?ZY20 = 1B(?ZZ20) -- You want the status of all requests, both inactive and active. ?ZY21 = 1B(?ZZ21) -- You want the status by mount ID. ?ZJ22 = 0B(?ZZ22) -- An output bit; explicit labeled mount request. ?ZY22 = 1B(?ZZ22) -- An output bit; implicit labeled mount request. ?ZY23 = 1B(?ZZ23) -- An output bit; unit mount request. ?ZY24 = 1B(?ZZ24) -- An output bit; waiting to be dismounted. ?ZY25 = 1B(?ZZ25) -- An output bit; next volume. ?ZY26 = 1B(?ZZ26) -- An output bit; mount error. ?ZY27 = 1B(?ZZ27) -- An output bit; specific volume. ?ZY28 = 1B(?ZZ28) -- An output bit; extend volid list. ?ZY29 = 1B(?ZZ29) -- An output bit; read only.
?ZMSG	Flags word. It contains bit flags that the operating system sets. ?ZY2G = 1B(?ZZ2G) -- An output bit; density is 800 bpi. ?ZY2H = 1B(?ZZ2H) -- An output bit; density is 1600 bpi. ?ZY2I = 1B(?ZZ2I) -- An output bit; density is 6250 bpi. ?ZY2J = 1B(?ZZ2J) -- An output bit; automatic density matching.
?ZMSP	The operating system returns the PID of the requestor.
?ZMSE	The operating system returns the PID of the immediate son of EXEC.
?ZMSK (double-word)	Get next entry key. The operating system uses this offset to hold the key indicator for EXEC.
?ZMSX (double-word)	Second get next entry key. The operating system uses this offset to hold the second key indicator for EXEC.

(continued)

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

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

Input

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

Output

AC0	Undefined
AC1	Byte count of the bytes read or written
AC2	Address of the packet

Error Codes in AC0

ERVWP Invalid word pointer passed as a system call argument

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

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

Why Use It?

You can use ?PRDB or ?PWRB to check for bad blocks on a disk, or to check for problems with an I/O device.

When the operating system encounters a transfer error (bad block) while it is executing a ?PRDB or a ?PWRB, it takes the normal return and reports the reason for the error in the packet, but transfers all or part of that bad block. (See Table 2–159.) However, when a device error occurs during a ?PRDB or a ?PWRB, the operating system immediately aborts the transfer and returns the device error code to the packet.

Who Can Use It?

There are no special process privileges needed to issue this call. You must have obtained a channel number via ?GOPEN or ?OPEN before issuing this call. Also, you have Read access to the file before issuing ?PRDB or Write access to the file before issuing ?PWRB.

What It Does

?PRDB and ?PRWB read and write physical blocks on disk, respectively. Before you issue one of these system calls, load AC1 with the channel number assigned to the target file at ?GOPEN time.

?PRDB and ?PWRB each require an extended packet. You can specify the packet address as an argument to the system call or as input to AC2 before you issue the system call. The parametric value for the length of the packet, including the extension, is ?PPBLT. Figure 2–178 shows the structure and Table 2–157 describes the contents of the ?PRDB/?PRWB packet. Table 2–158 lists the disk and tape types applicable to the offsets in the packet extension.

	0	15 16	31	
?PSTI	Status Word	Reserved (Set to 0.)		?PSTO
?PCAD	Word address of data buffer in your logical address space			
?PRNH	Block number of first block to be transferred			
?PRCL	Number of bytes in last disk or tape block transferred	Relative block number of last block transferred (disk) or running byte count (tape)		?PRBB
?PCS1	Controller status word 1	Controller status word 2		?PCS2
?PCS3	Controller status word 3	Controller status word 4		?PCS4
?PCS5	Controller status word 5	Controller status word 6		?PCS6
?PCS7	Controller status word 7	Controller status word 8		?PCS8

?PPBLT = packet length

Figure 2–178. Structure of ?PRDB / ?PWRB Packet

?PRDB/?PRWB Packet Offsets

Specify the tape or printing control options you want in the left byte of ?PSTI. Specify the number of blocks you want to read or write in the right byte of offset ?PSTI (block count). Use offset ?PCAD as a word pointer to the address of the data buffer you have reserved in your logical address space for the transfer. Offset ?PRNH must indicate the relative block number of the first block you want to transfer.

Use offset ?PRCL for ?PWRB (write block) only. Offset ?PRCL must contain the number of bytes in the last block you are writing. In effect, this value specifies the last valid byte in the block. If you are performing a write (?PWRB) to extend the file, the operating system places the end–of–file mark immediately after this byte. If you set offset ?PRCL to 0, the operating system sets the bytes in the last block to the default, which is 512 (a full block).

As Table 2–157 and Table 2–158 indicate, the operating system uses the ?PRDB/?PRWB packet extension to return status information about the block transfer. (You supply input values only for offsets ?PSTI through ?PRCL in the main packet.)

Table 2–157. Contents of ?PRDB/?PWRB Packet

Offset	Input Value	Output Value
?PSTI	Left Byte: Input options. ?IMIO = 1B(?IDIO) Use direct I/O with the MCA Protocol. The system ignores the block count, and transmits the number of bytes in ?PRCL. ?ENOV = 1B(?ENOR) For printers, enable a Vertical Form Unit load (part of Forms Control Utility). For tape, override the LEOT (logical end-of-tape) mark. ?SAFM = 1B(?SAFE) Enable a safe write to tape.	Unchanged.
	Right Byte: Block count.	Unchanged.
?PSTO	Reserved.	Unchanged.
?PCAD (doubleword)	Address of data buffer in your logical address space.	Unchanged.
?PRNH (doubleword)	Block number of first block to be transferred.	Unchanged.
?PRCL	Number of bytes in last disk or tape block transferred.	Unchanged.
?PRBB	Not applicable.	Number of blocks successfully transferred (disk) or running byte count (tape).
?PCS1 - ?PCS8	Not applicable.	Controller status words. (See Table 2-158.)

You can use offset ?PRBB to determine if an error was encountered, and if one was, the block number of the first bad block encountered during the transfer. Offset ?PRBB records the number of blocks that were successfully transferred. If ?PRBB does not contain the number you specified in offset ?PSTI, an error has occurred and offsets ?PCS1 through ?PCS8 in the packet extension record the reason for the transfer error.

Offsets ?PCS1 and ?PCS5 are always set on output, as appropriate for the device in use. Note that offsets ?PRBB through ?PCS5 are the only packet extension parameters currently in use.

Table 2–158. ?PRDB/?PWRB Packet: Controller Status Words

DPD/DPG Disk	Status Word	Use
6030 (Floppy) 6045/6050/6051 (10 Mbytes) 6070 (20 Mbytes)	?PCS1	DIA (Transfer) Status
DPI Disk	**Status Word**	**Use**
6097 (Floppy) 6098/6099 (12.5 Mbytes) 6100/6103 (25 Mbytes) 6225 (5 Mbytes) 6227 (15 Mbytes) 6234 (50 Mbytes)	?PCS1	DIA (Transfer) Status
DKB Disk	**Status Word**	**Use**
6063/6064/6065/6066 (Fixed Head)	?PCS1 ?PCS2 ?PCS3 ?PCS4	DIA (Transfer) Status Unused ERCC Word One ERCC Word Two
DPF Disk	**Status Word**	**Use**
6060 (96 Mbytes) 6061 (147 Mbytes) 6067 (50 Mbytes) 6122 (277 Mbytes) 6160 (73 Mbytes) 6161 (147 Mbytes) 6214 (602 M bytes)	?PCS1 ?PCS2 ?PCS3 ?PCS4	DIA (Transfer) Status DIB (Drive) Status ERCC Word One ERCC Word Two
DPM Disk	**Status Word**	**Use**
4514 (5 1/4 in. Floppy)	?PCS1	DIA (Transfer) Status
DPJ Disk	**Status Word**	**Use**
6236/6237 (354 Mbytes) 6239/6240/ 6290/6350 (592 Mbytes) 6309 (5 1/4 in. Floppy) 6310 (38 Mbytes) 6328 (70 Mbytes) 6329 (120 Mbytes)	?PCS1 ?PCS2 ?PCS3 ?PCS4 ?PCS5	CB Status Unit Status CB Error Status Disk Error Code Controller Type

(continued)

Table 2–158. ?PRDB/?PWRB Packet: Controller Status Words

MTB/MTD Tape	Status Word	Use
6026 (800/1600 bpi) 4307 (1600/6250 bpi) 6200 (1600/6250 bpi) 6300 (1600/6250 bpi)	?PCS1 ?PCS2	DIA (Transfer) Status DIC (Drive) Status
MTC Tape	**Status Word**	**Use**
6125 (1600 bpi) 6231/6311 (Cartridge)	?PCS1	DIA (Transfer) Status
MRC Disk	**Status Word**	**Use**
6236/6237 (354 Mbytes) 6239/6240/6290 (592 Mbytes) 6357/6398/ 6399/6400 (862 Mbytes) 6581/6582/6584 (500 Mbytes)	?PCS1 ?PCS2 ?PCS3 ?PCS4 ?PCS5 ?PCS6	Controller Type Unit Type Controller Error Code Error Status (ctrl/unit) Unit Status Drive Error Code
MRC Tape	**Status Word**	**Use**
6299/6300 (6250/1600 bpi) 4307-TL (6250/1600/800 bpi)	?PCS1 ?PCS2 ?PCS3 ?PCS4 ?PCS5 ?PCS6	Controller Type Unit Type Controller Error Code Error Status (ctrl/unit) Unit Status Drive Error Code

(concluded)

Table 2–159. Error Reports Returned in ?PRDB/?PWRB Offsets

Offsets	Contents	Meaning
?PSTI ?PRBB	20 20	The operating system transferred all blocks without error.
?PSTI ?PRBB	20 19	Last block was not transferred successfully. You might want to try again.

Notes

- Refer to the *ECLIPSE® MV/Family (32-Bit) Systems Principles of Operation* manual for details on the DIA, DIB, and DIC I/O instructions. Or, refer to the programmer's reference manual for your particular model.

?PRI
error return
normal return

Input

AC0	Priority number you wish to assign to the calling task
AC1	Reserved (Set to 0.)
AC2	Reserved (Set to 0.)

Output

AC0	Unchanged
AC1	Undefined
AC2	Undefined

Error Codes in AC0

No error codes are currently defined.

Why Use It?

?PRI is analogous to ?IDPRI, except that it changes the priority of the calling task, rather than another target task. Like ?IDPRI, ?PRI gives you some control over the operating system's task–scheduling activities. For example, you might use ?PRI to favor the calling task by assigning it a higher relative priority or, conversely, to favor other tasks by giving the calling task a lower relative priority.

Who Can Use It?

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

What It Does

?PRI changes the priority number of the calling task to the priority that you specify in AC0. If there are already other tasks at the new priority level, the calling task ranks last in the priority queue and, therefore, gains CPU access after all others at that priority level. Like ?IDPRI, ?PRI can cause immediate task rescheduling.

Before you issue ?PRI, load AC0 with the priority number that you want to assign to the calling task. Priority numbers for tasks range from 0 (the highest priority level) through 255 (the lowest priority level). If you specify a priority number greater than 255, the operating system truncates the priority number to its least significant 8 bits.

Who Can Use It?

Although there are no special process privileges needed to issue this call, many options affect the privileges the newly created process has. For example, setting bit ?PFDA in offset ?PFLG prevents the operating system from passing the father's default ACL to the son. These options appear throughout the rest of this explanation of ?PROC.

You must have Execute access to the directory in which the program file resides along with both Execute and Read access to the program file itself.

What It Does

?PROC creates a process and assigns it whatever characteristics you specify in the ?PROC packet. Before you issue ?PROC, set up a packet in your logical address space that is ?PLTH words long. You can either cite the packet address as an argument to ?PROC, or you can load the packet address into AC2 before you issue ?PROC. Figure 2–179 shows the structure of the ?PROC packet and Table 2–160 defines each offset and mask.

Among other characteristics, ?PROC defines the process's type and priority, its maximum and minimum working set size, and its creation privileges (that is, the right to create sons with additional ?PROC system calls). Note that you cannot create a son process with privileges the calling process does not have.

Note also that when the operating system terminates a father process, it simultaneously terminates all his sons.

The ?PROC extension packets let you specify the following items when you issue ?PROC:

- CPU selection.
- User locality.
- ?PROC–related username.
- ?GROUP access control list (AOS/VS II only).

Figures 2–180 and 2–181 show the structure of the ?PROC extension packets, and Tables 2–162 and 2–163 define each offset.

Offset ?PFLG

Offset ?PFLG in the packet contains specification bits for the following characteristics and privileges:

- Process type.

  Bit ?PFRS assigns the new process resident status; bit ?PFRP assigns it pre–emptible status. (Swappable is the default process type.) Bits ?PFRS and ?PFRP are mutually exclusive; if you set them both, the operating system takes the ?PROC error return and passes error code ERPTY to AC0.

- Default access control (AOS/VS only).

  Bit ?PFDA, the access privileges bit, prevents the operating system from passing the father's file access privileges to the son. But, bit ?PFDA is only checked by the operating system if a username is specified, and if ?PUNM does *not* contain –1. If you set ?PFDA, the operating system assigns the son the default access control list USERNAME, OWARE (Owner, Write, Append, Read, Execute access). The son can also receive USERNAME OWARE access under the following conditions:

  – When you assign the son a different username from its father.

  – When the father process has no default access privileges.

  For more information on default file access, see the description of ?DACL in this chapter.

- Control directive.

  Bit ?PFDB directs the operating system to pass control to the Debugger utility immediately after it creates the new process. By default, control passes to the start of the program associated with the new process.

- Concurrency.

  Bit ?PFEX blocks the calling process while the new process executes. If you don't set this bit, the son runs concurrently with its father. The calling process must have privilege ?PVEX to allow this. Under AOS/RT32, all processes have this privilege.

- Privileges mask.

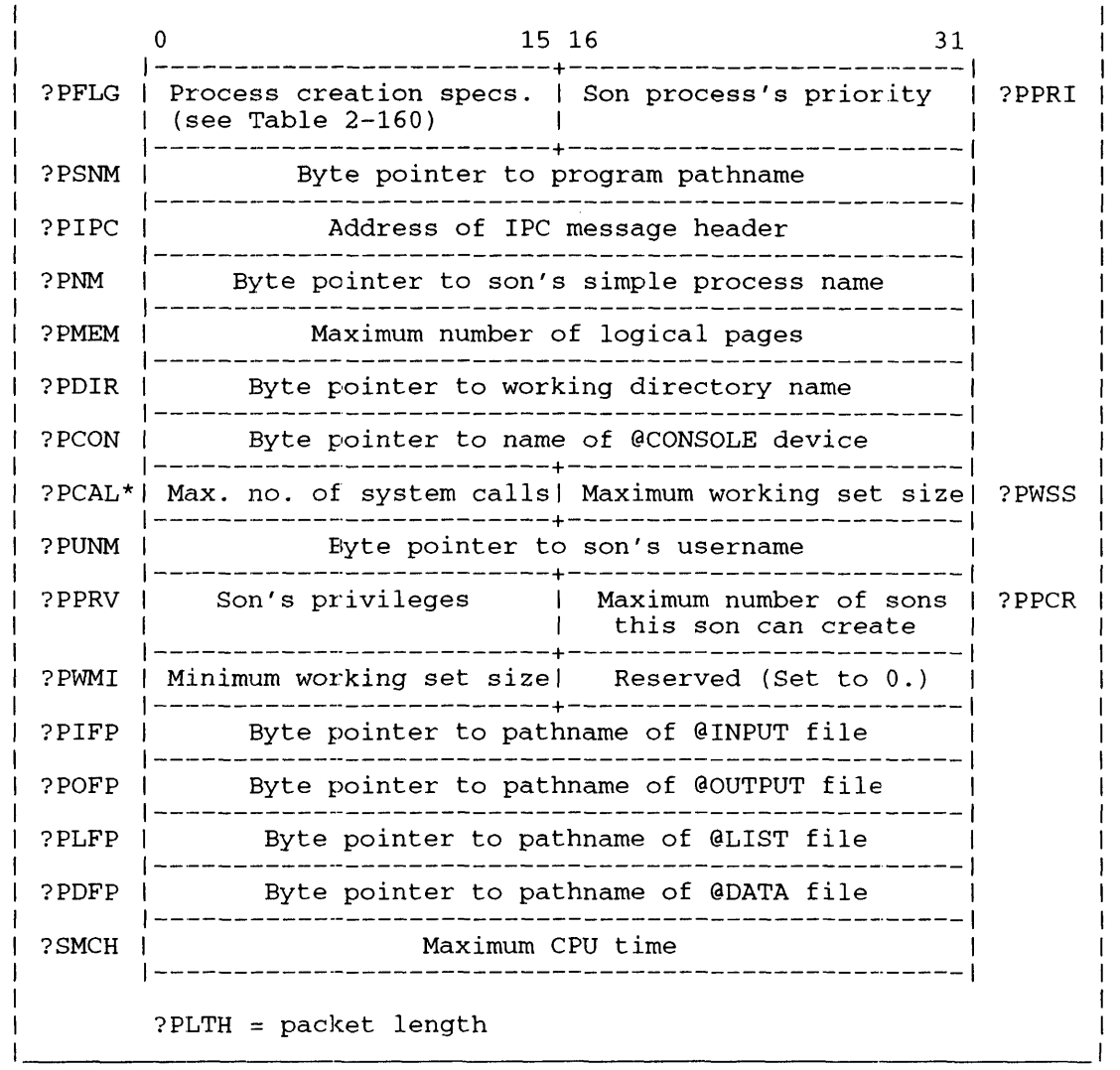
  Bit ?PFPM, the privileges mask, directs the operating system to give the son all the father's privileges except those specified in offset ?PPRV. Alternatively, if you set ?PFPM and set ?PPRV to –1, the son gets no privileges. If you just want the son to inherit the father's privileges, do NOT set ?PFPM, but set ?PPRV to –1.

- Break files and memory dumps.

  On an AOS/VS system, setting ?PBRK and not ?PDMP creates a break file if the process traps. Not setting ?PBRK and setting ?PDMP creates a dump file of Ring 7 if the process traps. Setting both bits has exactly the same effect as not setting ?PBRK and setting ?PDMP.

  On an AOS/RT32 system, all of the three combinations in the previous paragraph have the same effect. The effect is to create one break file with a memory dump of Ring 7. The filename has the format ?PID.TIME.7.BRK where PID and TIME are as described in Table 2–13 under offset ?ENFNP. The operating system creates another break file for each additional user ring (4, 5, 6) that is defined. A possible filename in this additional case is ?00035.11_41_23.6.BRK.

Table 2–161 describes each of the privilege bits in offset ?PPRV.



* This offset differs between AOS/VS and AOS/RT32. See Table 2-160.

Figure 2–179. Structure of ?PROC Packet

Table 2–160. Contents of ?PROC Packet*

Offset	Contents
?PFLG	Process creation specifications. ?PFPP--Reserved - set to 0. Control directive. ?PFDB--Control goes to the Debugger. DEFAULT = 0 (control goes to the starting address of the program file). Concurrency. ?PFEX--Caller is blocked while son executes. DEFAULT = 0 (son runs concurrently with the ?PROC caller). Privileges mask. ?PFPM--Mask son's privileges. DEFAULT = 0 (son process receives all privileges selected in ?PPRV). Access privileges. ?PFDA--Do not pass default ACL. ?PFDA is only checked if a username is specified, and ?PUNM does not contain -1. DEFAULT = 0 (son has same default ACL as father (the caller)). Break file. ?PBRK--Create a break file if process traps or terminates fatally. DEFAULT = 0 (do not create a break file if the process traps or terminates fatally). Block the son process. ?PFBS--Block son. DEFAULT = 0 (do not block the son). Extension packet. ?PFXP--An extension packet exists. DEFAULT = 0 (no extension packet exists).

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

Table 2–160. Contents of ?PROC Packet*

Offset	Contents
?PFLG (continued)	Process type ?PFRP--pre-emptible. ?PFRS--resident. DEFAULT = 0 (son is a swappable process). Process traps. ?PDMP--Dumps Ring 7. (If ?PBRK is also set, overrides ?PBRK.)
?PPRI	Priority of the son process. DEFAULT = -1 (same as caller's priority).
?PSNM (doubleword)	Byte pointer to pathname of program file for the new process to execute.
?PIPC (doubleword)	Address of an IPC message header; this message is sent to the new process. DEFAULT = -1 (no message header).
?PNM (doubleword)	Byte pointer to new process's simple process name. DEFAULT = -1 (Son's simple process name is the ASCII representation of its PID (five digits, including leading zeros).)
?PMEM (doubleword)	Maximum number of logical pages in new process. DEFAULT = -1 (same logical page maximum as the caller's).
?PDIR (doubleword)	Byte pointer to name of son's working directory; if 0, then use the caller's current working directory. DEFAULT = -1 (same initial working directory as the caller's).

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

Table 2–160. Contents of ?PROC Packet*

Offset	Contents
?PCON (doubleword)	Byte pointer to name of new process's @CONSOLE file; if 0, there is no @CONSOLE file. DEFAULT = -1 (same @CONSOLE device as caller).
?PCAL	Maximum number of system calls the son can issue concurrently. DEFAULT = -1. (Son is limited to two concurrent system calls.) AOS/RT32 ignores this offset; set to -1.
?PWSS	Maximum working set size for son. DEFAULT = -1 (no limit on working set size; otherwise, caller's limit). NOTE: The calling process must have the ?PVWS privilege to set this parameter.
?PUNM (doubleword)	Byte pointer to son's username. DEFAULT = -1 (same username as the caller's). NOTE: If ?PUNM contains -1 and a username is specified, ?PFDA is also checked.
?PPRV	Son's privileges; each bit in this offset (Table 2-161) assigns a privilege if ?PFPM is not set, or denies a privilege if ?PFPM is set. ?PFPM is a flag in offset ?PFLG (described earlier in this table). DEFAULT = -1. (If ?PFPM is also set, son has no privileges; if ?PFPM is not set, son has all the caller's privileges.)
?PPCR	Maximum number of sons this son can create; if 0, no sons. DEFAULT = -1 (same number of sons as the caller's (minus the number already created)).

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

Table 2–160. Contents of ?PROC Packet*

Offset	Contents
?PWMI	Minimum working set size. DEFAULT = -1 (no working set minimum; otherwise, caller's minimum). NOTE: The calling process must have ?PVWS privilege to set this parameter.
?PIFP (doubleword)	Byte pointer to pathname of son's @INPUT file; if 0, there is no @INPUT file. DEFAULT = -1 (same @INPUT file as the caller).
?POFP (doubleword)	Byte pointer to pathname of son's @OUTPUT file; if 0, there is no @OUTPUT file. DEFAULT = -1 (same @OUTPUT file as the caller).
?PLFP (doubleword)	Byte pointer to pathname of son's @LIST file; if 0, there is no @LIST file. DEFAULT = -1 (same @LIST file as the caller).
?PDFP (doubleword)	Byte pointer to pathname of son's @DATA file; if 0, there is no @DATA file. DEFAULT = -1 (same @DATA file as the caller).
?SMCH (doubleword)	Maximum CPU time allotted to the son. DEFAULT = -1 (Son receives remainder of father's time limit); if -1 and if father has no time limit, son has no time limit.

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

Table 2–161 describes each of the privilege bits in offset ?PPRV.

Table 2–161. Privilege Bits in Offset ?PPRV

Privilege	Meaning
?PVPC	The new process can create an unlimited number of sons.
?PVWS	The new process can ?PROC sons of a different program file type (that is, 16-bit or 32-bit program file).
?PVEX	The new process can remain unblocked while its sons execute.
?PVWM	The new process can define its sons' working set limit.
?PVPR	The new process can either assign itself a higher priority by issuing ?PRIPR or it can ?PROC a son with a higher priority than itself.
?PVTY	The new process can change its own process type (?CTYPE) or ?PROC a son with a different process type. (By default, sons have the same process types as their fathers.)
?PVUI	The new process can assign its sons different usernames.
?PVDV	The new process can define and access user devices.
?PVIP	The new process can issue the primitive IPC system calls ?ISEND and ?IS.R. (Note that connected processes, i.e., customer/servers, do not need this privilege to issue ?IS.R.)
?PVSM	The process can become System Manager.
?PVSU	The new process can issue ?SUSER to enter Superuser mode.
?PVSP	The new process can issue ?SUPROC to enter Superprocess mode.
?PVPP	The new process can be a peripheral process.

Offset ?PIPC

Offset ?PIPC points to an IPC message header. Frequently, you may want to send a CLI–format command line as the IPC message. This way, the son process can access arguments and switch values via a ?GTMES call. The explanation of system call ?GTMES explains a CLI–format line. In Appendix A sample programs BOOMER, DLIST, and TIMEOUT illustrate the ?GTMES call. Here's how to create such an IPC message:

- Set the system flag word (offset ?ISFL) to zero in the IPC message (as opposed to setting a bit if you were chaining to another process).
- Set bit ?RFCF in the user flag offset (?IUFL) to indicate CLI format.
- Set destination and origin port numbers to 0.
- Set offset ?ILTH to the word length of the message.
- Set offset ?IPTR to the word address of the message.

See Figure 2–83 for the structure of the ?ISEND header.

Suppose you issue ?PROC with offset ?PIPC containing the address of an IPC message header, and the target process will read the IPC message via ?GTMES. Then, offset ?IUFL of the IPC message header must contain ?RFCF to specify CLI format. Figure 2–83 and Table 2–60 describe the IPC message header; Figure 2–82 describes offset ?IUFL.

Other Offsets

Offset ?PPRI specifies a priority number for the new process, within the range 1 through 511. When you default this parameter (set offset ?PPRI to −1), the new process assumes its father's priority number. A father process must have privilege ?PVPR to assign to its son a higher priority than its own.

Offset ?PMEM specifies the maximum number of logical pages the new process can contain. By default, fathers and sons have the same logical page limit.

Offset ?PCAL specifies the maximum number of concurrent system calls the new process can issue. In general, critical processes should have higher ?PCAL parameters than less critical processes. Note that ?PCAL, and all specifications in ?PROC, should match the new process's needs as closely as possible.

Creating Offspring

The system determines the number of offspring that a process can create by checking its ?PROC packet for the following:

- The ?PVPC privilege in offset ?PPRV. This privilege allows the new process to create an unlimited number of sons. ?PVPC overrides the value in offset ?PPCR, described next.
- The ?PPCR offset which specifies the maximum number of offspring the new process can create. This offset is cumulative, which means that if a process with a ?PPCR value of 10 creates 2 sons, each with a ?PPCR value of 4, the original process cannot create any more sons, because 2 sons plus 8 (2*4) potential grandsons equals 10. This example is true as long as the 2 sons actually create 8 processes. If they actually create only 5 processes between them, the original process can create at most 3 processes (2 + 5 + 3 = 10).

- The ?PVEX bit in offset ?PPRV and the ?PFEX mask in offset ?PFLG. ?PVEX ensures that the new process remains unblocked while its sons execute (which allows it to create other sons). ?PFEX results in the ?PROC caller being blocked while the new process executes.

When a process which lacks the ?PVPC privilege tries to create a son, the operating system will create the son if

- The number of sons and their combined ?PPCR count (maximum number of sons which can be created) is less than or equal to the caller's ?PCPR value. (If the sum is more, the operating system signals an error.)
- Bit ?PVEX is set. (This lets the new process remain unblocked while its sons execute.)

Setting Maximum CPU Time

Offset ?SMCH specifies the maximum number of CPU milliseconds you want to assign the new process and all its subordinates. You must express this parameter as a double–precision unsigned integer. When a process exceeds its CPU time limit, the operating system terminates that process and returns error code ERTLM to its father.

When a process with a CPU time limit creates a son process, the operating system blocks the father and limits the son to the father's remaining CPU time. You can assign the son less than the father's remaining CPU time, but not more. If you assign more, the system takes the ?PROC error return and returns error code ERSMX (error in setting CPU time). As soon as the son terminates, the father receives whatever CPU time remains.

If you set offset ?SMCH to −1, the son acquires the rest of the father's CPU time. If you set offset ?SMCH to 0, the son has no CPU time limit, unless the father has a time limit. If the father has a time limit, the son gets the father's remaining time.

Setting the Working Set Size

Offsets ?PWMI and ?PWSS specify the new process's minimum and maximum working set size, respectively. Both values relate to the number of physical pages the process needs to execute efficiently. To set these values, the calling process must have privilege ?PVWS.

By default, the operating system adjusts the working set parameters dynamically, based on the process's type, page fault history, and general system overhead. When you set ?PWMI and ?PWSS, you direct the operating system to keep the working set within specific limits.

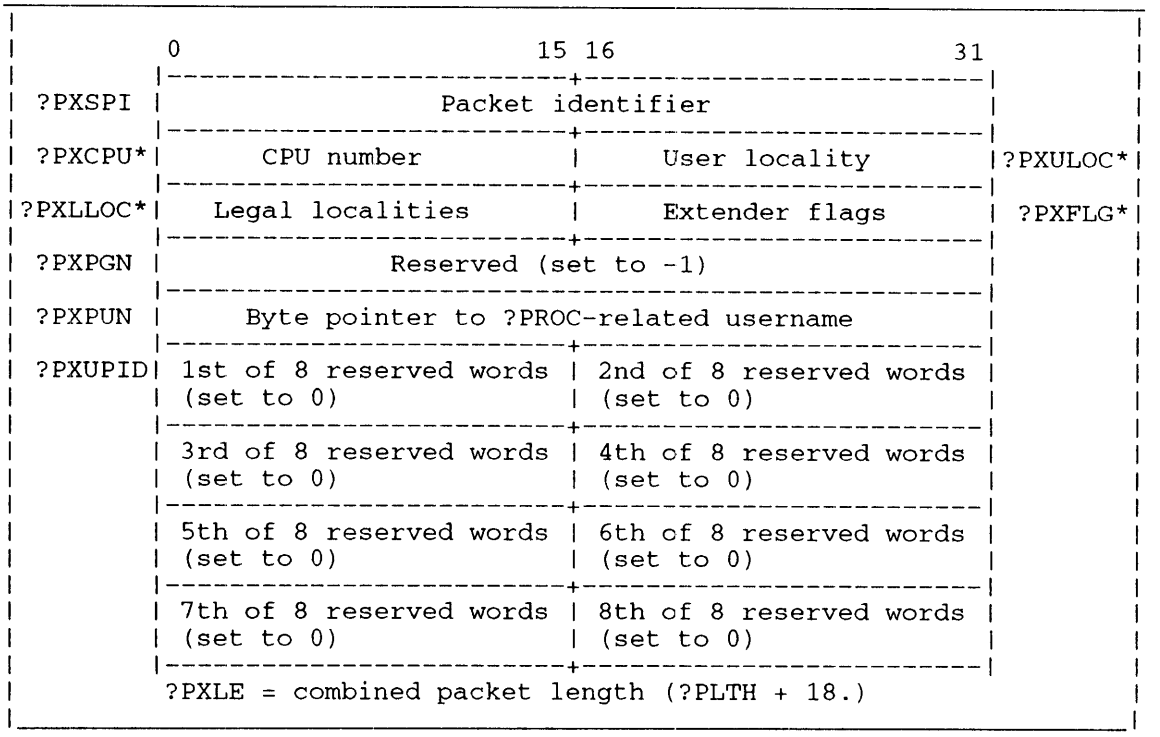
A process's working set size does not necessarily relate to the size of its logical address space. In fact, a larger process can require fewer physical pages than a smaller one if the larger process localizes its references and uses its databases efficiently. If you do set the working set size, take care not to set the maximum too low, because doing so increases paging I/O and processing time. Also, take care not to set the minimum working set size too high, because doing so can reduce efficient memory usage throughout your system.

CAUTION: *Generally, you will hurt the performance of the system if you define the size of a process's working set. Use this option with care; most of the time you'll be better off taking the default and allowing the operating system to make the necessary adjustments.*

Extension Packet

You specify the existence of an optional extension packet by setting bit ?PFXP in offset ?PFLG of the main parameter packet. Then place the extension packet itself immediately after the end of the main parameter packet. The length of the combined packet is ?PXLE words.

Figure 2–180 contains the structure of the ?PROC extension packet for AOS/VS and AOS/RT32 and Table 2–162 describes its contents. Figure 2–181 contains the structure of the ?PROC extension packet for AOS/VS II and Table 2–163 describes its contents.



* This offset differs between AOS/VS and AOS/RT32. See Table 2-162.

Figure 2-180. Structure of ?PROC Extension Packet for AOS/VS and AOS/RT32

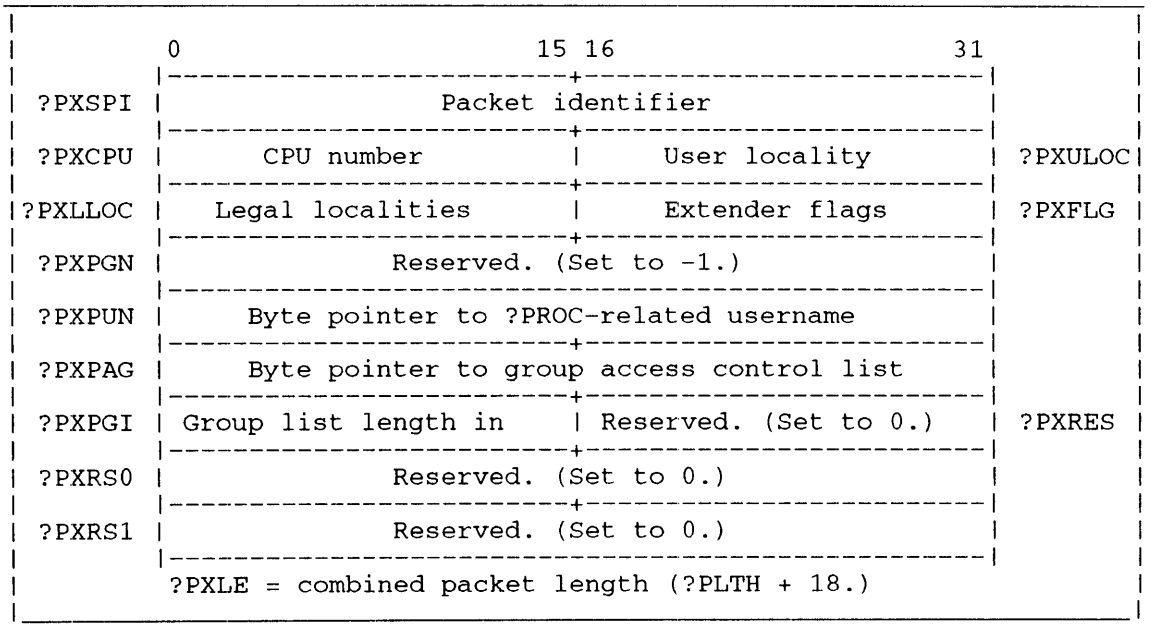


Figure 2-181. Structure of ?PROC Extension Packet for AOS/VS II

In Table 2–162, the column heads "VS" and "RT" represent AOS/VS and AOS/RT32, respectively. The entries in these columns have the following codes:

—	The operating system does not define a value for the given offset.
[no entry]	The operating systems handle the offset in the sameway.

Table 2–162. Contents of ?PROC Parameter Packet Extension for AOS/VS and AOS/RT32

Offset	VS	RT	Description
?PXSPI (dbl wd)			Packet identifier. Place 0 here.
?PXCPU		---	CPU number. Place here the ID of the CPU on which this process executes. AOS/RT32 does not use this value; supply -1.
?PXULOC		---	Place here the user locality as you have defined it. AOS/RT32 does not use this value; supply -1.
?PXLLOC		---	Legal user localities. This offset indicates the legal user localities that the new process may change to. The format is a bit map. If a bit is set, the new process may change to the corresponding user locality. Note that this format is the same one that ?PROFILE returns, so you can take the legal user localities from the profile and insert them directly into this extension packet. AOS/RT32 ignores this offset; supply -1.
?PXFLG		---	Extender flags word. This contains bit ?PFLL. If the bit is set, AOS/VS uses a default value for the son's legal locality. In other words, AOS/VS would NOT refer to offset ?PXLLOC to get the son's locality. If the bit is not set, AOS/VS obtains the son's locality from offset ?PXLLOC. AOS/RT32 does not use this information; supply -1.
?PXPGN (dbl wd)			Reserved. (Set to -1.)
?PXPUN (dbl wd)			Byte pointer to the ?PROC-related username that the new process will be created with, but only if the process has Superprocess privilege and has turned it on. Otherwise, supply -1. Normally only Data General software such as EXEC supplies a byte pointer here.
?PXUPID (8 words)			Reserved. (Set these 8 words to 0.)

Table 2–163. Contents of ?PROC Parameter Packet Extension for AOS/VS II

Offset	Description
?PXSPI	Packet identifier; ?PXSID defines the packet for AOS/VS II use.
?PXCPU	CPU number. Place the ID of the CPU on which this process executes.
?PXULOC	Specify the user locality.
?PXLLOC	Legal user localities. This offset indicates the legal user localities to which the new process may change. The format is a bit map. If a bit is set, the new process may change to the corresponding user locality. Note that this format is the same one that ?PROFILE returns, so you can take the legal user localities from the profile and insert them directly into this extension packet.
?PXFLG	Extender flags word. This word contains bit ?PFDLL. If the bit is set, AOS/VS uses a default value for the son's legal locality. In other words, AOS/VS II would NOT refer to offset ?PXLLOC to get the son's locality. If the bit is not set, AOS/VS II obtains the son's locality from offset ?PXLLOC.
?PXPGN	Reserved. (Set to 0.)
?PXPUN	Byte pointer to the ?PROC-related username that the new process creates, but only if the process has Superprocess privilege turned on. Otherwise, supply -1. Normally only Data General software such as EXEC supplies a byte pointer here.
?PXPAG	Byte pointer to the group access control list. A group access control list is a double, null-terminated list of group names. A group name is a null-terminated, ASCII string with a maximum length of 16 bytes, including the null character. A group name corresponds to a filename in the :GROUPS directory. If you specify -1, use the parent's group access control list. If you specify 0, use a null group access control list. If you call ?PROC for AOS/VS II with no extension packet or with the AOS/VS and AOS/RT32 extension packet, the call behaves as if you specified a -1 in ?PXPAG (process inherits its parent's list of groups).
?PXPGI	Length of the group access control list.
?PXRES ?PXRES0 ?PXRES1	Reserved. (Set to 0.) Reserved. (Set to 0.) Reserved. (Set to 0.)

?PFFD — (doubleword) field descriptor, continued.

- ?PBATCHP — batch priority, 1 word.
- ?PCNSPRV — terminal usage privilege, 1 bit.
- ?PBCHPRV — batch usage privilege, 1 bit.
- ?PMODPRV — modem usage privilege, 1 bit.
- ?PVCNPRV — virtual terminal usage privilege, 1 bit (negative logic).
- ?PRRAPRV — remote resource access privilege, 1 bit (negative logic).
- ?PPWDPRV — change password privilege, 1 bit (negative logic).
- ?PNCRYPT — system password encrypted indicator, 1 bit.
- ?PMGSYS — system manager privilege, 1 bit.
- ?PCOMMNT — user comment, 80 bytes including a null byte.
- ?PBWSS — batch working set size, 2 words; the internal format is

  word 0 — maximum
  word 1 — minimum

- ?PBLMEM — batch logical memory, 2 words; the internal format is

  word 0 — low order
  word 1 — high order

- ?PNBWSS — nonbatch working set size, 2 words; the internal format is

  word 0 — maximum
  word 1 — minimum

- ?PNBLMEM — nonbatch logical memory, 2 words; the internal format is

  word 0 — low order
  word 1 — high order

- ?PMYSONS — many sons privilege, 1 bit.
- ?PCHTYP — change type privilege, 1 bit.
- ?PCHPRI — change priority privilege, 1 bit.
- ?PPDPMGR — define PMGR privilege, 1 bit.
- ?PPRNBLK — PROC no-block privilege, 1 bit.
- ?PCHUSER — change username privilege, 1 bit.
- ?PACDEV — access devices privilege, 1 bit.
- ?PUIPCS — use IPC privilege, 1 bit.
- ?PSUSER — Superuser privilege, 1 bit.
- ?PPSUPP — Superprocess privilege, 1 bit.

?PFFD — (doubleword) field descriptor, continued.

- ?PWSON — wide son privilege, 1 bit.
- ?PMCTS — (read–only field) memory constraints, 10 words; the internal format is

words 0–1	?PBWSS
words 2–3	?PBLMEM
words 4–5	?PNBWSS
words 6–7	?PNBLMEM
word 8	?PSOPIO
word 9	?PBATCHP

- ?PHRDPRV — (read–only field) hard privileges, 1 word; the internal format is

bits 0–2	0
bit 3	?PCHWSSL
bit 4	?PWSON
bit 5	?PPSUPP
bit 6	?PMGSYS
bit 7	?PUIPCS
bit 8	?PACDEV
bit 9	?PCHUSER
bit 10	?PPRNBLK
bit 11	?PPDPMGR
bit 12	?PCHPRI
bit 13	?PSUSER
bit 14	?PCHTYP
bit 15	?PMYSONS

- ?PSFTPRV — (read–only field) soft privileges, 1 word; the internal format is

bits 0–9	0
bit 10	?PPWDPRV (negative logic)
bit 11	?PRRAPRV (negative logic)
bit 12	?PVCNPRV (negative logic)
bit 13	?PMODPRV
bit 14	?PBCHPRV
bit 15	?PCNSPRV

- ?PPRCINF — (read–only field) PROC information, 13 words; the internal format is

word 0	?PMXSONS
word 1	?PSFTPRV
word 2	?PHRDPRV
words 3–12	?PMCTS

- ?PINTDIR — initial directory, less than or equal to 64 bytes including a null byte.
- ?PCHWSSL — change WSS limit privilege, 1 bit.

Index

Within the index, a bold page number indicates a primary reference. A range of page numbers indicates the reference spans those pages. A reference such as ?RTODC_PKT... indicates that all references beginning with those characters are found on the referenced pages.

Symbols

Numbers

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

086–000195 updates
093–000542

T

086–000195 updates
093–000542

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