P/OS System Reference Manual

Order No. AA-N620B-TK

November 1985

This manual describes the Professional Operating System, and provides detailed reference information.

REQUIRED SOFTWARE: P/OS V3.0

DIGITAL EQUIPMENT CORPORATION
Maynard, Massachusetts 01754-2571

First Printing, December 1982
Updated, September 1983
Updated, December 1983
Revised, November 1985

CONTENTS

PART I -- SYSTEM OVERVIEW

PART II -- THE SYSTEM SERVICES

PART III -- THE I/O DRIVERS

PART II

THE SYSTEM SERVICES

APPENDIXES

FIGURES

TABLES

PREFACE

Manual Objectives

The *P/OS System Reference Manual* describes the base system software supporting the Professional 300 Series computer.

Intended Audience

You should be a programmer who is creating or modifying an application to run on P/OS (the Professional Operating System). Experience with RSX-11M-PLUS systems is especially helpful.

Structure of This Document

This manual contains three parts:

- **PART I** is a broad system overview that describes how to use the various components of the system.
- **PART II** provides details on the two forms of system services, callable routines and directives.
- **PART III** describes the system I/O capabilities and the bundled I/O drivers.

A chapter summary follows.

- **Chapter 1** introduces the P/OS system. It describes the hardware environment, the operating system components, and basic concepts. Also, it contrasts P/OS features with RSX-11M-PLUS features (on which P/OS is based) and provides application design suggestions.
- **Chapter 2** describes how the system handles logical names, which aid program development by providing device independence.
- **Chapter 3** presents general information on the system's trapping and synchronization mechanisms.

- **Chapter 4** details the parent/offspring task support available under P/OS.

- **Chapter 5** describes the memory management services that P/OS provides.

- **Chapter 6** provides details on the POSSUM library that allows programmers to easily perform often-used functions.

- **Chapter 7** shows you how to use the system directives.

- **Chapter 8** describes each P/OS system directive.

- **Chapter 9** details the system I/O conventions.

- **Chapter 10** describes the P/OS disk drivers (device handlers).

- **Chapter 11** describes the P/OS terminal driver.

- **Chapter 12** describes the virtual terminal driver.

- **Chapter 13** describes the P/OS communication driver.

The appendixes cover system error messages, I/O function codes and status codes, and provide a complete description of hardware-related values stored in the system configuration table.

Associated Documents

- **PDP-11 Architecture Handbook**

  This handbook describes the two processors used in the Professional computers, the F-11 (Professional 325, 350) and the J-11 (Professional 380). Topics covered are data representation, addressing modes, processor instruction set, floating point features, trap and interrupt handling, memory mapping, and bus structures. The handbook also contains a useful summary of differences among the PDP-11 family processors.

- **Professional 300 Series Technical Manual**

  This manual details the hardware components of the Professional computer, including system boards, controllers, drives, monitors, bit map modules, keyboard, and controls and indicators.

- **Professional 325/350 Pocket Service Guide**

  This guide contains detailed troubleshooting methods for software and hardware problems. It explains many of the software errors that the system returns.

- **Other Tool Kit manuals**

  If you are unfamiliar with P/OS and the Tool Kit, please read the *Tool Kit User's Guide*. For descriptions of advanced programming features not covered in this manual, see the *Guide to Writing a P/OS I/O Driver and Advanced Programmer's Notes*.

Conventions Used in This Document

Convention/Term	Meaning
[optional]	In a command line, square brackets indicate that the enclosed item is optional. In a file specification, square brackets are part of the required syntax.
UPPERCASE	Uppercase words and letters indicate that you should type the word or letter exactly as shown.
lowercase	Lowercase words and letters indicate that you should substitute a word or value of your own. Usually the lowercase word identifies the type of substitution required.
...	A horizontal ellipsis indicates that you can repeat the preceding item one or more times. For example: parameter [,parameter...]
. . .	A vertical ellipsis means that not all of the statements are shown.
red	Interactive input appears in red.
Tool Kit	This general term refers to the software you use to develop applications to run on a Professional computer.

Convention/Term	Meaning
Host Tool Kit	The Host Tool Kit is Tool Kit software that runs on a host computer, rather than on the Professional itself.
PRO/Tool Kit	The PRO/Tool Kit is the Tool Kit software that runs on the Professional computer.
10.	All numbers are decimal unless indicated otherwise. A decimal point emphasizes that a number is decimal.

PART I

SYSTEM OVERVIEW

CHAPTER 1

INTRODUCTION TO P/OS

P/OS is the *Professional Operating System*. Based on DIGITAL's RSX-11M-PLUS ("RSX") operating system for PDP-11, P/OS has many features found on operating systems designed for larger minicomputers. Like RSX, P/OS provides a resource-sharing environment that is ideal for multiple real-time activities. It supports multitasking and dynamic memory management, and has extensive I/O and file management services.

This chapter describes the hardware environment for P/OS, the structure of the operating system, and important system concepts. Also, it contrasts P/OS with RSX-11M-PLUS, and presents several application design suggestions.

1.1 P/OS HARDWARE ENVIRONMENT

P/OS runs on either of the two central processing units provided with the Professional 300 Series: the F-11 in the 325 and 350 series, and the J-11 in the 380 series. These processors are full-fledged members of the PDP-11 family. The J-11 is a more recent, higher-performance processor. The two processors share the same instruction set, which is documented in the *PDP-11 Architecture Handbook*.

The Professional includes diagnostic and bootstrap read-only memory in a component called the Base System ROM (BSR). Besides containing bootstrap and self-test instructions, the BSR initializes an area of main memory called the *configuration table*. This table contains information about other system hardware. Programs can access this information via an operating system directive (WIMP$).

Other components of the hardware environment are:

- **Memory Management Unit (MMU)** -- An integral part of both of the Professional's CPUs, the MMU translates virtual addresses into actual physical addresses.

- **Serial Number ROM** -- This read-only memory contains a 6-byte identification number that is unique for each Professional. The serial number is available to programs via the WIMP$ system directive.

- **Floating Point Processor (FPP)** -- The FPP includes microcoded instructions that provide high-speed arithmetic operations for floating-point data.

- **CTI Bus** -- An interconnect path for system option cards, the Computing Terminal Interconnect (CTI) Bus is a six-slot backplane mounted on the system module.

- **I/O Ports** -- The Professional has ports for the video/keyboard device, a serial printer, a communication line, and an Ethernet line. (To use an Ethernet line, you must install the Ethernet controller option.)

- **Peripheral Mass Storage Devices** -- Both the RX50 Diskette Subsystem and the RD series Hard Disk Subsystem are available. There are several Winchester hard disks from which to choose.

- **Time of Day Clock** -- Backed up by a built-in battery, the time of day clock maintains the system time and date.

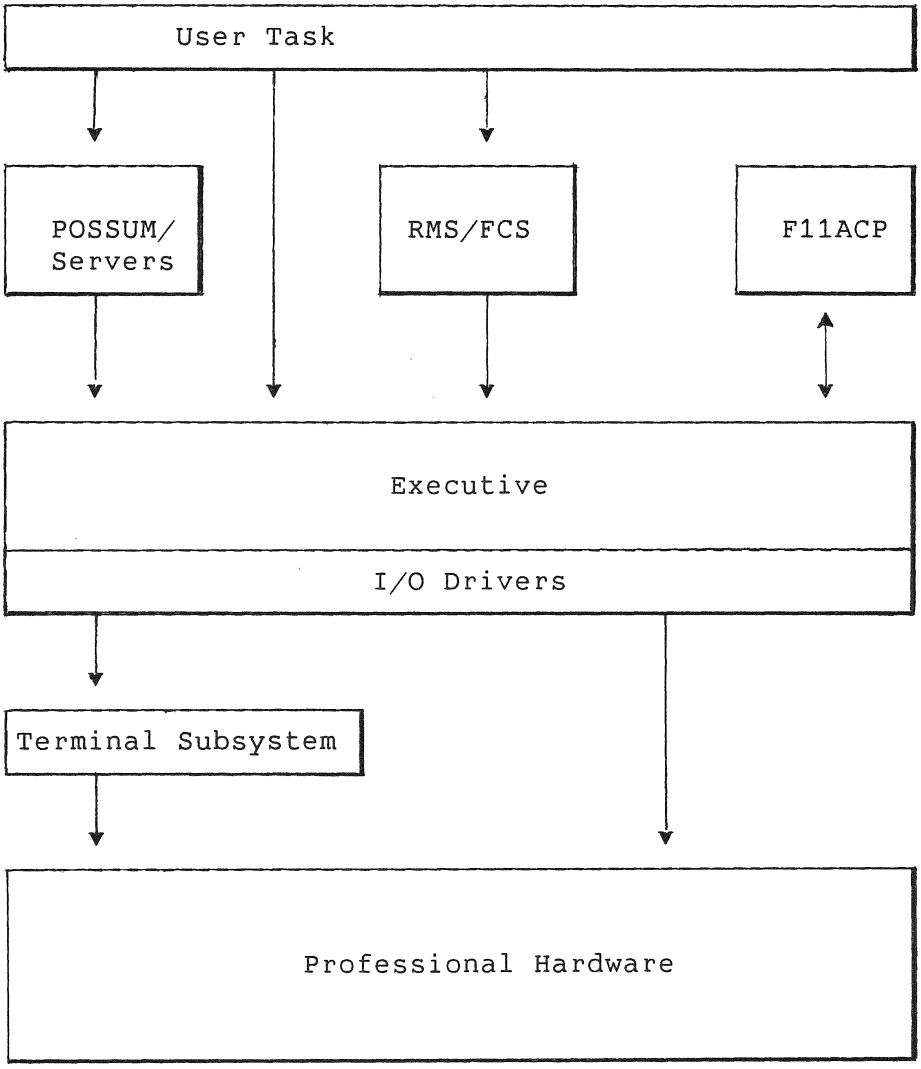
- **Controllers** -- Interrupt controllers handle interrupt arbitration for peripheral devices. The bit map video controllers provide an interface between the CPU and the video display. The RX50 and RD controllers provide an interface between the CPU and their respective devices.

1.2 P/OS SYSTEM COMPONENTS

Figure 1-1 shows the main components of P/OS. The figure illustrates the paths of communication between the components. User tasks, the top layer in the figure, are normally not part of the operating system but are managed by it. Professional hardware, the bottom layer, is also not a part of the operating system, but instead constitutes the hardware environment.

Sections following the figure describe each component.

P/OS SYSTEM COMPONENTS



KEY

F11ACP	FILES-11 Ancillary Control Processor
RMS	Record Management System
FCS	File Control Services
POSSUM	P/OS System Utility Modules

Figure 1-1: Main Components of P/OS

1.2.1 The Executive

The Executive is the foundation of P/OS. It coordinates and controls all activities and resources of the system by performing the following functions:

- **Task scheduling and processing control** -- Tasks are system or user entities that perform functions needed to achieve a desired result.
- **Main memory resource management and control** -- Main memory is the processor storage medium.
- **Interrupt processing** -- The Executive handles synchronous and asynchronous events that occur as a result of task execution.
- **Coordination of I/O and File Management facilities** -- These facilities perform data transfer and data processing operations requested by executing tasks.

1.2.2 I/O Drivers

I/O drivers are system components that interface hardware I/O controllers and their attached devices with the Executive. A device driver provides basic services for a particular type of device, thus removing device-dependent responsibiltity from the Executive. As shown in Figure 1-1, drivers are actually an integral part of the Executive.

The I/O drivers perform the following functions:

- Receive and service interrupts from I/O devices
- Initiate I/O operations as requested by the Executive
- Cancel in-progress I/O operations
- Perform other device-specific functions during system boot

Chapters 10, 11, and 13 describe the system's disk drivers, terminal driver, and communication driver in detail. Chapter 12 describes a special kind of driver that handles a *virtual* device.

The *Guide to Writing a P/OS I/O Driver and Advanced Programmer's Notes*, provided with the Tool Kit, describes driver concepts.

1.2.3 Terminal Subsystem

The Terminal Subsystem is software that provides an interface between the video/keyboard hardware and the terminal driver. It performs such video functions as character generation, blinking, scrolling, polygon fill, and vector generation. The graphics capability of the Terminal Subsystem is provided by GIDIS, the *General Image Display Instruction Set*.

For information on the functions performed by the Terminal Subsystem, see the *Terminal Subsystem Manual* and the *PRO/GIDIS Manual*, both supplied with the Tool Kit.

1.2.4 FILES-11 Ancillary Control Processor

The FILES-11 Ancillary Control Processor (F11ACP) is the P/OS file control processor. It catalogues and maintains files on disks and issues I/O requests to the disk drivers. Also, it controls the virtual and logical structures applied to data and performs translation of one to the other.

FILES-11 is the name of a DIGITAL-standard volume structure that the F11ACP imposes upon disks and diskettes.

1.2.5 Record Management and File Control Services

Record Management Services (RMS) and File Control Services (FCS) serve as translators between user tasks and other I/O facilities of the operating system, such as the F11ACP and device drivers. A user task can incorporate either RMS or FCS routines to enable it to perform record I/O and file I/O functions.

NOTE

> Although P/OS provides a full implementation of FCS, you are urged to always use RMS in new applications. Use FCS to port applications designed to run on RSX systems when such applications already use FCS.

Whereas the F11ACP handles stored data in units of files, RMS and FCS handle stored data in units of records, or file-relative blocks. RMS and FCS allow user tasks to define the internal structure of files--the size and arrangement of records within files--and provide operations that allow user tasks to read and write records in files.

The document *PRO/RMS-11: An Introduction*, provided with the Tool Kit, contains a complete overview of RMS. For further information on FCS, see your RSX-11M/M-PLUS documentation.

1.2.6 P/OS System Utility Modules and Executive Servers

P/OS System Utility Modules (POSSUM) are a set of callable routines that P/OS provides in a resident library called POSSUM. These routines allow user tasks to conveniently perform such functions as mounting volumes, translating logical names, and formatting hard disks. Most of the routines invoke Executive servers to perform their operations, rather than performing the operations themselves.

Chapter 6 describes the POSSUM routines and their servers.

1.3 P/OS BASIC CONCEPTS

The following sections describe important features of the operating system.

1.3.1 Tasks

A *task* is the basic unit of executable code on a P/OS system. An application usually consists of several tasks that work together. Tasks, which reside in files that have the .TSK extension, are sometimes referred to as *executable images*.

Tasks that are part of P/OS are called *system tasks*. Examples are system servers such as CREDEL and INSREM, and the F11ACP. Tasks that you create are called *user tasks*. An application program consists of one or more user tasks.

Whether on disk or in memory, a task is always contiguous.

Before a task can run, it must be installed into the system. *Installing* a task makes it known to P/OS. Several DCL commands, as well as the PROTSK system service (in the POSSUM library), allow you to install a task.

The Executive uses the following system data structures to store information about a task:

- **Task Control Block (TCB)**

  The TCB contains information that the system derives from a task's header, as well as from the directive used to activate the task. The TCB contains information that the Executive needs in order to run the task, such as the address of the task on disk, the priority of the task, and the memory partition in which the task will run.

- **System Task Directory (STD)**

  The STD is simply a linked list of TCBs, organized by priority, that the Executive holds in its working storage area called the *Dynamic Storage Region* (DSR), or *primary pool*. A task whose TCB is in the System Task Directory is known to the system.

- **Active Task List (ATL)**

  When a task becomes active, the Executive inserts the TCB in another linked list, the ATL, which contains the TCBs of all active tasks. A task whose TCB is in the Active Task List is eligible to be loaded into memory and executed.

A task can exist in one of several possible states. Figure 1-2 illustrates and describes the task states.

1.3.2 Memory

The primary units of memory used in P/OS are determined by the 16-bit data path of the Professional's hardware design. The units are:

- **Bit** -- Binary digit, either 1 or 0.
- **Byte** -- Eight bits, the smallest addressable unit of memory.
- **Word** -- Two bytes (16 bits); always begins on an even address

Since the Professional's primary addressing mechanism is the 16-bit word, the maximum physical memory that a task can access at a single moment is 32K words. However, the presence of hardware memory management enables a task to access more than 32K words by using the P/OS memory management directives.

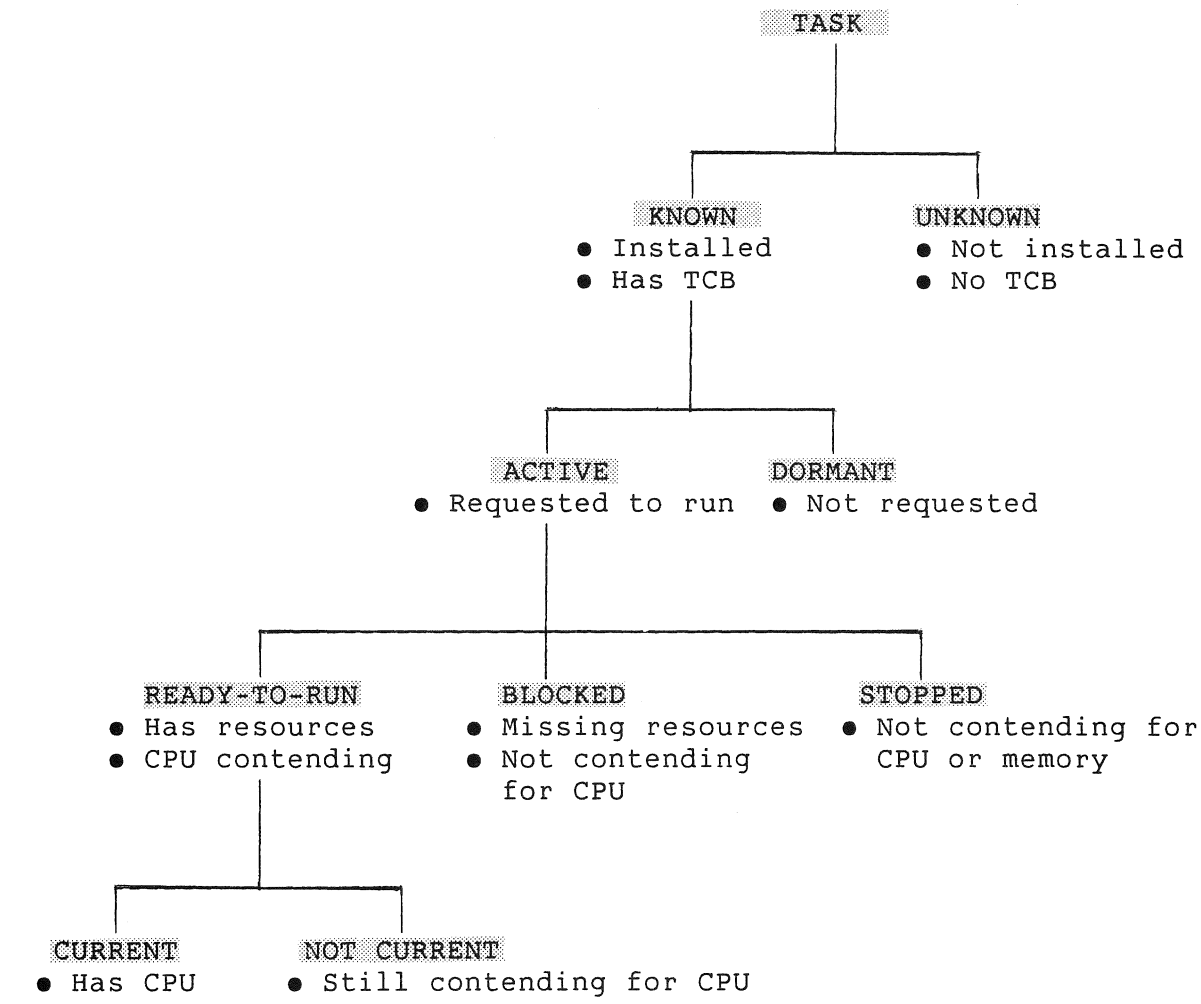


Figure 1-2: Task States

Physical addresses are locations in memory. Virtual addresses are the addresses within a task. Logical addresses are the actual physical memory addresses that the task can access. Virtual to physical address space mapping need not be contiguous.

Using P/OS system features to manipulate logical address space allows you to make use of more than 32K words of physical address space. Furthermore, the multitasking capabilities of P/OS allow you to design applications that can consist of multiple, cooperating, concurrent tasks.

Both Chapter 5 and the *RSX-11M/M-PLUS and Micro/RSX Task Builder Manual* contain greater detail on addressing concepts.

1.3.3 Checkpointing

Checkpointing is the process of writing a task or shared common to a file on a disk to make room for a higher priority task or common competing for memory. Given that a task or common is capable of being checkpointed, tasks and commons compete for memory based on their respective priorities. (The priority value of a common region is equal to one greater than the highest priority task mapped to that common region.)

Section 1.4.2 describes shared commons.

The following task states prohibit a checkpoint from occuring:

- A task region is specified at task-build time to be noncheckpointable.
- A task region has checkpointing disabled (DSCP$).
- A task is exiting.
- A region has resident, mapped tasks--that is, all currently mapped tasks must be checkpointed before the region itself is eligible for checkpointing.
- A region has outstanding I/O.

The following task states promote checkpointing:

- A stopped task has an effective memory priority of zero.
- A checkpointable task doing synchronous terminal I/O (since the task's terminal I/O is buffered and the task is stopped until the I/O completes).
- A task which previously had checkpointing disabled can issue the Enable Checkpointing directive (ENCP$).

1.3.4 System Pool

Throughout this manual we discuss the system's use of pool. *System pool* is a portion of memory that the Executive uses for working storage. For example, as mentioned in a previous section, the Executive uses pool to store task control blocks.

There are two types of pool:

- **Primary Pool** or **DSR**

  Primary pool is also known as the system Dynamic Storage Region. It contains such data structures as Task Control Blocks (TCBs), Offspring Control Blocks (OCBs), I/O packets, and File Control Blocks. The size of primary pool is limited by the size of the Executive's virtual address space.

- **Secondary Pool**

  Secondary pool contains large data structures such as command lines, Send Data packets, file window blocks, and logical name tables. The size of secondary pool is limited only by the physical memory present on a system.

1.4 APPLICATION DESIGN SUGGESTIONS

The following sections list suggestions for designing applications that make the most efficient use of the P/OS multitasking, resource-sharing capabilities. In particular, these suggestions can help you to design programs that might otherwise exceed the 32K word virtual address space limitation of a task.

The *Tool Kit User's Guide* contains additional suggestions for fine-tuning your application.

1.4.1 Use Cooperating Tasks

An application is a task or set of tasks that perform a needed function or set of functions. The application can consist of multiple, cooperating tasks that pass context (variables) between tasks by using data packets, command lines, and shared memory. A task can be requested using the following system directives:

- **SPWN$** -- Useful when passing a command line and there is a need to receive status from or synchronize with the cooperating task.

- **RPOI$** -- Useful when passing a command line and there is no need to receive status from or synchronize with the cooperating task.

- **SDRC$ and VSRC$** -- Useful when passing data packets and there is a need to receive status from or synchronize with the cooperating task.

- **RQST$** -- Useful when simply requesting a task and there is no need to receive status from or synchronize with the cooperating task.

You can pass additional context by using the SDAT$, VSDA$, and SREF$ directives. See Chapters 4 and 8 for details on using these directives.

1.4.2 Use Shared Regions

A shared region is a block of data or code that any number of tasks can use. Shared regions are useful because they make efficient use of physical memory. There are two kinds of shared regions:

- **Shared Common**

  A shared common contains only data. It is a read-write area that provides a way for two or more tasks to share data. When a shared common is not being accessed, the Executive can checkpoint the common by removing it from memory and writing it to the system checkpoint file.* Note that the Executive does that for any read-write area.

- **Shared Library**

  A shared library contains only code. It is a read-only area that provides a way for two or more tasks to share a single copy of commonly used subroutines. When a shared library is not being accessed, the Executive can checkpoint the library by removing it from memory (but *not* by writing it to a checkpoint file). Note that the Executive does that for any read-only area.

---

* This is a change from previous versions of the operating system. When checkpointing or removing a shared common on P/OS V2.0A systems and earlier, the Executive wrote the common to the file containing the initial copy of the common, rather than to the system checkpoint file. See Section 6.8.1 for information on how to change this behavior.

You can create a shared region by building it with the Task Builder and installing it into the system separately from the task that links to it. This type of region is called a *static shared region*. The *RSX-11M/M-PLUS and Micro/RSX Task Builder Manual* describes static shared regions in detail, and shows how to build them.

Alternatively, your task can create a shared region during execution, using the Executive's memory management directives. (Memory management is sometimes referred to as PLAS--Programmable Logical Address Space). This type of region is called a *dynamic shared region*, and is described in Section 5.3 later in this manual.

1.4.3 Use Disk-Resident Overlays

You can divide an application task into pieces called *segments*. Several segments of a task share a given section of the task's virtual address space, but only one segment can be in memory at one time. Segments are individually read from the disk into a section of the task's address space as needed, overwriting a previously read segment.

Disk-resident overlays reduce the memory and virtual address space needed by a task. The *RSX-11M/M-PLUS and Micro/RSX Task Builder Manual* describes segments and disk-resident overlays in detail.

1.4.4 Use Memory-Resident Overlays

Memory-resident overlays are different from disk-resident overlays, in that all of the task's segments are present in physical memory at the same time. By using the memory management directives, the overlay runtime system maps segments into a section of the task's virtual address space as needed. Virtual to physical address mapping changes as the new segments are mapped.

Memory-resident overlays reduce the virtual address space needed by a task, but do not reduce the physical memory requirements. However, tasks constructed of memory-resident overlays are faster since they do not involve disk I/O. The *RSX-11M/M-PLUS and Micro/RSX Task Builder Manual* describes memory-resident overlays in detail.

1.4.5 Use Clustered Resident Libraries

Clustered resident libraries (sometimes called *cluster libraries*) allow tasks to dynamically map memory-resident, shared libraries at run time. The advantage of using clustered resident libraries is that they save task virtual address space by using the same section of task virtual address space to map independent memory-resident, shared libraries. The *RSX-11M/M-PLUS and Micro/RSX Task Builder Manual* describes clustered resident libraries at length.

1.4.6 Use Fast Remapping Feature

Fast remapping is a high-performance method for a task to change the offset and length mapping of a currently mapped region. If your task is normally mapped to a given region and frequently remaps a window to that region, this feature can significantly increase the performance of your task.

Generally speaking, the system performs fast remap operations in one tenth the execution time of a remap using the MAP$ directive, while still providing the same level of region access control and protection present in the operating system.

See Section 5.7 later in this manual for details on fast remapping.

1.5 COMPARING RSX-11M-PLUS AND P/OS

The principal difference between P/OS and RSX is that the default user interface on P/OS is the Menu System, rather than the RSX Monitor Console Routine (MCR). Some RSX software features remain the same on P/OS, some have been removed, some have changed, and some new software features have been added. Some RSX utilities carried over to P/OS are now program-callable routines.

Table 1-1: Summary of Differences Between RSX and P/OS

RSX Features Not on P/OS	RSX Features Modified for P/OS	P/OS Features Not on RSX
Group global event flags	Terminal driver	Automatic volume mounting and dismounting
Batch processing	System utilities (FMT, BAD, INI, INSTALL, FIX, REMOVE, UFD)	Enhanced high-level language interface to the system and the utilities (POSSUM library)
Alternative CLI support	GET TIME	
Virtual Monitor Routines (VMR)	Print queue management	
Console Logging	Account server	
Error logging	LOAD,UNLOAD	
System accounting		
Shadow recording		
System generation		
Checkpoint Common Region directive (CPCR$)		

CHAPTER 2

LOGICAL NAMES

A logical name is a combination of a name (defined by you or by P/OS) and an equivalence value (any part of a file specification).* You can use a logical name to refer to all or part of a file specification.

Logical names provide programs with *device* and *file independence*. For example, from within a program you can refer to an input or output file using logical names rather than physical filenames. Then, between invocations of the program, you can change the input and output files simply by associating the logical names with new physical filenames.

This chapter describes how the system stores and translates logical names, and how you can perform operations on logical names from within your program by using several Executive directives and a callable system service.

2.1 LOGICAL NAME STORAGE

The system stores logical names as name-equivalence pairs. For every logical name stored by the system, an equivalence value must exist. The name and equivalence each consist of a string whose maximum length is 255 bytes.

Three tables, located in secondary pool, contain all logical name definitions. The purpose of the different tables is to enable programmers to define the *scope* of logical names. That is, you define a logical in a particular table in order to provide the desired level of access to that logical by other users or application tasks. Table 2-1 describes the tables.

---

* See the *Tool Kit User's Guide* for a list of system-defined logical names.

Table 2-1: Logical Name Tables

Name	Number	Description
LT.SYS	0	**System Table.** Any logical name that must be accessible to all users and applications on a system belongs in the system table. The scope of a logical defined in LT.SYS is any logged in user and any currently executing application. There can be only one system table on any system.
LT.SES	4	**Session Table.** A session table contains logical names that are part of the context associated with a user's login-logout period. The scope of logical names defined in LT.SES consists of the user who owns that table plus all applications currently executing for that user. The number of session tables on a system is equal to the number of users logged in to the system.
LT.USR	2	**User Table.** For each application running on the system, one user table is allocated. The scope of logical names defined in an LT.USR table consists only of the application associated with that particular table. Tasks that are not part of an executing application cannot access that application's LT.USR logical names. The number of user tables on a system is equal to the number of currently executing applications.

2.2 LOGICAL NAME MODIFIERS

Every equivalence value contains a modifier. A *modifier* is a means of distinguishing among equivalence values that are associated with the same logical name and that reside in the same table.

P/OS currently defines three values for a modifier:

- **Mod 2**

  Equivalence values that have a modifier value of 2 are said to be permanent. A *permanent* equivalence value is one that the system login process creates when a user logs into the system.

- **Mod 1**

  Equivalence values that have a modifier value of 1 are said to be temporary. A *temporary* equivalence is one that a user task creates to supersede a particular permanent equivalence. It differs from the permanent equivalence only in the mod value, which is 1 instead of 2.

- **Mod 0**

  The system does not store the modifier value 0 in an equivalence value. Instead, it maps mod 0 to the other modifier values during operations on logical names. Table 2-2 shows the mapping. In general, you should specify mod 0 when performing operations on logical names.

Table 2-2 describes the system's actions for all operations when you specify different modifier values.

Table 2-2: Operations on Logicals With Different Modifiers

Mod	Operation		
	Create	**Delete**	**Translate**
Mod 0	Create mod 1 logical.	Delete mod 1 logical.	Return mod 1 equivalence if found; if not found, return mod 2 equivalence (if present).
Mod 1	Create mod 1 logical.	Delete mod 1 logical.	Return mod 1 equivalence.
Mod 2	Create mod 2 logical.	Delete mod 2 logical.	Return mod 2 equivalence.

2.2.1 Modifiers in Duplicate Logical Names

A *duplicate logical name* is a logical name that is associated with more than one equivalence value. If duplicate logical names are in different tables, you might be able to distinguish them simply by indicating which table they are in. In this case, the mod values for the duplicates can be the same.

However, if the duplicates are in the same table, then each equivalence value must have a unique modifier to distinguish it from other duplicates. You can specify modifier values 128 through 255 (decimal) to create duplicates. Note that modifier values 0 through 127 are used by the system.

NOTE

Duplicate logical names are possible only when the user tasks handle the logical name translations. Within the context of the system software (such as RMS and volume mounting procedures), the only recognized value for the mod argument is 0.

If you create a logical name that duplicates an existing logical name with the same modifier value, the system supersedes the old equivalence value with the new one. This is true for any mod value.

The maximum number of equivalence values that can be associated with a single logical name in any table is 255 (decimal). However, the current size of available secondary pool, where the tables reside, sets a practical upper limit.

2.3 LOGICAL NAME TRANSLATION

As part of I/O processing in programs that use RMS, RMS translates logical names and returns their equivalence values. The following conventions govern RMS translation of logical names:

- RMS translates all logical names that occur within the context of a valid file specification.
- RMS continues to do translations of logical name strings until it encounters an equivalence name string beginning with an underscore (_), until it fails to translate a string, or until it reaches the maximum number of translations allowed.
- RMS performs a maximum of eight translations for a given logical name. If the number of logical name translations exceeds the maximum, RMS issues an error.

2.4 LOGICAL NAMES FOR FILES-11 VOLUMES

The FILES-11 ACP creates two logical names when it mounts a file-structured volume, such as a disk or diskette:

- F11ACP creates a logical whose name is the volume label that was previously given the volume when it was initialized. Its equivalence value is the physical device name of the device on which the volume is mounted.
- F11ACP creates a logical whose name is the physical device name and whose equivalence value is the volume label. This is the reverse of the first logical.

For example, suppose you mount in drive 1 a diskette having the volume name FINANCE. The ACP creates the two logicals shown in Table 2-3.

Table 2-3: Sample F11ACP-Created Logicals for Diskette

Logical Name	Equivalence Value
FINANCE:	_DZ001:
DZ001:	FINANCE:

An application program can refer to the diskette with the volume label FINANCE by using the logical name FINANCE:. RMS translates the logical name to determine the actual physical device. Similarly, an application programmer can use the logical name DZ001: to determine the volume label of the volume that is currently mounted.

2.4.1 Removable Versus Nonremovable Volumes

There are two classes of volumes that you can mount on a Professional computer: removable and nonremovable.

Removable volumes are those that are easily taken out of the system unit and transported to another system. Floppy disks are an example. *Nonremovable* volumes are those that are not easily transported. Hard disks are an example. (Hard disks can be transported, but they are not designed to be transportable.)

F11ACP creates logicals for removable media exactly as shown in Table 2-3. The general format for a logical representing a removable media volume label is:

```
vlabel:
```

where

vlabel Is the volume label of the removable media.

The format of the volume label logical for nonremovable media is different:

```
node$$vlabel[_n]:
```

where

node	Is the name of the node or system on which the hard disk resides.
vlabel	Is the volume label of the nonremovable media.
n	Is a number F11ACP assigns to the volume when there is more than 1 volume with the same volume label on a system. The duplicate volumes are numbered starting from 1.

Table 2-4 illustrates the two logical names that F11ACP creates when you mount a second hard disk whose volume label is DATAVOL on the node NODNAM.

Table 2-4: Sample F11ACP-Created Logicals for Hard Disk

Logical Name	Equivalence Value
NODNAM$$DATAVOL_1:	_DW003:
DW003:	NODNAM$$DATAVOL_1:

2.5 LOGICAL NAME DEFAULT DIRECTORY STRING

The system provides a special set of logical names known as the *default directory*. The default directory is a character string stored in secondary pool. You can get and set its equivalence value by using the GDIR$ and SDIR$ Executive directives.

If RMS encounters an input string with no specified directory, or if the input string contains a pair of closed empty brackets--an explicit request for the default directory--RMS returns the default directory string.

2.6 LOGICAL NAME OPERATIONS

The system provides several Executive directives, as well as a callable routine called PROLOG, to perform create, delete, and translate operations on logical names. General descriptions of the different operations follow.

For detailed information on any directives, refer to Chapter 8. For details on PROLOG, see Section 6.7.

2.6.1 Creating a Logical Name

Use the CLOG$ directive to create a logical name string and the associated equivalence name string. The length of each logical name string can be a maximum of 255(10) characters (bytes). Creation of the logical name string requires the use of the secondary pool, which is of limited size.

The following example shows how to create a logical name with the CLOG$ directive.

```
          .MCALL     CLOG$,DIR$
LNAME:    .ASCII     /EXPENSES:/    ;LOGICAL NAME STRING
LNAMSZ=   .-LNAME                   ;SIZE OF LOGICAL NAME STRING
ENAME:    .ASCII     /FINANCE:/     ;EQUIVALENCE NAME STRING
ENAMSZ=   .-ENAME                   ;DEFINE SIZE OF EQUIVALENCE
                                    ;NAME STRING
          .EVEN
NAMVOL:   CLOG$      ,LT.USR,LNAME,LNAMSZ,ENAME,ENAMSZ
START:    DIR$       #NAMVOL        ;CREATE LOGICAL NAME
```

2.6.2 Deleting a Logical Name

Use the DLOG$ directive to delete entries from a logical name table. When you code a call to the DLOG$ directive, you can delete a single logical name from the table, or you can delete all the logical names in the table.

The example below deletes all the mod 1 logical name entries from the user logical name table:

```
          .MCALL     DLOG$,DIR$
DELALL:   DLOG$      ,LT.USR
START:    DIR$       #DELALL        ;DELETE LOGICAL NAME
                  .
                  .
                  .
```

The next example deletes a single logical name entry from the user logical name table:

```
          .MCALL     DLOG$,DIR$
NAME:     .ASCII     /TMONK/
NAMESZ=   .-NAME
          .EVEN
NAMDEL:   DLOG$      ,LT.USR,NAME,NAMESZ
START:    DIR$       #NAMDEL       ;DELETE LOGICAL NAME
          .
          .
          .
```

2.6.3 Translating a Logical Name

Use the TLOG$ directive to translate a logical name string into its equivalence string. RMS issues the TLOG$ directive for each logical name translation necessary in a program.

The following example shows a call from a user program to the TLOG$ directive to translate the logical name EXPENSES:

```
          .MCALL     TLOG$,DIR$
SIZE:     .WORD 0                  ;SIZE OF EQUIVALENCE NAME
                                   ;IN BYTES
ENAME:    .BLKB 20.                ;BUFFER TO CONTAIN
                                   ;EQUIVALENCE NAME
ENAMSZ=   .-ENAME
LNAME:    .ASCII     /EXPENSES:/   ;BUFFER CONTAINING
                                   ;LOGICAL NAME
LNAMSZ=   .-LNAME
          .EVEN
GETNAM:   TLOG$      ,LT.USR,LNAM,LNAMSZ,ENAME,ENAMSZ,SIZE
START:    DIR$       #GETNAM       ;TRANSLATE LOGICAL NAME
          .
          .
          .
```

2.6.4 Setting a Default Directory String

Use the SDIR$ macro to establish a default directory. Be aware that the default directory belongs to and should be controlled by the user, not by an application. Thus, we recommend that you prompt the user for the default directory before you set the string.

The following example shows how to use the SDIR$ macro to set up a default directory string:

```
          .MCALL     SDIR$,DIR$
DDSNAM:   .ASCII     /[SOLOS]/
DDSSZ=    .-DDSNAM
          .EVEN
SETNAM:   SDIR$      ,DDSNAM,DDSSZ
START:    DIR$       #SETNAM         ;SET DEFAULT DIRECTORY
          .
          .
          .
```

NOTE

The PROLOG callable system routine is the preferred method of setting a default directory.

2.6.5 Retrieving a Default Directory String

Use the GDIR$ directive to retrieve a default directory string. The system returns the default directory string to the specified user buffer, along with the length of the string.

The following example shows how to use the GDIR$ macro to retrieve the default directory string:

```
          .MCALL     GDIR$,DIR$
DDSNAM:   .BLKB      100.            ;DEFINE BUFFER FOR DEFAULT
                                     ;DIRECTORY STRING
DDSSZ=    .-DDSNAM                   ;CALCULATE BUFFER SIZE
          .EVEN
GETNAM:   GDIR$      ,DDSNAM,DDSSZ
START:    DIR$       #GETNAM         ;GET DEFAULT DIRECTORY
                                     ;STRING
                .
                .
                .
```

CHAPTER 3

USING EVENT, TRAP, AND SYNCHRONIZATION SERVICES

This chapter introduces the concept of significant events and describes the ways in which your code can make use of event flags, synchronous and asynchronous system traps, and stop-bit synchronization.

3.1 SIGNIFICANT EVENTS

A significant event is a change in system status that causes the Executive to reevaluate the eligibility of all active tasks to run. A significant event is usually caused (either directly or indirectly) by a system directive issued from within a task. (All of the system directives named in this chapter are described in detail in Chapter 8.)

Significant events include the following:

- I/O completion
- Task exit
- Execution of a Send Data directive (SDAT$)
- Execution of a Send Data, Request and Pass OCB directive (SDRP$)
- Execution of a Send, Request, and Connect directive (SDRC$)
- Execution of a Send By Reference or a Receive By Reference directive (SREF$ or RREF$)
- Execution of an Alter Priority directive (ALTP$)

- Removal of an entry from the clock queue (for example, resulting from a Mark Time directive previously executed or the issuance of a rescheduling request)

- Execution of a Declare Significant Event directive (DECL$S)

- Execution of the round-robin scheduling algorithm at the end of a round-robin scheduling interval

- Execution of an Exit, an Exit With Status, or an Emit Status directive (EXIT$S, EXST$, or EMST$)

3.2 EVENT FLAGS

Event flags are a means by which tasks recognize specific events. (Tasks also use Asynchronous System Traps, ASTs, to recognize specific events. See Section 3.3.3.)

In requesting a system operation (such as an I/O transfer), a task can associate an event flag with the completion of the operation. When the event occurs, the Executive sets the specified flag. Several examples later in this section describe how tasks can use event flags to coordinate task execution.

To enable tasks to distinguish one event from another, 64 (decimal) event flags are available. Each event flag has a corresponding unique Event Flag Number, or EFN (all numbers are decimal):

- Numbers 1 through 32 form a group of local flags that are unique to each task and are set or cleared as a result of that task's operation.

- Numbers 33 through 64 form a second group of flags that are common to all tasks, hence their name *common flags*. Common flags can be set or cleared as a result of any task's operation.

- The last 8 flags in each group, local flags (25 through 32) and common flags (57 through 64) are reserved for use by DIGITAL software components.

Tasks can use the common flags for intertask communication, or they can use their own local event flags internally. They can set, clear, and test event flags by using Set Event Flag (SETF$), Clear Event Flag (CLEF$), and Read All Event Flags (RDAF$) directives.

CAUTION

Erroneous or multiple setting and clearing of event flags can result in software faults that are difficult to trace. We suggest that you avoid using common event flags.

Examples 1 and 2 illustrate the use of common event flags (33 through 64) to synchronize task execution. Examples 3 and 4 illustrate the use of local flags (1 through 32).

- **Example 1**

  Task B clears common event flag 35 and then blocks itself by issuing a Wait For directive that specifies common event flag 35.

  Subsequently another task, Task A, specifies event flag 35 in a Set Event Flag directive to inform Task B that it can proceed. Task A then issues a Declare Significant Event directive to ensure that the Executive will schedule Task B.

- **Example 2**

  To synchronize the transmission of data between Tasks A and B, Task A specifies Task B and common event flag 42 in a Send Data directive.

  Task B has specified flag 42 in a Wait For directive. When Task A's Send Data directive has caused the Executive to set flag 42 and to cause a significant event, Task B proceeds and issues a Receive Data directive because its Wait For condition has been satisfied.

- **Example 3**

  A task contains a Queue I/O Request directive and an associated Wait For directive; both directives specify the same local event flag. When the task queues its I/O request, the Executive clears the local flag. If the requested I/O is incomplete when the task issues the Wait For directive, the Executive blocks the task.

  When the requested I/O is completed, the Executive sets the local flag and causes a significant event. The task then resumes its execution at the instruction that follows the Wait For directive. Using the local event flag in this manner ensures that the task does not manipulate incoming data until the transfer is complete.

- **Example 4**

  A task specifies the same local event flag in a Mark Time and an associated Wait For directive. When the Mark Time directive is issued, the Executive first clears the local flag and subsequently sets it when the indicated time interval has elapsed.

  If the task issues the Wait For directive before the local flag is set, the Executive blocks the task, which resumes when the flag is set at the end of the proper time interval. If the flag has been set first, the directive is a no-op and the task is not blocked.

Specifying an event flag does not mean that a Wait For directive must be issued. Event flag testing can be performed at any time. The purpose of a Wait For directive is to stop task execution until an indicated event occurs. Hence, it is not necessary to issue a Wait For directive immediately following a Queue I/O Request directive or a Mark Time directive.

If a task issues a Wait For directive that specifies an event flag that is already set, the blocking condition is immediately satisfied and the Executive immediately returns control to the task.

Tasks can issue Stop For directives instead of Wait For directives. When this is done, an event flag condition not satisfied will result in the task's being stopped (instead of being blocked) until the event flag is set. A task that is blocked still competes for memory resources at its running priority. A task that is stopped competes for memory resources at priority 0.

The simplest way to test a single event flag is to issue the directive CLEF$ or SETF$. Both these directives can cause the following return codes:

IS.CLR - Flag was previously clear

IS.SET - Flag was previously set

For example, if a set common event flag indicates the completion of an operation, a task can issue the CLEF$ directive both to read the event flag and simultaneously to reset it for the next operation. If the event flag was previously clear (the current operation was incomplete), the flag remains clear.

3.3 SYSTEM TRAPS

System traps are transfers of control (also called software interrupts) that provide tasks with a means of monitoring and reacting to events. The Executive initiates system traps when certain events occur. The trap transfers control to the task associated with the event and gives the task the opportunity to service the event by entering a user-written routine.

There are two kinds of system traps:

- **Synchronous System Traps (SSTs)**

  SSTs detect events directly associated with execution of program instructions. They are synchronous because they always recur at the same point in the program when trap-causing instructions occur. For example, an illegal instruction causes an SST.

- **Asynchronous System Traps (ASTs)**

  ASTs detect events that occur asynchronously to the task's execution. That is, the task has no direct control over the precise time that the event--and therefore the trap--can occur. For example, the completion of an I/O transfer can cause an AST to occur if you specify the AST argument in the QIO directive.

A task that uses the system trap facility issues system directives to establish entry points for user-written service routines. Entry points for SSTs are specified in a single table. AST entry points are set by individual directives for each kind of AST. When a trap condition occurs, the task automatically enters the appropriate routine if its entry point has been specified.

3.3.1 Synchronous System Traps (SSTs)

SSTs can detect the execution of invalid instructions, instructions with invalid addresses, and trap instructions (TRAP, EMT, IOT, BPT).*

---

* See the *PDP-11 Architecture Handbook* for a description of processor instructions referred to in this chapter.

NOTE

If you use the Fast Remap feature, which operates via IOT instructions, the IOT entry point in your SST vector table is ignored. See Section 5.7 for details on the Fast Remap feature.

The user can set up an SST vector table, containing one entry per SST type. Each entry is the address of an SST routine that services a particular type of SST (a routine that services illegal instructions, for example). When an SST occurs, the Executive transfers control to the routine for that type of SST. If a corresponding routine is not specified in the table, the task is aborted.

The SST routine enables the user to process the failure and then return to the interrupted code. Note that if a debugging aid and the user's task both have an SST vector enabled for a given condition, the debugging aid vector is referenced first to determine the service routine address.

SST routines must always be reentrant if there is a possibility that an SST can occur within the SST routine itself. Although the Executive initiates SSTs, the execution of the related service routines is indistinguishable from the task's normal execution. An AST or another SST can therefore interrupt an SST routine.

3.3.2 SST Service Routines

The Executive initiates SST service routines by pushing the task's Processor Status (PS), Program Counter (PC), and trap-specific parameters onto the task's stack. After removing the trap-specific parameters, the service routine returns control to the task by issuing an RTI or RTT processor instruction. Note that the task's general purpose registers R0 through R5 and SP are not saved. If the SST routine makes use of them, it must save and restore them itself.

To the Executive, SST routine execution is indistinguishable from normal task execution, so that all directive services are available to an SST routine. An SST routine can remove the interrupted PS and PC from the stack and transfer control anywhere in the task; the routine does not have to return control to the point of interruption. Note that any operations performed by the routine (such as the modification of registers or the setting or clearing of event flags) remain in effect when the routine eventually returns control to the task.

A trap vector table within the task contains all the service routine entry points. You can specify the SST vector table by means of the Specify SST Vector Table For Task directive or the Specify SST Vector For Debugging Aid directive. The trap vector table has the format shown in Table 3-1.

Table 3-1: Trap Vector Table

Word	Offset	Vector	Trap
0	S.COAD	4	Odd address trap (PC380 only) or nonexistent memory error
1	S.CSGF	250	Memory protect violation
2	S.CBPT	14	T-bit trap or execution of a BPT instruction
3	S.CIOT	20	Execution of an IOT instruction (except when using Fast Remap feature)
4	S.CILI	10	Execution of a reserved instruction
5	S.CEMT	30	Execution of a non-RSX EMT instruction
6	S.CTRP	34	Execution of a TRAP instruction

Depending on the reason for the SST, the task's stack can also contain additional information, as follows:

TRAP instruction or EMT other than 377 (and 376 in the case of unmapped tasks and mapped privileged tasks) (complete stack)

SP+04 - PS
SP+02 - PC
SP+00 - Instruction operand (low-order byte) multiplied by 2, non-sign-extended

Memory protect violation (complete stack)

SP+10 - PS
SP+06 - PC
SP+04 - Memory protect status register (SR0)*
SP+02 - Virtual PC of the faulting instruction (SR2)*
SP+00 - Instruction backup register (SR1)*

All items except the PS and PC must be removed from the stack before the SST service routine exits.

3.3.3 Asynchronous System Traps (ASTs)

The primary purpose of an AST is to inform the task that a certain event has occurred (for example, the completion of an I/O operation). As soon as the task has serviced the event, it can return to the interrupted code.

Some directives can specify both an event flag and an AST; with these directives, ASTs can be used as an alternative to event flags or the two can be used together. Therefore, you can specify the same AST routine for several directives, each with a different event flag. Thus, when the Executive passes control to the AST routine, the event flag can determine the action required.

AST service routines must save and restore all registers used. If the registers are not restored after an AST has occurred, the task's subsequent execution may be unpredictable.

Although it cannot distinguish between execution of an SST routine and task execution, the Executive is aware that a task is executing an AST routine. An AST routine can be interrupted by an SST routine, but not by another AST routine.

The following notes describe general characteristics and uses of ASTs:

- If an AST occurs while the related task is executing, the task is interrupted so that the AST service routine can be executed.

---

* For details on SR0, SR1, and SR2, see the section on memory management in the *PDP-11 Architecture Handbook*.

- If an AST occurs while another AST is being processed, the Executive queues the latest AST (First-In-First-Out, or FIFO). The task then processes the next AST in the queue when the current AST service is complete (unless AST recognition was disabled by the AST service routine).

- If an AST suspends a task, the task remains stopped or suspended after the AST routine is executed, unless the task is explicitly resumed or unstopped either by the AST service routine itself, or by another task.

- If an AST occurs while the related task is waiting (or stopped) for an event flag to be set (a Wait For or Stop For directive), the task continues to wait after execution of the AST service routine unless the event flag is set upon AST exit.

- If an AST occurs for a checkpointed task, the Executive queues the AST (FIFO), brings the task into memory, and then activates the AST.

- The Executive allocates the necessary dynamic memory when an AST is specified. Thus, no AST condition lacks dynamic memory for data storage when it actually occurs. The AST reuses the storage allocated for I/O and Mark Time directives. Therefore, no additional dynamic storage is required.

- Two directives, Disable AST Recognition and Enable AST Recognition, allow a program to queue ASTs for subsequent execution during critical sections of code. (A critical section might be one that accesses data bases also accessed by AST service routines, for example.) If ASTs occur while AST recognition is disabled, they are queued (FIFO) and then processed when AST recognition is enabled.

3.3.4 AST Service Routines

When an AST occurs, the Executive pushes the task's Wait For mask word, the PS, the PC, and the DSW onto the task's stack. This information saves the state of the task so that the AST service routine has access to all the available Executive services.

The preserved Wait For mask word allows the AST routines to establish the conditions necessary to unblock the waiting task. Depending on the reason for the AST, the stack can also contain additional parameters.

Note that the task's general purpose registers R0 through R5 and SP are not saved. If the AST service routine makes use of them, it must save and restore them itself.

The Wait For mask word comes from the offset T.EFLM in the task's Task Control Block (TCB). The value of the Wait For mask word and the event flag range to which it corresponds depend on the last Wait For or Stop For directive issued by the task.

For example, if the last such directive issued was Wait For Single Event Flag 42 (event flags are decimal), the mask word has a value of 1000 (octal) and the event flag range is from 33 through 48. Bit 0 of the mask word represents flag 33, bit 1 represents flag 34, and so on.

The Wait For mask word is meaningless if the task has not issued any type of Wait For or Stop For directive.

Your code should not attempt to modify the Wait For mask while in the AST routine. For example, putting a zero in the Wait For mask results in an unclearable Wait For state.

After processing an AST, the task must remove the trap-dependent parameters from its stack--that is, everything from the top of the stack down to, but not including, the task's Directive Status Word. It must then issue an AST Service Exit directive with the stack set as indicated in the description of that directive. When the AST service routine exits, it returns control to one of two places: another AST or the original task.

There are several variations on the format of the task's stack; these variations occur as follows:

- **FPU Exception Trap Occurs**

  If a task needs to be notified when a Floating Point Processor exception trap occurs, it issues a Specify Floating Point Processor Exception AST directive. When the task specifies this directive, an AST occurs when a Floating Point Processor exception trap occurs. The stack contains the following values:

  SP+12 - Event flag mask word
  SP+10 - PS of task prior to AST
  SP+06 - PC of task prior to AST
  SP+04 - Task's DSW
  SP+02 - Floating exception code*
  SP+00 - Floating exception address

- **Data or Common Reference Received**

  If a task needs to be notified when it receives either a data packet or a reference to a common area, it issues either a Specify Receive Data AST or a Specify Receive By Reference AST directive. An AST occurs when the data packet or common reference is sent to the task. The stack contains the following values:

  SP+06 - Event flag mask word
  SP+04 - PS of task prior to AST
  SP+02 - PC of task prior to AST
  SP+00 - Task's DSW

- **I/O Request Completes**

  When a task queues an I/O request and specifies an appropriate AST service entry point, an AST occurs upon completion of the I/O request. The task's stack contains the following values:

  SP+10 - Event flag mask word
  SP+06 - PS of task prior to AST
  SP+04 - PC of task prior to AST
  SP+02 - Task's DSW
  SP+00 - Address of I/O status block for I/O request (or zero if none was specified)

- **Mark Time Interval Elapsed**

  When a task issues a Mark Time directive and specifies an appropriate AST service entry point, an AST occurs when the indicated time interval has elapsed. The task's stack contains the following values:

  SP+10 - Event flag mask word
  SP+06 - PS of task prior to AST
  SP+04 - PC of task prior to AST
  SP+02 - Task's DSW
  SP+00 - Event flag number (or zero if none was specified)

---

* Refer to the *PDP-11 Architecture Handbook* for a description of the FPU exception code values.

- **Offspring Returns Status with Exit AST**

  An offspring task, connected by a Spawn, Connect, or Send, Request and Connect directive, returns status to the connected (parent) task(s) upon exiting by the Exit AST. The parent task's stack contains the following values:

  SP+10 - Event flag mask word
  SP+06 - PS of task prior to AST
  SP+04 - PC of task prior to AST
  SP+02 - Task's DSW
  SP+00 - Address of exit status block

- **Task Aborted with SREA$-Specified AST Present**

  If a directive aborts a task when the Specify Requested Exit AST (SREA$) is in effect, the abort AST is entered. The task's stack contains the following values:

  SP+06 - Event flag mask word
  SP+04 - PS of task prior to AST
  SP+02 - PC of task prior to AST
  SP+00 - Task's DSW

- **Task Aborted with SREX$-Specified AST Present**

  If a directive aborts a task when the Extended Specify Requested Exit AST (SREX$) is in effect, the abort AST is entered. The task's stack contains the following values:

  SP+12 - Event flag mask word
  SP+10 - PS of task prior to AST
  SP+06 - PC of task prior to AST
  SP+04 - DSW of task prior to AST
  SP+02 - Trap dependent parameter
  SP+00 - Number of bytes to add to SP to clean stack

3.4 STOP-BIT SYNCHRONIZATION

Stop-bit synchronization allows tasks to be checkpointed during terminal (buffered) I/O or while waiting for an event to occur (for example, an event flag to be set or an Unstop directive to be issued). You can control synchronization between tasks by setting the task's Task Control Block (TCB) stop bit.

When the task's stop bit is set, the task is blocked from further execution, its priority for memory allocation effectively drops to zero, and it can be checkpointed by any other task in the system, regardless of priority.

If checkpointed, the task remains out of memory until its stop bit is cleared, at which time the task becomes unstopped, its normal priority for memory allocation becomes restored, and it is considered for memory allocation based on the restored priority.

If the stopped task receives an AST, the task becomes unstopped until it exits the AST routine. Memory allocation for the task during the AST routine is based on the task's priority before the stopped state. Note that a task cannot be stopped when an AST is in progress, but the AST routine can issue either an Unstop or Set Event Flag directive to reference the task. This causes the task to remain unstopped after it issues the AST Service Exit directive.

There are three ways in which a nonprivileged task can be stopped and three corresponding ways it can become unstopped. Only one method for stopping a task can be applied at a time.

- A task is stopped whenever it is in a Wait For state and has outstanding buffered I/O. A task is unstopped when the buffered I/O is completed or when the Wait For condition is satisfied.

- You can stop a task for event flag by issuing the directive Stop For Single Event Flag or Stop For Logical OR Of Event Flags. In this case, the task can only be unstopped by setting the specified event flag.

- You can stop a task by issuing the Stop or the Receive Data Or Stop directive. In this case, the task can only be unstopped by issuing the Unstop directive.

You cannot stop a task when an AST is in progress (AST state). Any directives that can cause a task to become stopped are illegal at the AST state.

When a task is stopped for any reason at the task state, the task can still receive ASTs. If the task is checkpointed, it becomes eligible for entrance back into memory when an AST is queued for it. The task retains its normal priority in memory while it is at the AST state or has ASTs queued. Once the task has exited the AST routine with no other ASTs queued, the task is again stopped and effectively has zero priority for memory allocation.

STOP-BIT SYNCHRONIZATION

You can use the following directives for stop-bit synchronization:

- **Stop**

  This directive stops the issuing task and cannot be issued at the AST state.

- **Receive Data Or Stop and Variable Receive Data Or Stop**

  These directives attempt to dequeue send data packets from the specified task (or any task if none is specified). If there is no such packet to be dequeued, the issuing task is stopped. These directives cannot be issued at the AST state.

- **Stop For Logical OR Of Event Flags**

  This directive stops the issuing task until at least one of the specified flags in the specified group of event flags become set. If any of the specified event flags are already set, the task does not become stopped. This directive cannot be issued at the AST state.

- **Stop For Single Event Flag**

  This directive stops the issuing task until the indicated event flag becomes set. If the specified event flag is already set, the task does not become stopped. This directive cannot be issued at the AST state.

- **Unstop**

  This directive unstops a task that has become stopped by the Stop or Receive Data Or Stop directive.

CHAPTER 4

USING PARENT/OFFSPRING TASKING SERVICES

Parent/offspring tasking allows you to establish and control the relationships between a governing (parent) task and any subordinate (offspring) tasks. A parent task starts or connects to an offspring task.

One application for the parent-offspring task relationship is a multitask application. In such an application, the main task controlling the application requires other tasks to perform subfunctions for the application. With parent/offspring tasking, you can set up the necessary relationships between the parent task and its offspring to control processing.

Starting (or activating) offspring tasks is called *spawning*. Spawning also includes the ability to establish task communications; a parent task can be notified when an offspring task exits and can receive status information from the offspring task.

Status returned from an offspring task to a parent task indicates successful completion of the offspring task or identifies specific error conditions.

This chapter first describes the task states and the directives that affect those states, and then describes how you can perform parent/offspring tasking operations.

4.1 TASK STATES

The Executive recognizes the existence of a task only after it has been successfully installed.

Once a task is known to the system, it exists in one of two states: dormant or active. Some system directives cause a task to change from one state to another.

A task is active from the time it is requested until the time it exits. Requesting a task means issuing the RQST$, RUN$, SPWN$, SDRC$, VSRC$, RPOI$, or SDRP$ directive. An active task is eligible for scheduling, whereas a dormant task is not.

The three substates of an active task are as follows:

- **Ready-to-run** - A ready-to-run task competes with other tasks for CPU time on the basis of priority. The highest priority ready-to-run task obtains CPU time and thus becomes the current task.

- **Blocked** - A blocked task is unable to compete for CPU time because a needed resource is not available. Task priority effectively remains unchanged, allowing the task to compete for memory space.

- **Stopped** - A stopped task is unable to compete for CPU time because of pending I/O completion, event flag(s) not set, or because the task stopped itself. When stopped, a task's priority effectively drops to zero and the task can be checkpointed by any other task, regardless of that task's priority. If an AST occurs for the stopped task, its normal task priority is restored only for the duration of the AST routine execution; once the AST is completed, task priority returns to zero.

4.1.1 Task State Transitions

This section describes task state transitions.

- **Dormant to Active**

  The following directives cause the Executive to activate a dormant task:

  - A RUN$ directive
  - An RQST$ directive
  - A SPWN$ directive
  - An SDRC$ directive
  - A VSRC$ directive

- An RPOI$ directive
- An SDRP$ directive

- **Ready-to-Run to Blocked**

  The following events cause an active, ready-to-run task to become blocked:

  - An SPND$ directive
  - An unsatisfied Wait For condition
  - Checkpointing of a task out of memory by the Executive

- **Ready-to-Run to Stopped**

  The following events cause an active, ready-to-run task to become stopped:

  - A STOP$S directive is executed, or an RCST$, RRST$, or VRCS$ directive is issued when no data packet is available
  - An unsatisfied Stop For condition
  - An unsatisfied Wait For condition while the task has outstanding buffered I/O

- **Blocked to Ready-to-Run**

  The following events return a blocked task to the ready-to-run state:

  - An RSUM$ directive issued by another task
  - A Wait For condition is satisfied
  - The Executive reads a checkpointed task into memory

- **Stopped to Ready-to-Run**

  The following events return a stopped task to the ready-to-run state, depending upon how the task became stopped:

- A task stopped by the STOP$, RCST$, or VRCS$ directive becomes unstopped by USTP$ directive execution

- A Wait For condition is satisfied for a task with outstanding buffered I/O

- A task stopped for an event flag becomes unstopped when the specified event flag becomes set

- **Active to Dormant**

  The following events cause an active task to become potentially dormant:

  - An EXIT$S, EXIF$, RCVX$, or VRCX$ directive, or an SDRP$ or RPOI$ directive that specifies the exit option

  - An ABRT$ directive

  - A Synchronous System Trap (SST) for which a task has not specified a service routine

- **Blocked to Stopped**

  The following event causes a task that is blocked due to an unsatisfied Wait For condition to become stopped:

  - The task initiates buffered I/O at AST state and then exits from AST state

- **Stopped to Blocked**

  The following event causes a task that is stopped due to an unsatisfied Wait For condition and outstanding buffered I/O to return to a blocked state:

  - Completion of all outstanding buffered I/O

4.2 DIRECTIVE SUMMARY

This section summarizes the directives for parent/offspring tasking and intertask communication.

4.2.1 Parent/Offspring Tasking Directives

There are two classes of parent/offspring tasking directives:

- **Spawning** - Directives that create a connection between tasks
- **Chaining** - Directives that transfer a connection

Three directives can connect a parent task to an offspring task:

- **Spawn**

  This directive requests activation of, and connects to, a specific offspring task.

  An offspring task spawned by a parent task can return current status information or exit status information to a connected parent task.

  Spawn directive options include:

  - Queuing a command line for the offspring task
  - Establishing the offspring task's TI: (terminal)
  - For privileged tasks, designating any terminal as the offspring TI:

- **Connect**

  This directive establishes task communications for synchronizing with the exit status or emit status issued by a task that is already active.

- **Send, Request, and Connect**

  This directive sends data to the specified task, requests activation of the task if it is not already active, and connects to the task.

Two directives support task chaining:

- **Request and Pass Offspring Information**

  This directive allows an offspring task to pass its parent connection to another task, thus making the new task the offspring of the original parent. The RPOI$ directive offers all the options of the Spawn directive.

- **Send Data, Request and Pass Offspring Control Block**

  This directive sends a data packet for a specified task, passes its parent connection to that task, and requests the task if it is not already active.

A parent task can use the Spawn and Connect directives to connect to more than one offspring task. In addition, the parent task can use the directives in any combination to multiply connect to offspring tasks.

An offspring task can be connected to multiple parent tasks. An appropriate data structure, the Offspring Control Block (OCB), is produced (in addition to those already present) each time a parent task connects to the offspring task.

4.2.2 Task Communication Directives

Two directives provide a mechanism for an offspring task to return status to connected parent tasks:

- **Exit With Status**

  This directive in an offspring task causes the offspring task to exit, passing status words to all connected parent tasks connected by a Spawn, Connect, or Send, Request, and Connect directive.

- **Emit Status**

  This directive causes the offspring task to pass status words either to the specified connected task, or to all connected parent tasks if no task is explicitly specified.

When status is passed to tasks in this manner, the parent task no longer remains connected.

Table 4-1 lists the standard offspring task status values that can be returned to parent tasks. Symbols shown in the table are defined in DIRSYM.MAC. They become locally defined when the EXST$ macro is invoked. However, the exit status can be any 16-bit value.

Table 4-1: Offspring Task Status

Symbolic	Value	Severity	Description
EX$WAR	0	Warning	Task succeeded, but irregularities are possible
EX$SUC	1	Success	Results should be as expected
EX$ERR	2	Error	Results are unlikely to be as expected
EX$SEV	4	Severe Error	One or more fatal errors detected, or task aborted

4.3 CONNECTING AND PASSING STATUS

Offspring task exit status can be returned to a connected (parent) task by issuing the Exit With Status directive. Offspring tasks can return status to one or more connected parent tasks at any time by issuing the Emit Status Directive. Note that only connected parent-offspring tasks can pass status.

The means by which a task connects to another task are indistinguishable once the connect process is complete. For example, Task A can become connected to Task B in one of the following ways:

- Task A spawned Task B when Task B was inactive.
- Task A connected to Task B when Task B was active.
- Task A issued a Send, Request, And Connect directive to Task B when Task B was either active or inactive.
- Task A either spawned or connected to Task C, which then chained to Task B by means of either an RPOI$ directive, an SDRP$ directive, or the PROTSK install/run/remove chain option.

Regardless of how Task A became connected to Task B, Task B can pass status information back to Task A, set the event flag specified by Task A, or cause the AST specified by Task A to occur by any means listed below. (Note that once offspring task status is returned to one or more parent tasks, the parent tasks become disconnected unless there are multiple connections; see Section 8.24 for details.)

- Task B issues a successful exit directive. Task A receives a status of EX$SUC.
- Task B is aborted. Task A receives severe error status EX$SEV.
- Task B issues an Exit With Status directive and return status to Task A upon completion of Task B.
- Task B issues an Emit Status directive specifying Task A. If Task A is multiply connected to Task B, the OCBs that contain information about these multiple connections are stored in a FIFO queue. The first OCB is used to determine which event flag, AST address, and exit status block to use.
- Task B issues an Emit Status directive to all connected tasks (no task name specified).

If a task specifies another task in a Spawn or Connect directive, or in a Send, Request, and Connect directive, and then exits, and if status is not yet returned, then this connection's OCB remains queued.

NOTE

This point is important. OCBs are not removed when the parent exits, but only when the offspring exits or emits status. OCBs can quickly exhaust primary pool. To avoid this, remove an offspring's OCBs when the parent exits by forcing the offspring to Emit Status or Exit.

Although the OCB is not removed, it is marked to indicate that the parent task has exited. When this OCB is subsequently dequeued by an Emit Status directive, or any type of exit, no action is taken because the parent task has exited.

This procedure helps a multiply-connected task remain synchronized with the requests, regardless of whether or not the parent has exited.

Note that when you invoke the Emit Status directive and the specified task is multiply connected to the issuing task, P/OS uses the first (oldest) OCB in the queue to return status.

Table 4-2 shows examples of using directives for intertask synchronization. (Macro call form for directives are shown.) Task A is the parent task and Task B is the offspring task.

Table 4-2: Intertask Synchronization Examples

Task A	Task B	Action
SPWN$	EXST$	Task A spawns Task B. When Task B completes, it returns status to Task A.
CNCT$	EXST$	Task A connects to active Task B. When Task B completes, it returns status to Task A.
SDRC$	RCVX$ EMST$	Task A sends data to Task B, requests Task B if it is not active, and connects to Task B. Task B receives the data, does some processing based on the data, returns status to Task A (possibly setting an event flag or declaring an AST), and becomes disconnected from Task A.
SDRC$ USTP$	RCST$ EMST$	Task A sends data to Task B, requests Task B if it is not active, connects to Task B, and unstops Task B (if Task B previously could not dequeue the data packet). Task B receives the data, does some processing based on the data, and returns status to Task A (possibly setting an event flag or declaring an AST).
SDAT$ USTP$	RCST$	Task A queues a data packet for Task B and unstops Task B; Task B receives the data.
SPWN$	RPOI$ SDRP$	Task A spawns Task B. Task B chains to Task C by issuing an RPOI$ or an SDRP$ directive. Task A is now Task C's parent. Task A is no longer connected to Task B.

CHAPTER 5

USING MEMORY MANAGEMENT SERVICES

The Professional's word size, 16 bits, limits the total virtual address space of a task to 32K words. That is, the highest address that a task can refer to is 177777 (octal).

P/OS provides mechanisms that allow a user task to overcome this limitation. Task overlays and cluster libraries are two such mechanisms. (Both are fully described in the *RSX-11M/M-PLUS and Micro/RSX Task Builder Manual* and the *Guide to Writing a P/OS I/O Driver and Advanced Programmer's Notes.*)

Another mechanism is the extended addressing capability provided by P/OS memory management directives. These directives overcome the 32K-word addressing restriction by allowing the task to dynamically change the physical locations that are referred to by a given range of addresses.

This chapter describes concepts with which you must be familiar in order to use the memory management directives.

5.1 ADDRESS MAPPING

Mapping is the process of associating task virtual addresses with physical memory locations. On the Professional, built-in hardware called the memory management unit transparently performs this *virtual to physical* mapping. Since the memory management unit performs the mapping, you cannot know where a task resides in physical memory.

5.1.1 Physical, Logical, and Virtual Address Space

The following concepts provide a basis for understanding the functions performed by the memory management directives:

- **Physical Address Space**

  A task's physical address space is the entire set of physical memory addresses.

- **Logical Address Space**

  A task's logical address space is the total amount of physical memory to which the task has access rights. This includes various areas called regions (see Section 5.3). Each region occupies a contiguous block of memory.

- **Virtual Address Space**

  A task's virtual address space corresponds to the 32K-word address range imposed by the 16-bit word length. The task can divide its virtual address space into segments called virtual address windows (see Section 5.2).

If the capabilities supplied by the memory management directives were not available, a task's virtual address space and logical address space would directly correspond; a single virtual address would always point to the same logical location. Both types of address space would have a maximum size of 32K words.

However, the ability of the memory management directives to assign or map a range of virtual addresses (a window) to different logical areas (regions) enables you to extend a task's logical address space beyond 32K words.

5.2 WINDOWS

To manipulate the mapping of virtual addresses to various logical areas, you must first divide a task's 32K words of virtual address space into *virtual address windows*.

Each window encompasses a contiguous range of virtual addresses, which must begin on a 4K-word boundary; that is, the first address must be a multiple of 4K. The number of windows defined by a task can range from 1 through 8. For all tasks, window 0 is not available to the task for remapping. The size of each window can range from a minimum of 32 words through a maximum of 32K words.

A task that includes directives to manipulate address windows dynamically must have window blocks set up in its task header. The Executive uses window blocks to identify and describe each currently existing window.

You can specify the required number of additional window blocks--the number of windows created by the memory management directives--to be set up by the Task Builder. (See the *RSX-11M/M-PLUS and Micro/RSX Task Builder Reference Manual*.) The number of blocks that you specify should equal the maximum number of windows that will exist at a time during task execution

A window's identification is a number from 0 through 7 (decimal) for user windows; it is an index to the window's corresponding window block. The address window identified by 0 is the window that maps the task's header and root segment. The Task Builder automatically creates window 0, which is mapped by the Executive and cannot be specified in any directive.

Figure 5-1 shows the virtual address space of a task divided into four address windows--0, 1, 2, and 3. The shaded areas indicate portions of the address space not included in any window (9K through 12K and 23K through 24K). Addresses within the ranges corresponding to the shaded areas cannot be used.

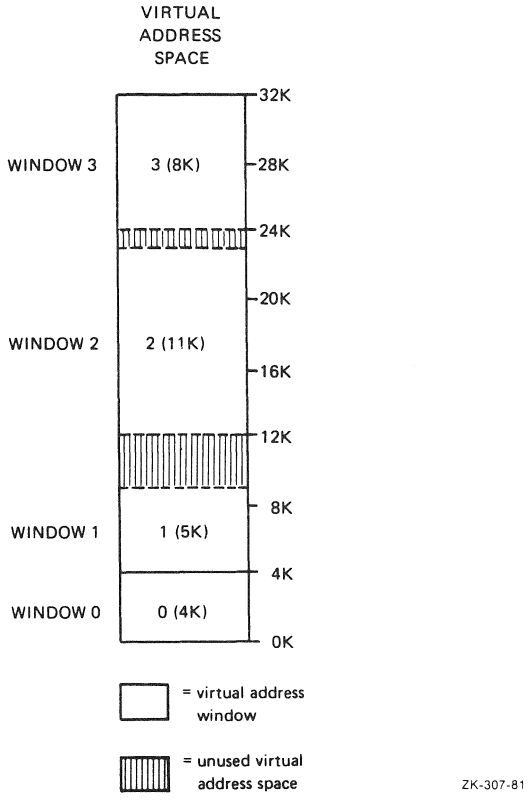


Figure 5-1: Virtual Address Windows

When a task uses memory management directives, the Executive views the relationship between the task's virtual and logical address space in terms of windows and regions. Unless a virtual address is part of an existing address window, reference to that address generates an illegal address trap. Similarly, a window can be mapped only to an area that is all or part of an existing region within the task's logical address space (see Section 5.3).

Once a task has defined windows and regions, it can issue memory management directives to perform operations such as:

- Map a window to all or part of a region
- Unmap a window from one region to map it to another region
- Unmap a window from one part of a region in order to map it to another part of the same region

5.3 REGIONS

A region is a portion of physical memory to which a task has, or may potentially have, access. The current window-to-region mapping context determines that part of a task's logical address space that the task can access at one time.

Regions fall into several categories:

- **Region Creation: Static and Dynamic**

  A *static* region is a region that you create separately from tasks that accesses it. To create a static region, you use the task builder. Before a task can access a static region, you must install the region into the system. Note that the Executive dynamically loads static regions whenever tasks request to map them. The *RSX-11M/M-PLUS and Micro/RSX Task Builder Manual* describes static regions in detail.*

  A *dynamic region* is a region that your task creates during execution, using the memory management directive CRRG$. When you build a task that creates a dynamic region, you specify how many additional window blocks the Task Builder places in the task header. See the description of CRRG$ in Chapter 8 for details on creating dynamic regions.

---

* The *Task Builder Manual* uses different terminology. It refers to a static region as a *resident region*.

- **Region Access: Shared and Unshared**

  A *shared region* is a region that more than one task can map to at a time. Shared regions are useful because they make more efficient use of physical memory than unshared regions.

  An *unshared region* can only be mapped by one task. The method you use to build the region, and the protection mask you specify for the region, determine whether or not it can be shared. See Section 5.3.2 for a description of region protection masks. Note that unshared dynamic regions are typically unnamed.

- **Region Contents: Common, Library, and Task**

  A *common* is a region that contains read-write data.* A *library* is a region that contains read-only code. Commons and libraries are almost always shared, although this is not a requirement.

  A *task region* is a special region. It is a contiguous block of memory in which a task executes. In other words, the task region is that portion of a task's logical address space that contains the actual executable portion of a task.

Tasks refer to a region by means of a region ID returned to the task by the Executive. A region ID from 0 through 23 refers to a task's static attachment. Region ID 0 always refers to a task's task region. Region ID 1 always refers to the read-only (pure code) portion of multiuser tasks. All other region IDs are actually addresses of the attachment descriptor maintained by the Executive in the system dynamic storage area.

Figure 5-2 shows a sample collection of regions that could make up a task's logical address space at some given time. The header and root segment are always part of the task region. Since a region occupies a contiguous area of memory, each region is shown as a separate block.

Figure 5-3 illustrates a possible mapping relationship between the windows and regions shown in Figures 7-1 and 7-2.

---

* Never use the Extend Task task builder option when building a static common.

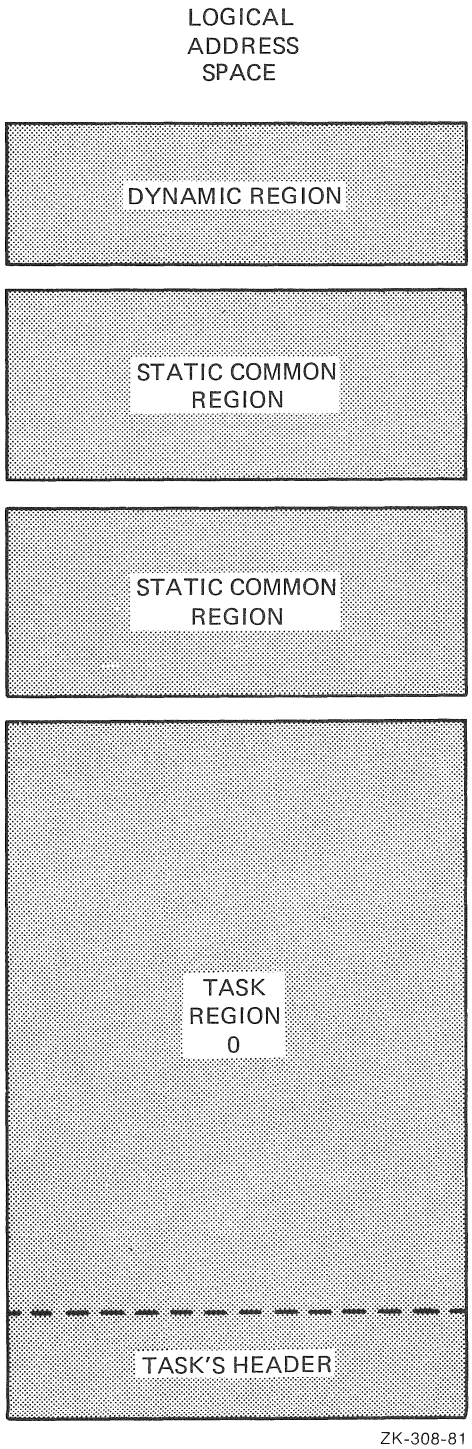


Figure 5-2: Region Definition Block

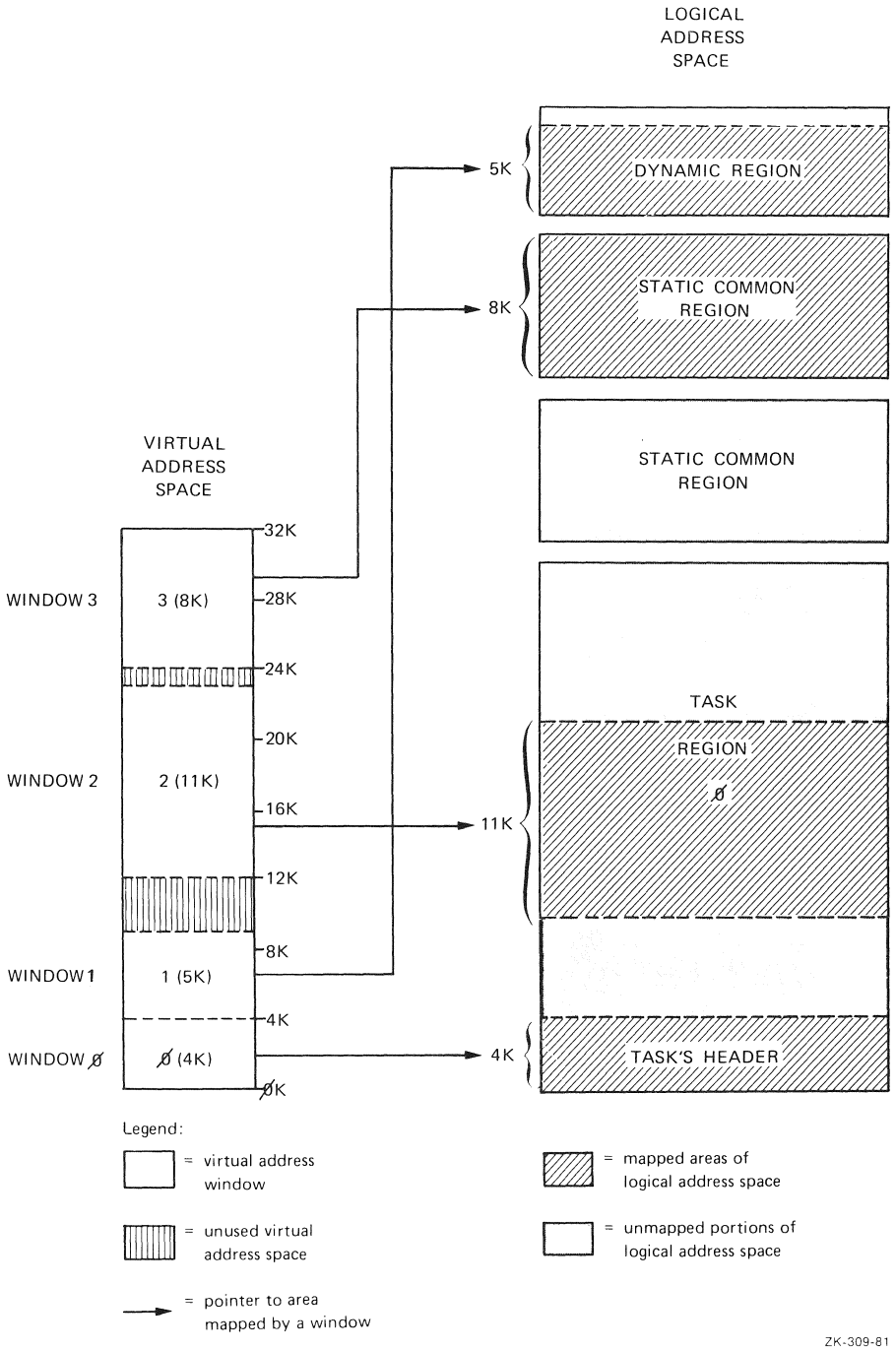


Figure 5-3: Mapping Windows to Regions

5.3.1 Attaching to Regions

Attaching is the process by which a region becomes part of a task's logical address space. A task can only map a region that is part of the task's logical address space. There are three ways to attach a task to a region:

- All tasks are automatically attached to regions that are linked to them at task-build time.
- A task can issue a directive to attach itself to a named static common region or a named dynamic region.
- A task can request the Executive to attach another specified task to any region within the logical address space of the requesting task.

Attaching identifies a task as a user of a region and prevents the system from deleting a region until all user tasks have been detached from it. (Note that fixed tasks do not automatically become detached from regions upon exiting.)

NOTE

Each Send By Reference directive issued by a sending task creates a new attachment descriptor for the receiving task. However, multiple Send By Reference directives that refer to the same region require only one attachment descriptor. After the receiving task issues a series of Receive By Reference directives and receives all pending data requests, the task should detach the region to return the attachment descriptors to pool.

You can avoid multiple attachment descriptors when sending and receiving data by reference. Setting the WS.NAT bit in the Window Definition Block (see Section 5.5.2) causes the Executive to create a new attachment descriptor for that region only if necessary (that is, if the task is currently not attached to the region).

5.3.2 Region Protection

A task cannot indiscriminately attach to any region. Each region has a protection mask to prevent unauthorized access. The mask indicates the types of access (read, write, extend, delete) allowed for each category of user (system, owner, group, world).

The Executive checks that the requesting task's User Identification Code (UIC) allows it to make the attempted access. The attempt fails if the protection mask denies that task the access it wants.

To determine when tasks can add to their logical address space by attaching regions, the following points must be considered (note that all considerations presume there is no protection violation):

- Any task can attach to a named dynamic region, provided the task knows the name. The only way to map to an unnamed region created by another task is to have that other task issue to your task a Send by Reference (SREF$) directive for the unnamed region.

- Any task can issue a Send By Reference directive to attach another task to any region, including the sender's task region. The reference sent includes the access rights with which the receiving task attaches to the region. The sending task can only grant access rights that it has itself.

- Any task can map to a named static common region.

5.4 DIRECTIVE SUMMARY

This section briefly describes the function of each memory management directive. Chapter 8 defines all the directives in detail.

5.4.1 Create Region Directive (CRRG$)

The Create Region directive creates a dynamic region in a designated system-controlled partition (for example, GEN), and optionally attaches the issuing task to it.

5.4.2 Attach Region Directive (ATRG$)

The Attach Region directive attaches the issuing task to a static common region or to a named dynamic region.

5.4.3 Detach Region Directive (DTRG$)

The Detach Region directive detaches the issuing task from a specified region. Any of the task's address windows that are mapped to the region are automatically unmapped.

5.4.4 Create Address Window Directive (CRAW$)

The Create Address Window directive creates an address window, establishes its virtual address base and size, and optionally maps the window. Any other windows that overlap with the range of addresses for the new window are first unmapped and then eliminated.

5.4.5 Eliminate Address Window Directive (ELAW$)

The Eliminate Address Window directive eliminates an existing address window, unmapping it first if necessary.

5.4.6 Map Address Window Directive (MAP$)

The Map Address Window directive maps an existing window to an attached region. The mapping begins at a specified offset from the start of the region and goes to a specified length. If the window is already mapped elsewhere, the Executive unmaps it before carrying out the map assignment described in the directive.

5.4.7 Unmap Address Window Directive (UMAP$)

The Unmap Address Window directive unmaps a specified window. After the window is unmapped, its virtual address range cannot be referenced until the task issues another mapping directive.

5.4.8 Send By Reference Directive (SREF$)

The Send By Reference directive inserts a packet containing a reference to a region into the receive queue of a specified task. The receiver task is automatically attached to the region referred to.

5.4.9 Receive By Reference Directive (RREF$)

The Receive By Reference directive requests the Executive first to select the next packet from the receive-by-reference queue of the issuing task, and then to make the information in the packet available to the task. Optionally the directive can map a window to the referenced region or cause the task to exit if the queue does not contain a receive-by-reference packet.

5.4.10 Get Mapping Context Directive (GMCX$)

The Get Mapping Context directive causes the Executive to return to the issuing task a description of the current window-to-region mapping assignments. The description is in a form that enables the user to restore the mapping context through a series of Create Address Window directives.

5.4.11 Get Region Parameters Directive (GREG$)

The Get Region Parameters directive causes the Executive to supply the issuing task with information about either its task region (if no region ID is given) or an explicitly specified region.

5.5 USER DATA STRUCTURES

Most memory management directives are individually capable of performing a number of separate actions. For example, a single Create Address Window directive can unmap and eliminate as many as seven conflicting address windows, create a new window, and map the new window to a specified region. The complexity of the directives requires a special means of communication between the user task and the Executive. The communication is achieved through data structures that:

- Allow the task to specify which directive options it wants the Executive to perform
- Permit the Executive to provide the task with details about the outcome of requested actions

There are two types of user data structures that correspond to the two key elements (regions and address windows) manipulated by the directives. The structures are called:

- The Region Definition Block (RDB)
- The Window Definition Block (WDB)

Every memory management directive, except Get Region Parameters, uses one of these structures as its communications area between the task and the Executive. Each directive issued includes in the DPB a pointer to the appropriate definition block.

The task assigns symbolic address offset values that point to locations within an RDB or a WDB. The task can change the contents of these locations to define or modify the directive operation. After the Executive has carried out the specified operation, it assigns values to various locations within the block to describe the actions taken and to provide the task with information useful for subsequent operations.

5.5.1 Region Definition Block (RDB)

Figure 5-4 illustrates the format of an RDB.

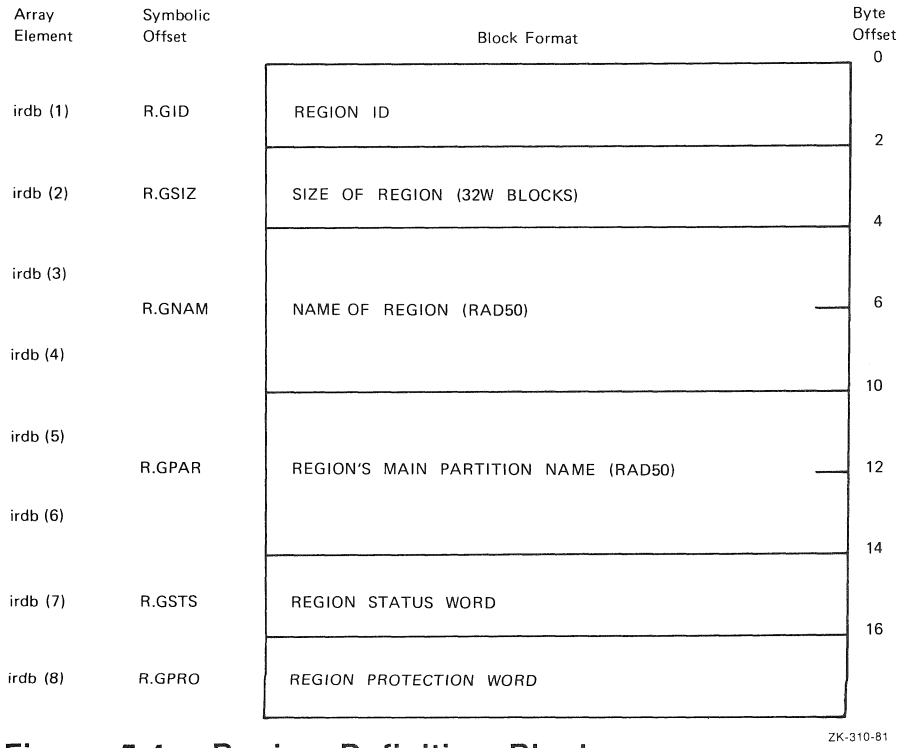


Figure 5-4: Region Definition Block

In addition to the symbolic offsets shown in Figure 5-4, the region status word R.GSTS contains defined bits that can be set or cleared by the Executive or the task. Table 5-1 shows the bits and their definitions. The symbols shown in the table are defined by the RDBDF$ macro, as described in Section 5.5.1.1.

Table 5-1: Region Status Word (R.GSTS) Bit Definitions

Bit n	Definition
RS.CRR=100000	Region was successfully created.
RS.UNM=40000	At least one window was unmapped on a detach.
RS.MDL=200	Mark region for deletion on last detach. When a region is created by a CRRG$ directive, the region is normally marked for deletion on last detach. However, if RS.NDL is set when the CRRG$ directive is executed, the region is not marked for deletion. Subsequent execution of a DTRG$ directive with RS.MDL set marks the region for deletion.
RS.NDL=100	Created region is not to be marked for deletion on last detach. (Unnamed regions are always deleted on the last detach.)
RS.ATT=40	Attach to created region.
RS.NEX=20	Created region is not extendable.
RS.DEL=10	Delete access desired on attach.
RS.EXT=4	Extend access desired on attach.
RS.WRT=2	Write access desired on attach.
RS.RED=1	Read access desired on attach.

The three memory management directives that require a pointer to an RDB are:

Create Region (CRRG$)
Attach Region (ATRG$)
Detach Region (DTRG$)

When a task issues one of these directives, the Executive clears the four high-order bits in the region status word of the appropriate RDB. After completing the directive operation, the Executive sets the RS.CRR or RS.UNM bit to indicate to the task what actions were taken. The Executive never modifies the other bits in the region status word.

5.5.1.1 Using Macros to Generate an RDB - The system provides two macros, RDBDF$ and RDBBK$, to generate and define an RDB. RDBDF$ defines the offsets and status word bits for a region definition block; RDBBK$ then creates the actual region definition block.

The format of RDBDF$ is:

```
RDBDF$
```

Since RDBBK$ automatically invokes RDBDF$, you need specify only RDBBK$ in a module that creates an RDB. The format of the call to RDBBK$ is:

```
RDBBK$   siz,nam,par,sts,pro
```

siz
The region size in 32-word blocks.

nam
The region name (RAD50).

par
The name of the partition in which to create the region (RAD50).

sts
Bit definitions of the region status word.

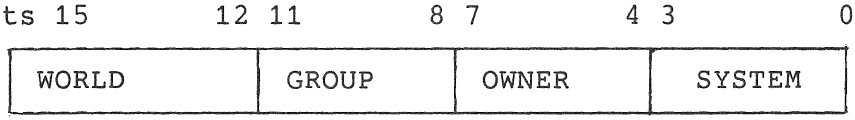
pro
The region's default protection word.

The sts argument sets specified bits in the status word R.GSTS. The argument normally has the following format:

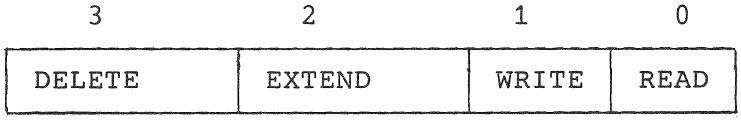
```
<bit1[!...!bitn]>
```

bit
A defined bit to be set.

The argument pro is an octal number. The 16-bit binary equivalent specifies the region's default protection as follows:



Each of these four categories has four bits, with each bit representing a type of access:



A bit value of 0 indicates that the specified type of access is to be allowed; a bit value of 1 indicates that the specified type of access is to be denied.

The macro call

```
RDBBK$   102.,ALPHA,GEN,<RS.NDL!RS.ATT!RS.WRT!RS.RED>,167000
```

expands to:

```
.WORD    0
.WORD    102.
.RAD50   /ALPHA/
.RAD50   /GEN/
.WORD    0
.WORD    RS.NDL!RS.ATT!RS.WRT!RS.RED
.WORD    167000
```

If a Create Region directive pointed to the RDB defined by this expanded macro call, the Executive would create a region 102 (decimal) 32-word blocks in length, named ALPHA, in a partition named GEN. The defined bits specified in the sts argument tell the Executive:

- Not to mark the region for deletion on the last detach
- To attach region ALPHA to the task issuing the directive macro call
- To grant read and write access to the attached task

The protection word specified as 167000 (octal) assigns a default protection mask to the region. The octal number, which has a binary equivalent of 1110 1110 0000 0000, grants access as follows:

```
World  (1110) - Read access only
Group  (1110) - Read access only
Owner  (0000) - All access
System (0000) - All access
```

If the Create Region directive is successful, the Executive will first return to the issuing task a region ID value in the location accessed by symbolic offset R.GID, and then will set the defined bit RS.CRR in the status word R.GSTS.

Table 5-2: RDB Array Format

Word	Contents
irdb(1)	Region ID
irdb(2)	Size of the region in 32-word blocks
irdb(3) irdb(4)	Region name (2 words in Radix-50 format)
irdb(5) irdb(6)	Name of the partition that contains the region (2 words in Radix-50 format)
irdb(7)	Region status word
irdb(8)	Region protection code

5.5.1.2 Using High-Level Languages to Generate an RDB - When programming in high-level languages, you must create an eight-word, single-precision integer array as the RDB to be supplied in the subroutine calls:

```
CALL ATRG     (Attach Region directive)
CALL CRRG     (Create Region directive)
CALL DTRG     (Detach Region directive)
```

An RDB array has the format shown in Table 5-2. You can modify the region status word irdb(7) by setting or clearing the appropriate bits. See the list in Section 5.5.1 that describes the defined bits. The bit values are listed beside the symbolic offsets.

5.5.2 Window Definition Block (WDB)

Figure 5-5 illustrates the format of a WDB.

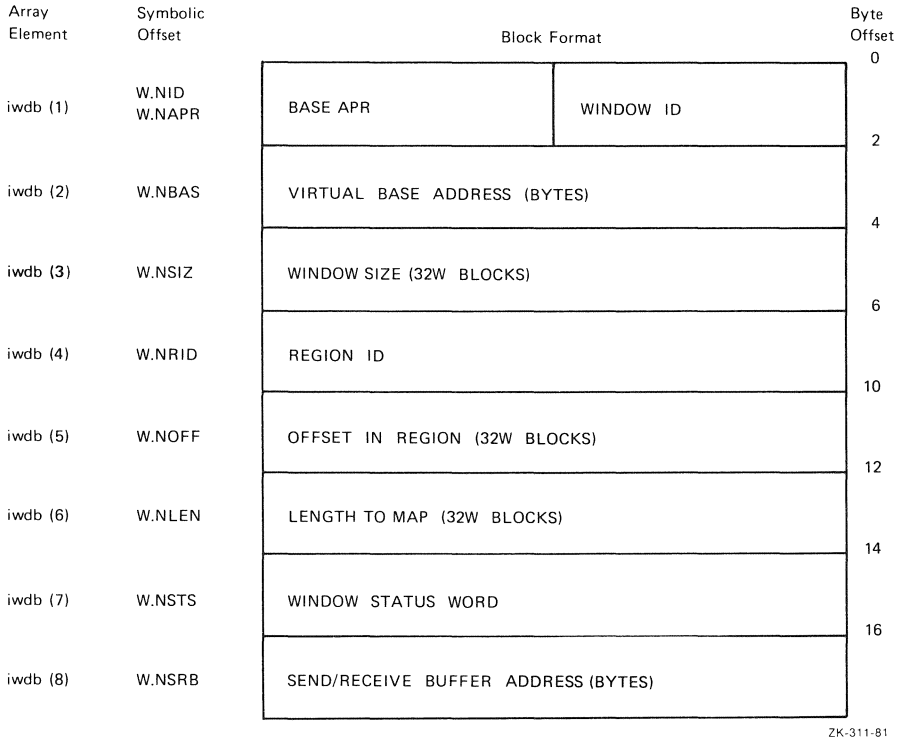


Figure 5-5: Window Definition Block

The WDB consists of a number of symbolic address offsets to WDB locations. One location is the window status word W.NSTS, which contains defined bits that can be set or cleared by the Executive or the task. Table 5-3 lists the bits and their definitions. The symbols shown in the table are defined by the WDBDF$ macro, as described in Section 5.5.2.1.

The following directives require a pointer to a WDB:

Create Address Window (CRAW$)
Eliminate Address Window (ELAW$)
Map Address Window (MAP$)
Unmap Address Window (UMAP$)
Send By Reference (SREF$)
Receive By Reference (RREF$)

When a task issues one of these directives, the Executive clears the four high-order bits in the window status word of the appropriate WDB. After completing the directive operation, the Executive can then set any of these bits to tell the task what actions were taken. The Executive never modifies the other bits.

5.5.2.1 Using Macros to Generate a WDB - The system provides two macros, WDBDF$ and WDBBK$, to generate and define a WDB. WDBDF$ defines the offsets and status word bits for a window definition block; WDBBK$ then creates the actual window definition block.

The format of WDBDF$ is:

```
WDBDF$
```

Since WDBBK$ automatically invokes WDBDF$, you need specify only WDBBK$ in a module that generates a WDB. The format of the call to WDBBK$ is:

```
WDBBK$   apr,siz,rid,off,len,sts,srb
```

apr
A number from 0 through 7 that specifies the window's base Active Page Register (APR). The APR determines the 4K boundary on which the window is to begin. APR 0 corresponds to virtual address 0, APR 1 to 4K, APR 2 to 8K, and so on.

siz
The size of the window in 32-word blocks.

rid
A region ID.

Table 5-3: Window Status Word (W.NSTS) Bit Definitions

Bit	Definition
WS.CRW=100000	Address window was successfully created.
WS.UNM=40000	At least one window was unmapped by a Create Address Window, Map Address Window, or Unmap Address Window directive.
WS.ELW=20000	At least one window was eliminated in a Create Address Window or Eliminate Address Window directive.
WS.RRF=10000	Reference was successfully received.
WS.NBP=4000	Do not bypass cache for CRAW$ directives.
WS.BPS=4000	Always bypass cache for MAP$ directives.
WS.RES=2000	Map only if resident.
WS.NAT=1000	Create attachment descriptor only if necessary (for Send By Reference directives).
WS.64B=400	Defines the task's permitted alignment boundaries--0 for 256-word (512-byte) alignment, 1 for 32-word (64-byte) alignment.
WS.MAP=200	Window is to be mapped in a Create Address Window or Receive By Reference directive.
WS.RCX=100	Exit if no references to receive.
WS.DEL=10	Send with delete access.
WS.EXT=4	Send with extend access.
WS.WRT=2	Send or map with write access.
WS.RED=1	Send with read access.

off
: The offset (in 32-word blocks) within the region to be mapped.

len
: The length (in 32-word blocks) within the region to be mapped (defaults to the value of siz).

sts
: The bit definitions of the window status word.

srb
: A send/receive buffer virtual address.

The argument sts sets specified bits in the status word W.NSTS. The argument normally has the following format:

```
<bit1[!...!bitn]>
```

bit
: A defined bit to be set.

The macro call

```
WDBBK$   5,76.,0,50.,,<WS.64B!WS.MAP!WS.WRT>
```

expands to:

```
.BYTE    0,5    (Window ID returned in low-order byte)
.WORD    0      (Base virtual address returned here)
.WORD    76.
.WORD    0
.WORD    50.
.WORD    0
.WORD    WS.64B!WS.MAP!WS.WRT
.WORD    0
```

If a Create Address Window directive pointed to the WDB defined by the expanded macro call, the Executive would:

- Create a window 76 (decimal) blocks long beginning at APR 5--virtual address 20K or 120000 (octal).
- Map the window with write access (<WS.MAP!WS.WRT>) to the issuing task's task region (because the macro call specified 0 for the region ID).

- Start the map 50 (decimal) blocks from the base of the region, and map an area either equal to the length of the window--76 (decimal)--or to the length remaining in the region, whichever is smaller (because the macro call defaulted the len argument) and align the window on a 64-byte boundary.

- Return values to the symbolic W.NID (the window's ID) and W.NBAS (the window's virtual base address).

5.5.2.2 Using High-Level Language to Generate a WDB - You must create an eight-word, single precision array as the WDB to be supplied in the subroutine calls:

```
CALL CRAW     (Create Address Window directive)
CALL ELAW     (Eliminate Address Window directive)
CALL MAP      (Map Address Window directive)
CALL UNMAP    (Unmap Address Window directive)
CALL SREF     (Send By Reference directive)
CALL RREF     (Receive By Reference directive)
```

A WDB array has the format shown in Table 5-4. You can modify the window status word iwdb(7) by setting or clearing the appropriate bits. See the list in Section 5.5.2 that describes the defined bits. The bit values are listed alongside the symbolic offsets.

Notes:

- The contents of bits 8 through 15 of iwdb(1) must normally be set without destroying the value in bits 0 through 7 for any directive other than the Create Address Window.

- A call to GETADR (see Section 7.4.4) can be used to set up the address of the send/receive buffer. For example:

  ```
  CALL GETADR(IWDB,,,,,,,,IRCVB)
  ```

  This call places the address of buffer IRCVB in array element 8. The remaining elements are unchanged. The subroutines SREF and RREF also set this value. If you use the SREF and RREF routines, you do not need to use GETADR.

Table 5-4: WDB Array Format

Word	Contents
iwdb(1)	Bits 0 through 7 contain the window ID; bits 8 through 15 contain the window's base APR.
iwdb(2)	Base virtual address of the window.
iwdb(3)	Size of the window in 32-word blocks.
iwdb(4)	Region ID.
iwdb(5)	Offset length (in 32-word blocks) within the region at which map begins.
iwdb(6)	Length (in 32-word blocks) mapped within the region.
iwdb(7)	Window status word.
iwdb(8)	Address of send/receive buffer.

5.5.3 Assigned Values or Settings

The exact values or settings assigned to individual fields within the RDB or the WDB vary according to each directive. Fields that are not required as input can have any value when the directive is issued. Chapter 4 describes which offsets and settings are relevant for each memory management directive. The values assigned by the task are called input parameters, whereas those assigned by the Executive are called output parameters.

5.6 PRIVILEGED TASKS

A privileged task can be mapped to the Executive and the I/O page; the system normally dedicates six APRs to this mapping (five for the Executive and primary pool and one for the I/O page). Use the taskbuilder switch /PR to build a privileged task mapped to the Executive.

5.7 FAST REMAP OPERATIONS

Fast remap operations provide a high-performance method for a task to change the offset and length mapping of a currently mapped region. It can increase the performance of your task if you are normally mapped to a given region and are remapping a task's window to that region frequently.

For example, you can normally perform a fast remap operation in approximately one tenth the execution time of a remap using the MAP$ directive.

Fast remap provides the the same level of region access control and protection afforded by the MAP$ directive.

5.7.1 Performing a Fast Remap

To perform fast remap operations, you must task build with either the /FAST_MAP qualifier (on the LINK command), or the /FM qualifier (on your TKB command line). The qualifier is available beginning with Version 3.0 Tool Kits.

To actually perform the fast remap, issue an IOT instruction within your code. The Executive interprets all IOT instructions as fast remap requests when your task has been built with the /FAST_MAP or /FM qualifier.

The system returns status values in R0.

Input Parameters

You specify input parameters in user registers R0, R1, and R2:

R0

R0 contains two types of information:

- High bit--This bit, 100000 (octal) is a flag that, if set, indicates that you have specified a length change request. If clear, then the flag indicates that you have not specified a length change request--the operation uses the existing window length.

- Remaining bits--The remaining bits of R0 constitute the ID of the first APR corresponding to the base of the the already mapped window to be remapped. See Table 5-5 for a list of ID values for APRs. Also, see Section 5.7.2 for requirements concerning the window to be mapped.

The following sample octal value for R0 indicates that you are making a length change request, and that APR 2 corresponds to the base APR of the window to be remapped.

100020

R1

R1 contains the offset within the region in units of 32 (decimal) word blocks.

R2

R2 is optional; it contains the length of the region segment to be remapped, also in units of 32. word blocks.

The system preserves R4 and R5, but destroys the initial contents of R0, R1, R2, and R3 during the fast remap operation.

Table 5-5: ID Values for APRs

APR	ID Value (Octal)	User Mapping Register
0	APR 0 not allowed	--
1	10	UISAR1
2	20	UISAR2
3	30	UISAR3
4	40	UISAR4
5	50	UISAR5
6	60	UISAR6
7	70	UISAR7

5.7.2 Requirements for Using Fast Remap

Note the following requirements:

- The window to be remapped must already be mapped, either via the MAP$ directive, or via the CRAW$ directive with WS.MAP=1 in the WDB.

- You cannot change the region access of the original MAP$ request. For example,if you did not map the region for write access, you cannot write to the region after the fast remap operation.

- If you specify the length of the region segment to be mapped as an input parameter, the length cannot be larger than either of the following:

  - The window size

  - The length remaining between the specified offset within the region and the end of the region

- If the length of the region segment to be remapped is zero and you specify a length change in a fast remap operation, the length defaults to the smaller of:

  - The window size

  - The length remaining between the specified offset within the region and the end of the region

- Your task cannot use IOT intructions for purposes other than fast remap operations if you task build it with the /FAST_MAP or /FM qualifiers.

5.7.3 Status Codes for Fast Remap

The system returns the following status information in R0 after a fast remap operation:

IS.SUC	Success.
IE.ALG	Your task specified an invalid region offset and length combination.
IE.NVW	Your task specified an invalid ID, which did not correspond to the base of a previously-mapped window.

CHAPTER 6

SYSTEM UTILITY MODULES (POSSUM)

P/OS provides a set of callable routines that perform various useful functions. The routines reside in a resident library called POSSUM (P/OS System Utility Modules). Some of the routines use system tasks called *servers* to perform their functions.

Table 6-1 shows each POSSUM routine, the server it uses (if any), and the function it performs.

Table 6-1: POSSUM Routines

Routine	Server	Description
PROATR	None	Gets or sets a file's attributes.
PRODIR	CREDEL	Creates or deletes a directory.
PROFBI	SUMFBI	Initializes and checks for bad blocks on a disk or diskette, and formats hard disks.
PROLOG	None	Translates, creates, and deletes a logical name.
PROTSK	INSREM	Installs, removes, or fixes a task or common.
PROVOL	SUMPBB	Performs operations related to volume mount/dismount, bootstrapping, and free disk space and file headers.

Another callable system routine, PROLOD, allows you to load a user-written driver into P/OS. Although its entry point is in the POSSUM library, we describe PROLOD in the *Guide to Writing a P/OS I/O Driver and Advanced Programmer's Notes*.

6.1 LINKING PROGRAMS WITH POSSUM

To use the routines, you must link your program with the POSSUM resident library.* You can include POSSUM as part of a cluster of libraries with RMSRES and other libraries. See Chapter 1 and the *RSX-11M/M-PLUS and Micro/RSX Task Builder Manual* for details on cluster libraries.

You can specify either of the options LIBR or CLSTR in your task build command (.CMD) file to include the POSSUM library in your task:

- Use the LIBR option to link a task to the POSSUM resident library:

  ```
  LIBR=POSSUM:RO
  ```

- Use the CLSTR option to link a task to a cluster library, which includes the POSSUM resident library:

  ```
  CLSTR=POSSUM,OTHER:RO
  ```

  The cluster library is null rooted, and thus it does not have to be a default member of a cluster.

6.1.1 Impact of POSSUM on Your Task Image

Note the following regarding POSSUM's impact on your task image:

- POSSUM is a position-independent, vectored library.
- The library uses only one APR.
- The impure area associated with POSSUM, which is allocated from your task's root, is 336 (decimal) bytes.

---

* When you link programs to run on the Professional, invoke the Task Builder using the name PAB (Professional Application Builder) rather than TKB.

6.2 CONVENTIONS FOR CALLABLE SYSTEM ROUTINES

This section defines the general mechanism used for calling all the defined system routines in the POSSUM resident library.

6.2.1 PDP-11 R5 Calling Convention

The R5 calling convention applies not only to the POSSUM routines, but to the high-level language routines described in Chapter 8 as well.

NOTE

> In addition to using the R5 calling convention, POSSUM preserves all registers, unlike the high-level language routines described in Chapter 8.

Your program must use register 5 (R5) to pass the address of an argument list that resides in your task's data space. The argument list itself is of variable length, so that only the necessary arguments are passed.

The general MACRO-11 coding sequence of the call follows.

Instruction space coding sequence:

```
        MOV     #ARGLST,R5      ; argument list address to pass
        JSR     PC,SUB          ; call the subroutine
```

Data space coding sequence:

```
ARGLST: .BYTE   NUMBER,0        ; NUMBER is number of arguments
                                ; following in list
        .WORD   ADDR1           ; address of 1st argument
        ...                     ; other arguments
        .WORD   ADDRn           ; the nth argument
```

For high-level languages that support the R5 calling sequence (such as BASIC-PLUS-2 or FORTRAN-77), see your language reference manual or user guide for correct syntax. High-level language examples in this chapter are written in BASIC-PLUS-2.

In BASIC-PLUS-2, you can invoke the previous MACRO-11 call as follows:

```
    120         CALL SUB BY REF (ADDR1%,...,ADDRn%)
```

BASIC-PLUS-2 internally formats an R5 calling block and issues the call to the system routine for you.

Note that all arguments passed to the system routines are by reference. This means that you are passing the address of the value in your program to the routine in POSSUM (not the actual value).

6.2.2 Other Conventions for POSSUM Routines

You must follow certain conventions when calling the POSSUM routines:

- Every routine shares a common format in that the first argument (status) is the address of an eight-word Status Control Block to which the routines return completion status. The Status Control Block is always 8 words in length, so be sure to allocate the proper amount of space in your program.

- Every routine requires a request parameter. All of the routines are multipurpose, and this one-word request parameter is the method for specifying which options to execute.

- When specifying either a device or filename string as a required element in an argument list, always specify the accompanying size field in bytes. (A byte corresponds to one ASCII character.)

- A task cannot call a POSSUM routine from AST state. With some exceptions in the PROTSK routine, all of the POSSUM routines are synchronous.

- POSSUM disables ASTs for the duration of a call. On return from the call, POSSUM reenables ASTs if the ASTs were enabled on entry.

6.2.3 Status Control Block Format

The eight-word Status Control Block has the following format:

word 0 **Status parameter count.** This is the count of the number of status parameters passed back to the Status Control Block upon completion of the routine.

word 1 **Overall call status.** This is a one-word value defined as follows:

+1 = *Success*

-1 = *Directive Status Error.* The actual $DSW error is in word 2.

-2 = *QIO error.* The contents of the two-word QIO status block are in words 2 and 3.

-3 = *RMS-11 error.* The RMS-11 STS and STV fields are returned in words 2 and 3.

-4 = *Server-specific error.* The contents of words 2 through 7 are defined for each routine in Sections 6.4 through 6.9.

-5 = *Interface error.* An error occured when trying to interpret the argument block. Currently, one of the following values would be in word 2:

-1 = Feature not supported.

-2 = Impure area is invalid, or missing. Usually indicates that you have not correctly taskbuilt your program.

-3 = Invalid number of parameters (too few or too many).

-4 = Server not installed.

-5 = Illegal device specification.

-6 = User buffer too small for returned data.

-7 = Incompatibility between POSSUM library and task. Relink task to resolve the incompatibility.

word 2 - 7 As defined above.

NOTE

Words 2 through 7, depending on their use, can represent integers (such as error codes) or ASCII strings (such as volume labels in the case of the PROFBI and PROVOL routines).

6.3 FORMAT OF POSSUM ROUTINE DESCRIPTIONS

Each routine description includes most or all of the following elements:

Format

The call for each routine shows the name of the subroutine and the definitions for each of its parameters, given in a FORTRAN-like CALL statement. Note, however, that not all high-level languages use the CALL statement to execute external routines.

Status Codes

If a routine returns server-specific errors, this section lists those status codes the server can return.

Notes

The notes section contains additional important information.

Example

Each description includes a BASIC-PLUS-2 example of the routine.

The following sections describe each callable routine in detail.

6.4 PROATR

The PROATR routine provides two forms of accessing file attributes. You can use PROATR to:

- Get attributes of a file
- Set attributes of a file

Given a file ID and an attribute list, the GET function uses the attribute list to determine which attributes to read and where to store the associated information. Conversely, the SET function writes the attribute information specified in the attribute list to the file header.

The PROATR routine does not require a server to execute.

Format

To get or set attributes, call PROATR with the following format:

```
CALL PROATR(status,request,att_list,file_id,lun)
```

status — The address of the eight-word Status Control Block.

request — The address of a word that contains the decimal value of the operation to be performed. The values are:

0 = Get file attributes

1 = Set file attributes

att_list — The address of the attribute list. The attribute list, terminated by a byte that contains the value 0, contains a variable number of two-word entries. Each entry is associated with an accessible file attribute as defined in Table 6-2. The maximum number of entries in the attribute list is 6.

An entry in the attribute list has the following format:

```
.BYTE   Attribute code
.BYTE   Size of attribute buffer
.WORD   Address of attribute buffer
```

The attribute code and the size of the attribute buffer (derived from Table 6-2) are in the low and high bytes, respectively, of the first word of each entry. For GET, the attribute buffer is initially empty and receives attribute information from the file header. For SET, the contents of the attribute buffer are transferred to the file header.

file_id — The address of a buffer that contains a FILES-11 File ID (FID). The file identification block is a three-word block that contains the file number, the file sequence number, and a reserved word.

```
FID:  File number
 +2:  File sequence number
 +4:  Reserved
```

The values in the file identification block can be obtained from the FID field in the NAM block used by RMS-11.

lun — The address of a word that contains the LUN number used to obtain the file ID. The LUN number can be obtained from the LCH field in the FAB block used by RMS-11.

6.4.1 Status Codes Returned by PROATR

PROATR does not use a server and, therefore, does not return any server-specific status codes. For other status codes, refer to Section 6.2.3 (Status Control Block Format).

Table 6-2: Accessible File Attributes

Code (Octal)	Attribute	Buffer Size (Octal Bytes)
1	File owner	6
2	Protection	4
3	File characteristics	2
4	Record I/O area	40
5	File name, type, version number	12
6	File type	4
7	Version number	2
11	Statistics block	12
16	Placement control	16

NOTE

The file name contained in the header is not associated with the name in a directory entry except by convention. Therefore, you cannot use the file ID to get the file name as specified in the directory; the name that the ACP returns is the name contained in the header.

6.5 PRODIR

The PRODIR routine provides two forms of directory manipulation. You can use PRODIR to:

- Create a directory on a device
- Delete a directory on a device

The name of the server used to execute PRODIR is CREDEL. This server must be installed in your system to perform any of indicated services. Otherwise, PRODIR returns a directive error in the Status Control Block (see Section 6.2.3).

Format

To create or delete a directory, invoke the PRODIR routine as follows:

```
CALL PRODIR(status,request,dir_name,dir_size
                [,[dir_owner][,dir_prot]])
```

status — The address of the eight-word Status Control Block.

request — The address of a word that contains the decimal value indicating the operation to be performed. The values are:

1 = Create directory

2 = Delete directory

dir_name — The address of a buffer that contains an ASCII device and directory specification. See the Notes for details.

dir_size — The address of a byte value that contains the number of characters in file_name.

dir_owner — The address of a word that contains the owner UIC of the directory. The default value is the protection UIC of the calling task. The low byte of the owner UIC word is the member and the high byte is the group.

Note that all parameters are positional. That is, to omit this parameter and specify a value for the next parameter, you must leave a placeholder. Specify a comma.

dir_prot — The address of a word that contains the file protection mask applied to the directory being created. The protection mask has the following form:

SYSTEM	Bits 0-3
OWNER	Bits 4-7
GROUP	Bits 8.-11.
WORLD	Bits 12.-15.

A set bit denies access, whereas a cleared bit allows access. (Note that this is the opposite of the bit polarity for the GI.PRO subfunction of the WIMP$ directive.) See the description of the protection mask in Section 8.88 later in this manual for further details.

If you omit the dir_prot parameter, the directory protection is:

SY:R,OW:RWE,GR:R,WO:R

Notes

1. In the dir_name argument, the device specification takes the form of a normal valid device specification.

   The device specification DZ1: is an example.

2. In the dir_name argument, the directory specification takes one of the following forms:

   UIC Format: [ggg,mmm] such as [301,3]

   or

   Alphabetic Format: [name] such as [WILEY]

   or

   Numeric Format: [gggmmm] such as [301003]

   where:

ggg = Group

mmm = Member

name = Any alphabetic type directory name

You cannot mix UIC format with either alphabetic or numeric format.

6.5.1 Status Codes Returned by PRODIR

PRODIR does not return any server-specific status codes. For other status codes, refer to Section 6.2.3 (Status Control Block Format).

6.5.2 PRODIR Example

The following BASIC-PLUS-2 example shows the use of PRODIR to create and delete directories.

```
10  ! Program to Create/Delete directories
    DIM Status%(7)          ! eight-word status array
    DIM Text$(2)

20  Text$(1) = "Create"
    Text$(2) = "Delete"
    PRINT "Create or Delete (C/D) :";
    LINPUT #0,Choice$       ! get choice
    Request% = 1            ! default to create
    IF LEFT$(Choice$,1) = "C" THEN GOTO 100
    ELSE IF LEFT$(Choice$,1) <> "D" THEN GOTO 20
         ELSE Request% = 2

100 PRINT "Name of Directory to "; &
                    Text$(Request%); " (ddn:[dirspec]) :";
    LINPUT #0,Dfile$        ! get directory name
    CALL Prodir BY REF (Status%(), Request%, Dfile$, LEN(Dfile$))
    FOR K = 0 TO 7          ! print status array after call
    PRINT "Status"; K, Status%(K)
    NEXT K
999 END
```

6.6 PROFBI

The PROFBI routine provides the mechanism for preparing media for use on the system. The PROFBI routine allows you to:

- Format a volume
- Check a volume for bad blocks
- Initialize a volume

Any call to the PROFBI routine destroys data on the specified disk.

Format

To format or initialize a volume, or to check it for bad blocks, invoke the PROFBI routine with the following arguments:

```
CALL PROFBI(status,request,dev_spec,dev_size[,att_list,att_size])
```

status — The address of the eight-word Status Control Block. When a volume is successfully initialized (STATUS = +1), the last six words of the Status Control Block contain the volume label expressed as an ASCII string.

request — The address of a word that contains the decimal value indicating the operation to be performed. The values are:

1 = Format a volume (hard disk only)

2 = Check a volume for bad blocks

4 = Initialize a volume

NOTE

A bad block request also formats the hard disk.

dev_spec — The address of a buffer that contains a character string which is the device specification of the volume to be formatted, initialized, or checked for bad blocks.

dev_size — The address of a word that contains the number of characters in dev_spec.

att_list — The address of the attribute list. The attribute list is a buffer of legal attributes (see Notes). Legal attributes in PROFBI are:

1 = Volume label

2 = ACS buffer (allocate checkpoint space)

att_size — The address of a word that contains the total size of the attribute list.

Notes

1. The minimum length of dev_spec is three characters--a two-character device mnemonic followed by a colon.

   If you are initializing a volume, part of the device specification can be the volume label, which can be up to 12 characters (in the form DW1:SPECTROSCOPY). You can specify the volume label in the attribute list instead. If you specify the volume label in both the dev_spec argument and the att_list argument, the dev_spec argument overrides the att_list argument.

2. If you omit the volume label when initializing a volume, PROFBI creates a default volume label using the date and time the volume was initialized. The default volume label format is:

   DDMMMYYHHMMS

3. The argument dev_spec can also be a logical name string. The logical name string must end with a colon. The number of logical name translations cannot exceed eight.

4. The att_list argument is the means of specifying optional parameters. The attribute list for PROFBI is simply a buffer of legal attributes. The high byte in the first word of the attribute list specifies the attribute type. The low byte specifies the size of the attribute list buffer in bytes.

You can use the attribute list as an alternative means of specifying a volume label. That is, you can omit the volume label in the dev_spec argument and specify it in the att_list. However, if you specify the volume label in both arguments, PROFBI overrides the att_list specification with the label specified in dev_spec.

5. The attribute list for PROFBI also contains two additional, contiguous words as the Allocate Checkpoint Space (ACS) buffer. The high byte in the first word of the ACS buffer (2) identifies it as the ACS buffer. The low byte in the buffer specifies the number of bytes in that buffer. The second word in the ACS block identifies the number of blocks in the checkpoint file.

6. Each attribute list argument must begin on a word boundary.

7. You must check a volume for bad blocks, if you have not already done so, before you can initialize it.

6.6.1 Status Codes Returned by PROFBI

The server-specific status codes returned by PROFBI are listed in Table 6-3.

- A success status code (+1) is returned in word 1 (second word) of the Status Control Block. In that case, for PROFBI, words 2 through 7 of the Status Control Block contain the volume label expressed as an ASCII string.

- A server-specific error is indicated with the value -4 being returned in word 1 of the Status Control block. In addition, the particular error code value (see Table 6-3) is returned in word 2.

- The location of status codes from other sources is specified in Section 6.2.3.

Table 6-3: PROFBI Status Codes (Server Specific)

Status Code (Decimal)	Description
+1	Success
-1	Illegal device
-2	Device not in system
-3	Failed to attach device
-4	Block zero bad--disk unusable
-5	At least one LBN (0 through 25) is bad (cannot initialize)--disk unusable
-6	Bad block file overflow
-7	Unrecoverable error
-8	Device write-locked
-9	Device not ready
-10	Failed to write bad block file
-11	Privilege violation
-12	Device is an alignment cartridge
-13	Fatal hardware error
-14	Allocation failure
-15	I/O error sizing device
-16	Allocation for sys file exceeds volume limit
-17	Homeblock allocate write error
-18	Bootblock write error--disk unusable
-19	Index file bitmap I/O error
-20	Bad block header I/O error

Status Code (Decimal)	Description
-21	MFD file header I/O error
-22	Null file header I/O error
-23	Checkpoint file header I/O error
-24	MFD write error
-25	Storage bitmap file header I/O error
-26	Failed to read bad block descriptor file
-27	Volume name too long
-28	Unrecognized disk type
-29	Preallocation insufficient to fill first index file header
-30	Preallocated too many headers for single header index file
-31	Preallocation insufficient to fill first and second index file headers
-32	Bad block limit exceeded for device
-33	Driver not resident
-34	Bitmap too large--increase cluster factor
-35	Storage bitmap I/O error
-36	Homeblock I/O error
-37	Index file header I/O error
-38	Dismount of device failed
-39	Cannot mount device foreign
-40	Cannot mount device FILES-11
-41	Cannot format DZ--no driver support
-42	Cannot detach device

Status Code (Decimal)	Description
-43	Checkpoint file header overflow--specify smaller checkpoint file
-44	Nonalphanumeric character(s) in volume name--invalid
-45	Cannot format DZ--no hardware support

6.6.2 PROFBI Example

The following BASIC-PLUS-2 example shows the use of PROFBI to check a volume for bad blocks and then initialize the volume.

```
10  ! Sample program: PROFBI bad blocking and initialization
    MAP (Sarray) INTEGER Stat(7)    ! 8 word status block
    MAP (Sarray) BYTE Volname(15)   ! BYTE mapping of Status
    PRINT "Load floppy into DZ1:. Press RESUME to continue."
    CALL Wtres BY REF()

    CALL Profbi BY REF( Stat(), 2%, "DZ1:", 4%, "", 0%)
    GOSUB 20                        ! print status returned

    CALL Profbi BY REF( Stat(), 4%, "DZ1:APPLDATA", 12%, "", 0%)
    GOSUB 20                        ! print status returned

    ! Translate and print volume label
    Volumelabel$ = ""
    FOR I% = 4% TO 15%
      Volumelabel$ = Volumelabel$ + CHR$(Volname(I%))
    NEXT I%
    PRINT Volumelabel$
    GOTO 99

20  ! Subroutine to print status returned from PROFBI
      FOR I% = 0% TO 7%
       PRINT "Status"; I%; Stat(I%)
       NEXT I%
     RETURN

99 END
```

6.7 PROLOG

The PROLOG routine provides five forms of logical name manipulation. You can use PROLOG as follows:

- Create a logical name for a device specification
- Delete a logical name for a device specification
- Translate a logical name to a device specification
- Set the default directory and/or device
- Show the default directory and device

CAUTION

Do not use this routine to create or delete logical or directory names that are used by the P/OS system, such as DW001: and BIGVOLUME:. (See the section on system logical names in the *Tool Kit User's Guide*.)

6.7.1 Create or Translate a Logical Name

Format

To create or translate a logical name, invoke the PROLOG routine with the following arguments:

```
CALL PROLOG(status,request,logname,logname_size,equiv,equiv_size)
```

status — The address of the eight-word Status Control Block.

request — The address of a word that contains the decimal value indicating the operation to be performed. The values are:

3 = Create logical

4 = Translate logical

logname	The address of a buffer that contains an ASCII string (which can contain alphanumeric characters only, 1 byte/character). Refer to the *Tool Kit User's Guide* for P/OS conventions regarding logical names, device names, and reserved names.
logname_size	The address of a byte value that contains the number of characters in logname.
equiv	The address of a buffer that contains an equivalence string. It can be any string up to 255 characters in length.
equiv_size	For CREATE: The address of a byte value that contains the number of characters in equiv. For TRANSLATE: The address of a byte value that contains the maximum number of characters in the equiv buffer.

For the TRANSLATE function, the equiv argument is an output argument returned by PROLOG. The length of the string returned in the equiv buffer is returned in word 2 of the status block.

Note

PROLOG does the following by default:

- It changes any lowercase logical name value you specify to uppercase. This is because RMS and the F11ACP expect logical names to be uppercase.
- It specifies a default modifier value of 0. See Section 2.2 for details.
- For a translate operation, PROLOG specifies a default inhibit mask of 0. That is, the system searches all tables for the equivalence value.
- For a create or delete operation, PROLOG specifies that the operation be performed on the LT.USR table only.

6.7.2 Delete a Logical Name and Set/Show Default

Format

To delete a logical name or to set or show the default device and/or directory, invoke the PROLOG routine with the following arguments:

```
CALL PROLOG(status,request,logname,logname_size)
```

status	The address of the eight-word Status Control Block.
request	The address of a word that contains the decimal value indicating the operation to be performed. The values are: 1 = Set default 2 = Show default 5 = Delete logical
logname	The address of a buffer that contains an ASCII string (1 byte/character) which can contain alphanumeric characters only. The user must have already created the logname.
logname_size	For SET and DELETE, the address of a byte value that contains the number of characters in logname. For SHOW, the address of a byte value that contains the maximum number of characters in the logname buffer.

Notes

1. When used to set the default device or directory, PROLOG does not check whether or not the device or directory does in fact exist. No error or status code is returned if the device or directory does not exist.

2. For the SET DEFAULT function, the logname string can contain a directory specification of the form:

   ```
   USERDISK:[DIRECTORY]
   ```

   This is a logical name with a directory specification appended to it. A directory specification has one of the following forms:

UIC Format:	[ggg,mmm]	such as [301,3]
or		
Alphabetic Format:	[name]	such as [WILEY]
or		
Numeric Format:	[gggmmm]	such as [301003]

where:

ggg = Group

mmm = Member

name = Any alphabetic type directory name

You cannot mix UIC format with either the alphabetic or numeric format.

3. When issuing a call for the SET DEFAULT function, you can specify a device, a directory, or both. If you specify both, then both the default device and directory are changed. If you specify only one, the other does not change.

CAUTION

> The default directory and device belong to the user, not to your application. We strongly suggest that you prompt the user for the new default values, and that you change the values back to their original state before exiting.

4. For both the DELETE and SET DEFAULT functions, there is no output argument; PROLOG returns the call status in the Status Control Block. For SET DEFAULT, PROLOG does not check whether the device or directory does in fact exist. No error status code is returned if the device or directory does not exist. PROLOG uses the SDIR$ directive, which also does not check whether the directory exists.

5. For the SHOW DEFAULT function, logname is an output argument that contains the default directory string. The length of the string returned in logname is returned in word 2 of the status block.

6.7.3 Status Codes Returned by PROLOG

Most error returns from PROLOG are Directive Status errors (see CLOG$, DLOG$, and TLOG$ logical name directives in Chapter 8. The numerical equivalents of the status codes are in the appendixes).

The server-specific status codes returned by PROLOG are listed in Table 6-4.

- A success status code (+1) is returned in word 1 (second word) of the Status Control Block.
- A server-specific error is indicated with the value -4 being returned in word 1 of the Status Control block. In addition, the particular error code value (see Table 6-4) is returned in word 2.
- The location of status codes from other sources is as specified in Section 6.2.3.

Table 6-4: PROLOG Status Codes (Server Specific)

Status Code	Description
+1	Success
-1	Error in parsing the set default string. Either a badly formed specification was passed or something other than a device or directory was found in string.
-2	Cannot determine type of service requested.

6.7.4 PROLOG Examples

The following BASIC-PLUS-2 example shows the use of PROLOG to create and then translate a logical name.

```
10  !   Sample program to create and then translate a logical
```

```
    DIM Stat%(7%)                   ! set up status array
    Req% = 3%                       ! request is CREATE
    Logname$ = 'DISKETTE1'          ! logical name to create
    Logsize% = LEN(Logname$)        ! logical name length
    Eqv$     = 'DZ1:'               ! equivalence to create
    Eqvsize% = LEN(Eqv$)            ! equivalence length

    CALL Prolog BY REF(Stat%(),Req%,Logname$,Logsize%,Eqv$,Eqvsize%)
    GOSUB 20                        ! print returned status

    PRINT 'New logical name  = '; Logname$
    Req% = 4%                       ! set request to translate
    Eqv$ = SPACE$(40%)              ! clear buffer to receive equivalence

    CALL Prolog BY REF(Stat%(),Req%,Logname$,Logsize%,Eqv$,LEN(Eqv$))
    GOSUB 20                        ! print returned status
    PRINT 'Equivalence name  = '; Eqv$

    GOTO 99

20 ! Subroutine to print status returned from PROLOG calls
    FOR I% = 0% TO 7%
      PRINT 'status'; I%; ' '; Stat%(I%)
    NEXT I%
    RETURN

99 END
```

The next BASIC-PLUS-2 example shows the use of PROLOG to delete a logical, show the default directory, and set a current directory.

```
10  ! Sample program to delete a logical, show default directory, set
    ! directory, and set the current directory.

    DIM Stat%(7%)                 ! set up status array
    Logname$ = 'DISKETTE1'        ! delete an existing logical
    Req% = 5%                     ! request = delete

    CALL Prolog BY REF(Stat%(), Req%, Logname$, LEN(Logname$))
    GOSUB 20                      ! print returned status

    PRINT 'Logical deleted = ';Logname$
    Req% = 2%                     ! request = show default
    Showdir$ = SPACE$(40%)        ! buffer for default directory
    Length% = LEN(Showdir$)       ! length of buffer

    CALL Prolog BY REF(Stat%(), Req%, Showdir$, Length%)
    GOSUB 20                      ! print returned status

    PRINT 'Current default directory = ';TRM$(Showdir$)
```

```
    Req% = 1%                       ! request = set default
    Dirname$ = 'USERDISK:[USERFILES]' ! new default directory

    CALL Prolog BY REF(Stat%(), Req%, Dirname$, LEN(Dirname$))
    GOSUB 20                        ! print status returned

    PRINT 'New default directory = ';Dirname$
    GOTO 99

20 ! Subroutine to print status returned from PROLOG
    FOR I% = 0% TO 7%
      PRINT 'Status'; I%; ' '; Stat%(I%)
    NEXT I%
    RETURN

99 END
```

6.8 PROTSK

The PROTSK routine provides four forms of task manipulation. You can use PROTSK to:

- Install a task or static region
- Remove a task or static region
- Fix an installed task or region in memory
- Install, run, and remove an offspring task through a parent task

In addition, you can specify that an installed task not be aborted or removed if the application exits or if the user presses INTERRUPT/DO at the terminal. This feature, called NOREMOVE, can be used, for example, to ensure that a noninteractive background task, such as a file transfer, is not aborted inadvertently.

CAUTION

Use care with the NOREMOVE option. If the name of the task is the same as that used by another application which is to be run subsequently, the second task by that name will not be installed unless the first is removed.

Also, unless you specify the install/run/remove option, the only means of removing a task installed with the NOREMOVE option is either by powering down the system or by running an application that, knowing the task's name, can remove it. For example, you can remove the task with the DCL REMOVE command.

See the descriptions of the REQUEST argument for Install a Task and for Install/Run/Remove a Task.

The name of the server used to execute PROTSK is INSREM. This server must be installed in your system to perform any of indicated services. Otherwise, PROTSK returns a directive error in the Status Control Block. (See Section 6.2.3.)

6.8.1 Install a Task or Static Region

Format

To install a task or static region, call the routine PROTSK with all the following arguments:

```
CALL PROTSK(status,request,task_name,file_name,file_size)
```

status — The address of the eight-word Status Control Block.

request — The address of a word that contains the decimal value indicating the type of task manipulation to be performed:

- 1 = Install a task or region.
- 4 = Fix a task or region.
- 8 = Region is read only.
- 32. = Install task or region with name supplied in the task_name argument.
- 64. = NOREMOVE. The task will not be aborted or removed if the application exits or if the user presses INTERRUPT/DO. In addition the task will not be removed upon exit. (see CAUTION at beginning of PROTSK section).
- 128. = WRITEBACK. This option forces the system to checkpoint a read-write common back to its own task image rather than to the system checkpoint file. By default, if you do not specify WRITEBACK, the system checkpoints a read-write common only to the system checkpoint file. (Applies only to P/OS V3.0 systems or later.)

  The WRITEBACK option additionally forces the read-write common to be written back when removed.

If you require more than one task manipulation function, add the decimal values of each function together to produce a single decimal value.

For example, to install a task and fix it in memory with your Radix-50 task name, specify the request value as 37. Obtain the request value by adding the values for install (1), fix (4), and name supplied in the task_name argument (32).

task_name — The address of a two-word Radix-50 task name. Upon completion, PROTSK returns the two-word Radix-50 installed task name at this location.

If you selected value 32 as the request option (install task with name supplied in task_name), then supply the Radix-50 task name in the two words at the address before calling PROTSK.

file_name — The address of the buffer that contains an ASCII file specification of the task image or static region to be installed.

file_size — The address of a value describing the number of characters in file_name.

6.8.2 Remove a Task or Static Region

Format

To remove a task or static region, call the routine PROTSK with all of the following arguments:

```
CALL PROTSK(status,request,task_name)
```

status — The address of the eight-word Status Control Block.

request — The address of a word that contains the decimal value indicating the operation to be performed:

2 = Remove

16. = Entity to be removed is a static region

If you are removing a static region, specify the sum of the two options (18) as the value of the request.

task_name — The address of a two-word Radix-50 task name that identifies the task or static region to be removed.

6.8.3 Fix a Task or Region in Memory

Format

To fix an installed task or region in memory, call the routine PROTSK with the following arguments:

```
CALL PROTSK(status,request,task_name)
```

status — The address of the eight-word Status Control Block.

request — The address of a word that contains the decimal value indicating the type of task manipulation to be performed:

4 = Fix a task or region

16 = Entity to be fixed is a region

If you are fixing a region, specify the sum of the two options (20.) as the value of the request.

task_name — The address of a two-word Radix-50 task name that identifies the task or region to be fixed in memory.

6.8.4 Install/Run/Remove an Offspring Task

By requesting install/run/remove, a parent task can install an offspring task, have it run immediately, and have it removed upon exit. Install/run/remove can be executed in two ways, which can be compared to either *calling* the offspring task or *chaining* to the offspring task. The distinction between CALL and CHAIN install/run/remove is as follows:

- With CALL the parent task initiates execution of the offspring task and still continues its own execution. While the offspring is being installed and started, the parent is stopped. After the install/run/remove request has been completed, the parent can continue its own execution.

- With CHAIN the parent task initiates execution of the offspring task, passes offspring information, and then exits. If the offspring has been installed and initiated successfully, the parent exits. Otherwise, an error condition is returned to the parent, and the parent does not exit. The parent must perform any necessary cleanups, such as closing files, before chaining to the offspring.

Format

To use install/run/remove, call the routine PROTSK with the following arguments:

```
CALL PROTSK(status,request,task_name,file_name,file_size
     [,cmd_line,cmd_size] [,event_flag]
     [,exit_status])
```

The arguments are defined below. Use all of the arguments that are not enclosed in brackets. The arguments enclosed in brackets are optional, with the following provisions:

- If none of the optional arguments are used, they can all be omitted.

- Use the arguments event_flag and exit_status only with the CALL install/run/remove option.

- The optional arguments are positional. If you use event_flag and/or exit_status but do not use cmd_line, then the word in cmd_size must have the value 0.

The arguments are defined as follows:

status — The address of the eight-word Status Control Block.

request — The address of a word that contains the decimal value indicating the type of task manipulation to be performed:

3 = Install/run/remove a task

32. = Install task with name supplied in the task_name argument.

64. = NOREMOVE. The task will not be aborted or removed if the application exits or the user presses INTERRUPT/DO. However, the task will be removed upon exit. (See CAUTION at beginning of PROTSK section.)

128. = To select the CALL install/run/remove option, add 128 to the decimal value in the request word. To select CHAIN, disregard this value. (CHAIN is the default option.)

If you require more than one task manipulation function, add the decimal values of each function together to produce a single decimal value.

For example, to select install/run/remove with the CALL option and your Radix-50 task name, specify the request value as 163. Obtain the request value by adding the values for install/run/remove (3), name supplied in the task_name argument (32), and the CALL option (128).

When specifying install/run/remove, it is illegal to also specify "Fix a task in memory" or "Install a common or library".

task_name — The address of a two-word Radix-50 (offspring) task name. Upon completion, PROTSK returns the two-word Radix-50 installed task name at this location.

If you selected value 32 as the request option (install task with name supplied in task_name), then supply the Radix-50 task name in the two words at the address before calling PROTSK.

file_name — The address of the buffer that contains an ASCII file specification of the (offspring) task image to be installed.

file_size — The address of value describing the number of characters in file_name.

cmd_line — (Optional.) The address of a buffer that contains a command line to be queued to the offspring task.

cmd_size — The address of a value describing the number of characters in cmd_line. The maximum number of characters is 255 (decimal). If cmd_line is not specified, then the word in cmd_size must have the value 0.

event_flag — (For CALL option only.) The event flag to be cleared on issuance and set when the offspring task exits or emits status.

exit_status — (For CALL option only.) The address of an eight-word status block to be written when the offspring task exits or emits status.

Word 0 = Offspring task exit status

Word 1 = System abort code

Word 2-7 = Reserved

NOTE

The exit status block defaults to one word. To use the eight-word exit status block, you must specify the logical OR of the symbol SP.WX8 and the event flag number in the EVENT_FLAG parameter above.

6.8.5 Status Codes Returned by PROTSK

The server-specific status codes returned by PROTSK are listed in Table 6-5.

- A success status code (+1) is returned in word 1 (second word) of the Status Control Block.
- A server-specific error is indicated with the value -4 being returned in word 1 of the Status Control block. In addition, the particular error code value (see Table 6-5) is returned in word 2.
- The location of status codes from other sources is as specified in Section 6.2.3.

Table 6-5: PROTSK Status Codes (Server Specific)

Status Code	Description
+ 1	Success.
- 1	Task name in use.
- 2	File not found.
- 3	Specified partition too small.
- 4	Task and partition base mismatch.
- 7	Common block length mismatch.
- 8	Common block base mismatch.
- 9	Too many common block requests.
-11	Checkpoint area too small.
-13	Not enough APRs for task image.
-14	File not a task image.
-15	Base address must be on 4k boundary.
-16	Illegal first APR.
-18	Common block parameter mismatch.

Status Code	Description
-20	Common block not loaded.
-22	Task image virtual address overlaps common block.
-23	Task image already installed.
-24	Address extensions not supported.
-29	Illegal UIC.
-30	No pool space.
-31	Illegal use of partition or region.
-32	Access to common block denied.
-33	Task image I/O error.
-34	Too many LUNs.
-35	Illegal device.
-36	Task may not be run.
-37	Task active.
-39	Task fixed.
-40	Task being fixed.
-41	Region attached or referenced by an installed task.
-43	Common/task not in system.
-44	Region or common fixed.
-45	Cannot do receive from requestor.
-46	Cannot attach to requestor.
-47	Invalid request.
-48	Cannot return status.
-49	Error encountered on file open operation.
-50	Error encountered on file close operation.

Status Code	Description
-51	Cannot get file LBN to process label blocks.
-53	Unable to create or map to region (indicates resource allocation problem).
-54	Versions of P/OS later than 2.0A do not support resident headers. All tasks are installed with external headers. You must not build your task using the /-XH task builder qualifier for versions of P/OS after 2.0A.
-55	Cannot install a common that has been built with the EXTTSK task builder option. IMPORTANT: A common using the EXTTSK option is likely to corrupt the disk from which it is installed.
-56	Cannot fix a prototype task.

6.8.6 PROTSK Example

The following BASIC-PLUS-2 example shows the use of PROTSK to install a task.

```
10  ! Sample program to install a task using PROTSK.

    DIM status%(7),taskn%(1)                !Status block
    type% = 1                               !Request = install
    tfile$="dz1:[mytasks]payroll1.tsk"      !file to install

    CALL protsk BY REF (status%(),  &
                        type%,      &
                        taskn%(),   &
                        tfile$,     &
                        LEN(tfile$))

    GOTO 99 IF status%(1) = 1           !Proper install?
    PRINT "Error--task did not install properly" \ STOP

99  END
```

6.9 PROVOL

The PROVOL routine provides several volume-related functions. You can use PROVOL to:

- Mount or dismount a volume
- Plug the bootblock, or plug the bootblock and bootstrap a volume
- Unplug the bootblock
- Bootstrap a volume
- Get the free space
- Get the free space and file header usage
- Establish a secondary boot device

6.9.1 Mount/Dismount

Format

To mount or dismount a volume, invoke the PROVOL routine with the following arguments:

```
CALL PROVOL(status,request,dev_spec,dev_size,[att_list,att_size])
```

status — The address of the eight-word Status Control Block. When mounting a nonforeign volume, the last six words of the Status Control Block contain the volume label expressed as an ASCII string, provided that the operation is successful (status = +1).

request — The address of a word that contains the decimal value indicating the operation to be performed. The values are:

0 = Mount a volume

1 = Mount a foreign volume

2 = Dismount a volume

dev_spec — The address of a buffer that contains a character string that is the device specification of the volume to be mounted or dismounted. All letters in the string must be uppercase.

Part of the device specification can be the volume label, which can be up to 12 characters. If you omit the volume label from dev_spec, PROVOL by default gets the label from the specified disk. If you specify a volume label in a dev_spec argument, the specified label must match the label on the volume; otherwise the operation fails.

dev_size — The address of a word that contains the number of characters in dev_spec.

att_list — The address of the attribute list. The att_list is the means of specifying optional parameters. The attribute list for mount/dismount is simply a buffer of valid attributes. The high byte in the first word of the attribute list specifies the attribute type. The low byte specifies the size of the buffer in bytes.

You can use the attribute list as an alternative means of specifying a volume label. That is, you can omit the volume label in the dev_spec argument and supply it in the att_list argument. However, if you specify the volume label in both arguments, PROVOL overrides the att_list specification with the label specified in dev_spec.

The valid attribute for mount/dismount is:

1 = Volume label

att_size — The address of a word that contains the size of the attribute list.

6.9.2 Bootstrap a Volume

Format

To bootstrap a volume, call PROVOL with the following arguments:

```
CALL PROVOL(status,request,dev_spec,dev_size)
```

status	The address of the eight-word Status Control Block.
request	The address of a word that contains the decimal value indicating the operation to be performed. The value is: 8 = Bootstrap a Volume
dev_spec	The address of a buffer that contains a character string that is the device specification.
dev_size	The address of a word that contains the number of characters in dev_spec.

6.9.3 Plug Bootblock/Plug Bootblock and Boot

Format

To plug the bootblock, or plug the bootblock and boot the volume, call PROVOL with the following arguments:

```
CALL PROVOL(status,request,dev_spec,dev_size)
```

status	The address of the eight-word Status Control Block.
request	The address of a word that contains the decimal value indicating the operation to be performed. The values are: 9 = Plug a bootblock on a volume 10. = Plug a bootblock on a volume and bootstrap the volume
dev_spec	The address of a buffer that contains a character string that is the device, directory and filename of the system image to

be used to plug the bootblock. You must supply the device; the directory and filename are optional. The default file specification is [ZZSYS]POS.SYS.

dev_size — The address of a word that contains the number of characters in dev_spec.

6.9.4 Unplug a Bootblock

Format

To unplug the bootblock on a volume, call PROVOL with the following arguments:

```
CALL PROVOL(status,request,dev_spec,dev_size)
```

status — The address of the eight-word Status Control Block.

request — The address of a word that contains the decimal value indicating the operation to be performed. The value is:

13. = Unplug a bootblock from a volume

dev_spec — The address of a buffer that contains character string that is the device specification.

dev_size — The address of a word that contains the number of characters in dev_spec.

6.9.5 Get Free Space

Format

To determine the free space available on a volume, call PROVOL with the following arguments:

```
CALL PROVOL(status,request,dev_spec,dev_size)
```

status — The address of the eight-word Status Control Block.

request — The address of a word that contains the decimal value indicating the operation to be performed. The value is:

14. = Get free space on a volume

dev_spec — The address of a buffer that contains a character string that is the device specification.

dev_size — The address of a word that contains the number of characters in dev_spec.

When you request free space, PROVOL returns the number of free blocks in words 2 and 3 of the Status Control Block.

As shown in Table 6-6, the number of free blocks is contained in three bytes. The low byte of word 2 of the Status Control Block contains the high-order word of the value, and word 3 contains the low-order word of the value.

Table 6-6: Get Free Space Status Block

Status Word	Contents
2	Total free disk blocks, high-order byte
3	Total free disk blocks, low-order word

6.9.6 Get Free Space and File Header Usage

Format

To determine the free space and file header usage on a volume, call PROVOL with the following arguments:

```
CALL PROVOL(status,request,dev_spec,dev_size)
```

status — The address of the eight-word Status Control Block.

request — The address of a word that contains the decimal value indicating the operation to be performed. The value is:

15. = Get free space and file header usage

dev_spec — The address of a buffer that contains a character string that is device specification.

dev_size — The address of a word that contains the number of characters in dev_spec.

When you request free space and file header usage, PROVOL returns the number of free blocks, the largest number of contiguous free blocks, the maximum number of file headers, and the total free file headers in words 2 through 7 of the Status Control Block. Table 6-7 illustrates the Status Control Block.

Table 6-7: Get Free Space and File Headers Status Block

Status Word	Contents
2	Total free disk blocks, high-order byte
3	Total free disk blocks, low-order word
4	Total contiguous blocks, high-order byte
5	Total contiguous blocks, low-order word
6	Maximum number of file headers
7	Total free file headers

The number of free blocks and the largest number of contiguous blocks are three bytes each, the low byte of the first word being the high-order byte of the value, the second word being the low-order word of the value.

Note that this call requires many disk accesses and takes a long time to complete. If you only want to get the number of free blocks, we recommend that you use the get free space PROVOL call described in the previous section.

6.9.7 Establish Secondary Boot Device

Format

The system always attempts to boot from the primary boot device--the floppy drives--before searching for a secondary boot device, normally the hard disk. To establish an alternative secondary boot device, call PROVOL with the following arguments:

```
CALL PROVOL(status,request,device_id,unit_num,slot_num)
```

status	The address of the eight-word Status Control Block.
request	The address of a word that contains the decimal value indicating the operation to be performed. The value for establishing a secondary boot device is: 16. = Establish a secondary boot device
device_id	The address of a buffer that contains a one-word integer value representing the device identification number of the device you are establishing as the secondary boot device. Table C-2 in Appendix C contains a list of device ID numbers.
unit_num	The address of a one-word integer value representing the physical unit number of the device you are establishing as the secondary boot device. Although the system ignores this value on a Professional 350 system (it is significant on a Professional 380), you must always supply it.
slot_num	The address of a 1-byte integer representing the physical slot number of the secondary boot device. The slots are numbered from 0 through 7. Although the system ignores this value on a Professional 350 system, (it is significant on a Professional 380), you must always supply it.

6.9.8 Note For PROVOL

The argument dev_spec can be a logical name string. In this case, the logical name string must end with a colon. The number of logical name translations cannot exceed eight.

6.9.9 Status Codes Returned by PROVOL

The server-specific status codes returned by PROVOL are listed in Table 6-8.

- A success status code (+1) is returned in word 1 (the second word) of the Status Control Block. In that case, for mount/dismount operations only, words 2 through 7 of the Status Control Block contain the volume label expressed as an ASCII string.

- A server-specific error is indicated by the value -4 returned in word 1 of the Status Control Block. In addition, the particular error code value (see Table 6-8) is returned in word 2.

- A QIO error is indicated by the value -2 returned in word 1 of the Status Control Block. As usual, the actual QIO error is returned in word 2 of the Status Control Block. However, if the QIO error is IE.ABO, then word 3 of the Status Control Block contains one of the subcodes listed in Table 6-9. These subcodes provide further information on the mount or dismount failure.

- The location of status codes from other sources is as specified in Section 6.2.3.

Table 6-8: PROVOL Status Codes (Server Specific)

Status Code	Description
+1	Success
-1	File is not a system image
-2	Invalid boot device

Table 6-9: IE.ABO Subcodes for PROVOL Mount/Dismount Failure

Decimal	Octal	Description
+07.	000007	Syntax error
+10.	000012	Home block not found
+11.	000013	Wrong volume
+12.	000014	Checkpoint file still active
+14.	000016	Volume already marked for dismount
+15.	000017	Volume not mounted
+16.	000020	Volume already mounted FILES-11
+17.	000021	Volume already mounted foreign

6.9.10 PROVOL Example

The following BASIC-PLUS-2 example shows the use of PROVOL to mount and dismount a volume.

```
10 ! Sample program to test PROVOL requests.
     MAP (Sarray) INTEGER Stat(7)
     MAP (Sarray) BYTE Volname(15)

     Device$ = 'DZ1:'                ! device specification
     Devlen% = LEN(Device$)          ! specification size
     PRINT 'Insert floppy into DZ1:.'
     PRINT 'Press RESUME to DISMOUNT.'
     CALL Wtres BY REF()             ! POSRES Wait for Resume

     Req% = 2%                       ! dismount request
                                     ! must first dismount
                                     ! because closing door
                                     ! causes automatic mount

     CALL Provol BY REF(Stat(), Req%, Device$, Devlen%, '',0%)

     GOSUB 20                        ! print returned status
```

```
      Req% = 0%                       ! mount request
      PRINT 'Press RESUME to MOUNT.'
      CALL Wtres BY REF()             ! POSRES Wait for Resume

      CALL Provol BY REF(Stat(), Req%, Device$, Devlen%, '',0%)

      GOSUB 20                        ! print returned status
      Volumelabel$ = ""
      FOR I% = 4% TO 15%              ! translate ASCII volume label
        Volumelabel$ = Volumelabel$ + CHR$(Volname(I%))
      NEXT I%
      PRINT 'Label of mounted volume: ',Volumelabel$

      Req% = 9%                       ! plug bootblock request
      Device$ = "DZ1:[ZZSYS]BOOT.SYS"
                                      ! device specification
                                      ! this file must already exist.
      Devlen% = LEN(Device$)          ! size of device specification
      PRINT 'Press RESUME to PLUG BOOTBLOCK.'
      CALL Wtres BY REF()             ! POSRES Wait for Resume

      CALL Provol BY REF(Stat(), Req%, Device$, Devlen%, '', 0%)

      GOSUB 20                        ! print returned status
      IF Stat(1) <> 1%
                    THEN GOTO 99          ! plug bootblock unsuccessful
      END IF
      Req% = 8%                       ! bootstrap device request
      Device$ = "DZ1:"                ! device specification
      Devlen% = LEN(Device$)          ! device specification size
      PRINT 'Press RESUME to BOOTSTRAP.'
      CALL Wtres BY REF()             ! POSRES Wait for Resume

      CALL Provol BY REF(Stat(), Req%, Device$, Devlen%, '', 0%)

      GOSUB 20                        ! print returned status
      GOTO 99

20 ! Subroutine for printing status returned from PROVOL calls
      FOR I% = 0% TO 7%
        PRINT 'Status'; I%; ' '; Stat(I%)
      NEXT I%
      RETURN
99 END
```

CHAPTER 7

USING THE SYSTEM DIRECTIVES

When your task requires the Executive to perform an operation, it can issue a system directive to make the request. Directives provide control over task execution and interaction.

If you are a MACRO-11 programmer, you usually issue directives in the form of macros defined in the system macro library, LB:[1,5]RSXMAC.SML. You should use the directive macros rather than hand-coding calls to directives.

If you are a high-level language programmer and your language supports the PDP-11 R5 calling convention, you can call system-supplied subroutines to make directive requests. These subroutines reside in the system object module library, LB:[1,5]SYSLIB.OLB.

System directives enable tasks to:

- Obtain task and system information
- Measure time intervals
- Perform I/O functions
- Spawn other tasks
- Communicate and synchronize with other tasks
- Manipulate a task's logical and virtual address space
- Suspend and resume execution
- Exit
- Manipulate logical names

Directives are implemented by the Emulator Trap (EMT) 377 processor instruction. EMT 0 through EMT 376 are considered to be non-P/OS EMT synchronous system traps. They cause the Executive to abort the task unless the task has specified that it wants to receive control when such traps occur.

See the *PDP-11 Architecture Handbook* for a description of the EMT instruction.

Sections 7.1 and 7.2 are intended for all users. Section 7.3 specifically describes the use of macros, while Section 7.4 describes the use of high-level language subroutine calls.

7.1 HOW THE SYSTEM PROCESSES DIRECTIVES

P/OS processes a system directive in four steps:

1. A user task issues a directive with parameters that the Executive uses. The directive code and parameters that the task supplies to the system are known as the Directive Parameter Block (DPB). The DPB can be either on the user task's stack or in a user task's data section.

2. The Executive receives an EMT 377 generated by the directive macro (or a DIR$ macro), or high-level language subroutine.

3. The Executive processes the directive.

4. The Executive returns directive status information to the task's Directive Status Word (DSW).

Note that the Executive preserves all task registers when a task issues a directive.

The user task issues an EMT 377 (generated by the directive) together with the address of a DPB or a DPB itself, on the top of the user task's stack. When the stack contains a DPB address, the Executive removes the address after processing the directive, and the DPB itself remains unchanged. When the stack contains the actual DPB rather than a DPB address, the Executive removes the DPB from the stack after processing the directive.

The first word of each DPB contains a Directive Identification Code (DIC) byte and a DPB size byte. The DIC indicates which directive is to be performed; the size byte indicates the DPB length in words. The DIC is in the low-order byte of the word, and the size is in the high-order byte.

The DIC is always an odd-numbered value. This allows the

Executive to determine whether the word on the top of the stack (before EMT 377 was issued) was the address of the DPB (even-numbered value) or the first word of the DPB (odd-numbered value).

The Executive normally returns control to the instruction following the EMT. Exceptions to this are directives that result in an exit from the task that issued them and an Asynchronous System Trap (AST) exit, if at AST state.

The Executive also clears or sets the Carry bit in the Processor Status Word (PSW) to indicate acceptance or rejection, respectively, of the directive. The DSW, addressed symbolically as $DSW, is set to indicate a more specific cause for acceptance or rejection of the directive.*

The DSW usually has a value of +1 for acceptance, and a range of negative values for rejection. Exceptions to this rule are as follows:

- Directives such as CLEF$, SETF$, GMCR$, GPRT$, and any of the receive data directives, have multiple success return codes.
- The Instrument Society of America (ISA) subroutines START and WAIT, provided for FORTRAN programs, have positive numeric error codes, as required by ISA. See Sections 8.42 and 8.58 for details.

The Executive associates DSW values with symbols, using mnemonics that report either successful completion or the cause of an error (see Section 7.2). In the case of successful Exit directives, the Executive does not return control to the task. If an Exit directive fails, however, control returns to the task with an error status in the DSW.

On Exit, the Executive frees task resources as follows:

- Detaches all attached devices
- Flushes the AST queue and despecifies all specified ASTs
- Flushes the receive and receive-by-reference queues
- Flushes the clock queue for outstanding Mark Time requests for the task

---

* The task builder resolves the address of $DSW.

- Closes all open files (files open for write access are locked)
- Detaches all attached regions except in the case of a fixed task, where no detaching occurs
- Runs down the task's I/O
- Disconnects from interrupts
- Breaks the connection with any offspring tasks
- Frees the task's memory if the task was not fixed

If the Executive rejects a directive, it usually does not clear or set any specified event flag. Thus, the task can wait indefinitely if it executes a Wait For directive corresponding to a previously issued Mark Time directive that the Executive has rejected. You should always ensure that a directive has been completed successfully.

7.2 ERROR RETURNS

As stated earlier, the Executive associates the error codes with mnemonics that report the cause of the error. In the text of this manual, the mnemonics are used exclusively.

High-level language programmers do not have access to the mnemonics, and must refer to Appendix B in this manual for interpretations of the DSW error codes.

Each directive description in Chapter 8 contains specific, directive-related interpretation of the error codes.

7.3 USING DIRECTIVE MACROS

If you are programming in MACRO-11, you must decide how to create the DPB before you issue a directive. The DPB can either be created on the stack at run time (see Section 7.3.1.3, which describes the $S form of directive) or created in a data section at assembly time (see Sections 7.3.1.1 and 7.3.1.2, which describe the $ form and $C form, respectively). If the code must be reentrant and the parameters vary, you must create the DPB on the stack.

Figures 7-1 and 7-2 illustrate the alternative directives and also show the relationship between the stack pointer and the DPB.

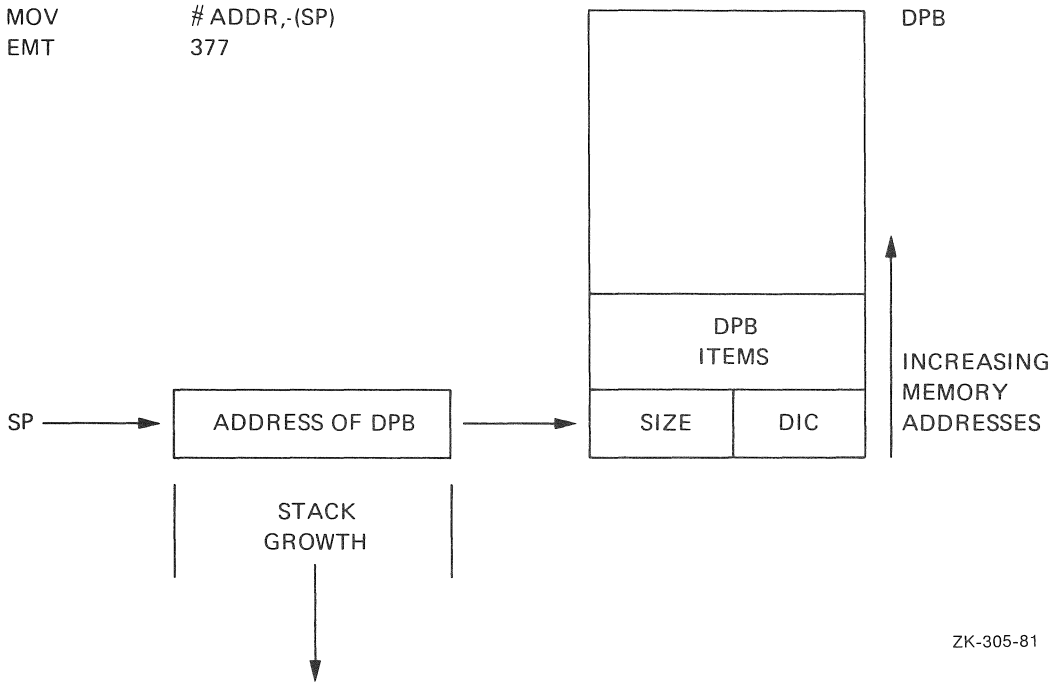


Figure 7-1: Directive Parameter Block (DPB) Pointer on the Stack

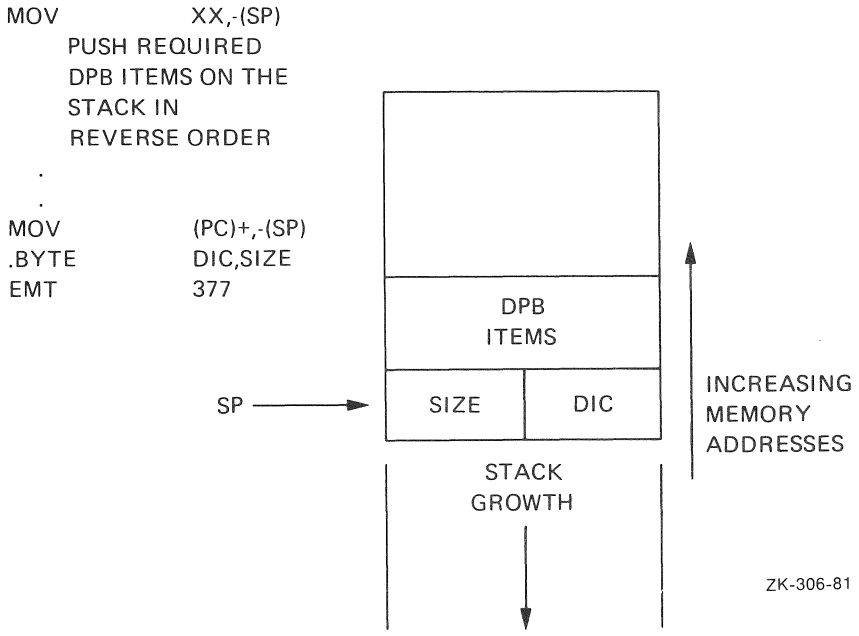


Figure 7-2: Directive Parameter Block (DPB) on the Stack

7.3.1 Macro Name Conventions

When you are programming in MACRO-11, you use system directives by including directive macro calls in your programs. The .MCALL assembler directive makes these macros available to a program. The .MCALL arguments are the names of all the macros used in the program. For example:

```
;
; CALLING DIRECTIVES FROM THE SYSTEM MACRO LIBRARY
; AND ISSUING THEM.
;

   .MCALL  MRKT$S,WTSE$S
      .
      .
      .
   Additional .MCALLs or code
      .
      .
      .
   MRKT$S  #1,#1,#2,,ERR    ;MARK TIME FOR 1 SECOND
   WTSE$S  #1               ;WAIT FOR MARK TIME TO COMPLETE
      .
      .
      .
```

Macro names consist of as many as four letters, followed by a dollar sign ($) and, optionally, a C or an S. The optional letter, or its absence, specifies which of three possible macro expansions the programmer wants to use.

7.3.1.1 $ Form - The $ form is useful for a directive operation that is issued several times from different locations in a non-reentrant program segment. The $ form is most useful when the directive is issued several times with varying parameters (one or more but not all parameters change), or in a reentrant program section when a directive is issued several times even though the DPB is not modified.

The $ form produces only the directive's DPB and must be issued from a data section of the program. The code for actually executing a directive that is in the $ form is produced by a special macro, DIR$ (described in Section 7.3.2.

The following consequences result from the fact that execution of a directive is separate from the creation of the directive's DPB:

1. A $ form of a given directive needs to be used only once (to produce its DPB).

2. A DIR$ macro associated with a given directive can be issued several times without incurring the cost of generating a DPB each time it is issued.

3. The directive's parameters can be easily accessed and changed by labeling the start of the DPB and using the offsets defined by the directive.

When a program issues the $ form of macro call, the parameters required for DPB construction must be valid expressions for MACRO-11 data storage instructions (such as .BYTE, .WORD, and .RAD50). You can alter individual parameters in the DPB. You might do this if you want to use the directive many times with varying parameters.

7.3.1.2 $C Form - Use the $C form when a directive is to be issued only once. The $C form eliminates the need to push the DPB (created at assembly time) onto the stack at run time. Other parts of the program, however, cannot access the DPB because the DPB address is unknown.

Note, in the $C form of the macro expansion, that the new value of the assembler's location counter redefines the DPB address $$$ each time an additional $C directive is issued.

The $C form generates a DPB in a separate p-section called DPB.*

The DPB is followed first by a return to the user-specified p-section, then by an instruction to push the DPB address onto the stack, and finally by an EMT 377. To ensure that the program reenters the correct p-section, you must specify the p-section name in the argument list immediately following the DPB parameters. If the argument is not specified, the program reenters the blank (unnamed) p-section.

The $C form also accepts an optional final argument that specifies the address of a routine to be called (by a JSR instruction) if an error occurs during the execution of the directive (see Section 7.3.2).

---

* See the *PDP-11 MACRO-11 Language Reference Manual* for a description of p-sections (program sections).

When a program issues the $C form of a macro call, the parameters required for DPB construction must be valid expressions for MACRO-11 data storage instructions (such as .BYTE, .WORD, and .RAD50). (This is not true for the p-section argument and the error routine argument, which are not part of the DPB.)

7.3.1.3 $S Form - Program segments that need to be reentrant and use DPBs with dynamic parameters should use the $S form. Only the $S form produces the DPB at run time. The other two forms produce the DPB at assembly time.

In this form, the macro produces code to push a DPB onto the stack, followed by an EMT 377. In this case, the parameters must be valid source operands for MOV-type instructions. For a two-word Radix-50 name parameter, the argument must be the address of a two-word block of memory that contains the name.

Note that if your task refers to stack locations using an offset from the stack pointer, you must take into account the fact that code generated by the $S form pushes DPB parameters onto the stack.

The $S form of a macro processes arguments from right to left.

7.3.2 DIR$ Macro

The DIR$ macro allows you to execute a directive with a DPB previously defined by the $ form of a directive macro. This macro pushes the DPB address onto the stack and issues an EMT 377 instruction.

The DIR$ macro generates an Executive trap using a previously defined DPB:

```
Macro Call:  DIR$  adr,err
```

Note: adr and err are optional.

adr

The address of the DPB. If specified, adr must be a valid source address for a MOV instruction. If adr is not specified, the DPB or its address must be on the stack.

err

The error return address (see Section 7.3.3). If an error return is not specified, an error sets the PSW Carry bit.

NOTE

DIR$ is not an Executive directive and does not behave as one. There are no variations in the spelling of this macro.

7.3.3 Optional Error Routine Address

The $C and $S forms of macro calls and the DIR$ macro can accept an optional final argument. The argument must be a valid assembler destination operand that specifies the address of a user error routine. For example, the DIR$ macro

```
DIR$    #DPB,ERROR
```

generates the following code:

```
MOV     #DPB,-(SP)
EMT     377
BCC     .+6
JSR     PC,ERROR
```

Since the $ form of a directive macro does not generate any executable code, it does not accept an error routine address argument.

7.3.4 Symbolic Offsets

Most system directive macro calls generate local symbolic offsets describing the format of the DPB. The symbols are unique to each directive, and each is assigned an index value corresponding to the offset of a given DPB element.

Because the offsets are defined symbolically, you can refer to or modify DPB elements without knowing the offset values. Symbolic offsets also eliminate the need to rewrite programs if a future release of the system changes a DPB specification.

All $ and $C forms of macros that generate DBPs longer than one word generate local offsets. All informational directives including the $S form, generate local symbolic offsets for the parameter block returned as well.

If the program uses either the $ or $C form and has defined the symbol $$$GLB (for example $$$GLB=0), the macro generates the symbolic offsets as global symbols and does not generate the DPB itself. The purpose of this facility is to enable the use of a

DPB defined in a different module. The symbol $$$GLB has no effect on the expansion of $S macros.

When using symbolic offsets, you should use the $ form of directives.

7.3.5 Examples of Macro Calls

The following examples show the expansions of the different macro call forms:

- The $ form generates a DPB only, in the current p-section.

  ```
  MRKT$   1,5,2,MTRAP
  ```

  generates the following code:

  ```
  .BYTE   23.,5     ; "MARK-TIME" DIC & DPB SIZE
  .WORD   1         ; EVENT FLAG NUMBER
  .WORD   5         ; TIME INTERVAL MAGNITUDE
  .WORD   2         ; TIME INTERVAL UNIT (SECONDS)
  .WORD   MTRAP     ; AST ENTRY POINT ADDRESS
  ```

- The $C form generates a DPB in p-section DPB and generates the code to issue the directive in the specified section.

  ```
  MRKT$C  1,5,2,MTRAP,PROG1,ERR
  ```

  generates the following code:

  ```
  .PSECT  $DPB$.
  $$$=.             ; DEFINE TEMPORARY SYMBOL
  .BYTE   23.,5     ; "MARK-TIME" DIC & DPB SIZE
  .WORD   1         ; EVENT FLAG NUMBER
  .WORD   5         ; TIME INTERVAL MAGNITUDE
  .WORD   2         ; TIME INTERVAL UNIT (SECS)
  .WORD   MTRAP     ; AST ENTRY POINT ADDRESS
  .PSECT  PROG1     ; RETURN TO THE ORIGINAL PSECT
  MOV     #$$$,-(SP)    ; PUSH DPB ADDRESS ON STACK
  EMT     377       ; TRAP TO THE EXECUTIVE
  BCC     .+6       ; BRANCH ON DIRECTIVE ACCEPTANCE
  JSR     PC,ERR    ; ELSE, CALL ERROR SERVICE ROUTINE
  ```

- The $S form generates code to push the DPB onto the stack and to issue the directive.

  ```
     MRKT$S         #1,#30,#2,R2,ERR
  ```

  generates the following code:

```
MOV     R2,-(SP) ; PUSH AST ENTRY POINT
MOV     #2,-(SP) ; TIME INTERVAL UNIT (SECONDS)
MOV     #30,-(SP)                ; TIME INTERVAL MAGNITUDE
MOV     #1,-(SP) ; EVENT FLAG NUMBER
MOV     (PC)+,-(SP)              ; AND "MARK-TIME" DIC & DPB
SIZE
.BYTE   23.,5    ; ON THE STACK
EMT     377      ; TRAP TO THE EXECUTIVE
BCC     .+6      ; BRANCH ON DIRECTIVE ACCEPTANCE
JSR     PC,ERR   ; ELSE, CALL ERROR SERVICE ROUTINE
```

- The DIR$ macro issues a directive that has a predefined DPB.

```
DIR$    R1,(R3)  ; DPB ALREADY DEFINED.

; DPB ADDRESS IN R1.
```

generates the following code:

```
MOV     R1,-(SP) ; PUSH DPB ADDRESS ON STACK
EMT     377      ; TRAP TO THE EXECUTIVE
BCC     .+4      ; BRANCH ON DIRECTIVE ACCEPTANCE
JSR     PC,(R3)  ; ELSE, CALL ERROR SERVICE ROUTINE
```

7.4 USING HIGH-LEVEL LANGUAGE SUBROUTINES

The system provides a set of high-level language subroutines to perform system directive operations. In general, one subroutine is available for each directive.

You can call the subroutines described in this manual from any Tool Kit high-level language that supports the PDP-11 R5 calling convention (described in Section 6.2.1). You can also call the subroutines from MACRO-11 assembly language.

All the subroutines described in this manual are in the system object module library, LB:[1,5]SYSLIB.OLB. When you link your program to form a task, the Task Builder first checks to see whether each specified subroutine is user defined. If a subroutine is not user defined, the Task Builder automatically searches for it in SYSLIB.OLB. Upon locating the subroutine in SYSLIB.OLB, the Task Builder includes it in the task image.

For details about a particular high-level language subroutine, see the directive description in Chapter 8.

Figures 7-3, 7-4, 7-5, and 7-6 show sample programs written in three high-level languages: FORTRAN, PASCAL, and BASIC-PLUS-2.

Sections following the figures provide general usage information about the subroutine calls, using the examples.

```
            program rqst

c  ------------------------<Comments>--------------------------
c  This sample FORTRAN program calls the REQUES subroutine to
c  request a task named gtim. The program illustrates use of
c  the call statement to invoke the subroutine, the rad50
c  predeclared function to convert the task name to rad50 format,
c  omission of optional arguments, and declaration of the
c  Directive Status Word (dsw) as a one-word integer.

c  Notice that the task name passed to the rad50 function must
c  appear in uppercase. Also, the gtim task must be installed
c  before it is requested.

c  In order to run this or any FORTRAN program on the Professional,
c  you must first install the run-time library, [ZŻSYS]PROF77.
c  ------------------------------------------------------------

            INTEGER*2               IDS

            TYPE *,'->Now calling reques:'

            CALL REQUES(RAD50('GTIM'),,IDS)

            TYPE *,'->Reques has been called.'
            TYPE *,'->DSW=',ids

            END
```

Figure 7-3: Sample FORTRAN Program

```
            program rqst;

{   ------------------------<Comments>-------------------------
    This sample PASCAL program calls the REQUES subroutine to
    request a task named gtim. The program illustrates the
    declaration of REQUES with the external and seq11
    attributes, invocation of the subroutine, use of the %r
    radix specifier to convert the task name to rad50 format,
    and the omission of optional arguments.

    Notice that you must blank-fill the task name to a length
    of six. Also, the gtim task must be installed before it is
    requested.
    ------------------------------------------------------------}

            type
                    four_array      =array [1..4] of integer;

            var
                    tsk             :long_integer;
                    opt             :four_array;
                    ids             :integer;

            [external(reques)] procedure request_task  (
                    var tsk:[readonly] long_integer;
                    var opt:           four_array;
                    var ids:           integer ); seq11;

            begin

            writeln('->Now calling reques:');

            request_task(%r'gtim  ',,ids);

            writeln('->Reques has been called.');
            writeln('->DSW=',ids);

            end.
```

Figure 7-4: Sample PASCAL Program

```
program rqstdir;

{  ------------------<comment>----------------------
   This PASCAL program uses the DIR$ predeclared
   function to call the RQST$ system directive.

   You must construct the Directive Parameter Block
   (DPB) in a variable or structured constant, and
   then pass it as a parameter to DIR$.
   -----------------------------------------------}

            var

            {Directive Parameter Block:}
            rqst_dpb                : packed record
                                        dic: [pos(0)]  0..255;
                                        dpb: [pos(8)]  0..255;
                                        tsk: [pos(16)] long_integer;
                                        prt: [pos(48)] integer;
                                        pri: [pos(64)] integer;
                                        ugc: [pos(80)] 0..255;
                                        umc: [pos(88)] 0..255;
                                      end;

            begin

            {Insert info into DPB:}
            rqst_dpb.dic   := 11;
            rqst_dpb.dpb   := 7;
            rqst_dpb.tsk   := %r'gtim  ';

            writeln ('Now invoking RQST$:');

            {Invoke DIR$ and pass it the DPB:}
            dir$(rqst_dpb);

            writeln ('RQST$ invoked.');

            end. {rqst}
```

Figure 7-5: Sample PASCAL Program Using DIR$ Function

```
10 !------------------------<Comments>-----------------------
   ! This sample BASIC-PLUS-2 program calls the REQUES subroutine
   ! to request a task named gtim. The program illustrates use of
   ! the call by ref statement to invoke the subroutine, use of
   ! the FSS$ function to obtain the task name in rad50 format,
   ! omission of optional arguments, and declaration of the
   ! Directive Status Word (ids) as one-word integer.
   !
   ! Note that you must install the gtim task before requesting it.
   !-------------------------------------------------------------

            declare  integer        ids
            map (a)  string         fss_output = 30
            map (a)  string         fill = 6, word tsk_r50(3)

   !-----------------convert task name to rad50---------------

            fss_output = FSS$("GTIM.TSK",1%) !output in MAP

   !----------tsk_r50 now contains rad50 version of GTIM-------

            print "->Now calling reques:"

            call reques by ref (tsk_r50(),,ids)

            print "->Reques has been called."
            print "->DSW=",ids

            end
```

Figure 7-6: Sample BASIC-PLUS-2 Program

7.4.1 Calling the Subroutines

For FORTRAN, PASCAL, and BASIC-PLUS-2, you can simply omit any optional arguments in a call, as long as you retain the comma that immediately follows the argument. If an optional argument is the last argument in an invocation, then you can also omit the comma.

Note that some subroutines use a default value for unspecified optional arguments. Such default values are noted in each subroutine description in Chapter 8.

Language-specific information on calling methods follows.

- **FORTRAN Calling Method**

  You use the CALL statement in FORTRAN to invoke a high-level language subroutine. Figure 7-3 illustrates a FORTRAN call to the REQUES subroutine.

- **PASCAL Calling Method**

  In PASCAL you must declare each high-level language subroutine that you call in your program. The declaration must begin with the EXTERNAL attribute specifying the actual name of the subroutine, and must end with the SEQ11 attribute to ensure that the compiler generates the R5 calling sequence.

  Figure 7-4 illustrates an invocation of the reques subroutine.

  To improve performance, PASCAL provides a DIR$ predeclared function that allows you to pass a Directive Parameter Block directly to the Executive. You can construct the DPB as a packed record, assign required values to the record items, and then pass the record as an argument to the DIR$ function. The DPB contains the Directive Identification Code indicating which directive you are calling.

  When you are building a DPB for use with DIR$, refer to the section in the directive description called *Macro Expansion* for a description of the DPB.

  Figure 7-5 illustrates a call to DIR$ that specifies the Request Task (RQST) directive.

- **BASIC-PLUS-2 Calling Method**

  You must use the CALL BY REF statement in BASIC-PLUS-2 to invoke a high-level language subroutine. Do not omit the pass mechanism BY REF. Figure 7-6 illustrates a BASIC-PLUS-2 call to the subroutine reques.

7.4.2 Specifying Task Names

P/OS imposes the following requirements on task names:

- The maximum length of a task name is six characters.
- The only characters permitted are the letters A through Z, the numerals 0 through 9, and the special characters dollar sign and period.
- The system recognizes a task name only if it is stored in Radix-50 format. (This permits as many as three characters to be encoded in one word.)
- Task names containing the dollar sign character are reserved for DIGITAL use.

The high-level language subroutines require that you define a task name as a two-word variable or array that contains the task name in Radix-50 format.

Language-specific information follows.

- **FORTRAN Task Names**

  In FORTRAN, you can define a task name variable as INTEGER*4 or as an INTEGER*2 array.

  You can define a task name variable at program compilation time by using a DATA statement, which gives a REAL variable an initial value (a Radix-50 constant). For example, if you use a 5-character task name such as CCMF1 in a system directive call, you could define the task name and use it as a parameter as follows:

```
DATA CCMF1/5RCCMF1/
   .
   .
   .
CALL REQUES (CCMF1)
```

  Similarly, a four-character task name definition could be:

```
DATA CCMF1/4RTNAM/
```

  Instead of using a data statment to define a static task name, a FORTRAN program can define task names during execution by using the IRAD50 subroutine or the RAD50 function. Figure 7-3 illustrates the use of the RAD50 function.

- **PASCAL Task Names**

  To specify a task name in PASCAL, use the %R radix specifier before the string constant containing your task name. Note that you must pad your task name with spaces up to six characters to ensure that PASCAL generates the required two-word value.

  Figure 7-4 illustrates the use of the %R radix specifier.

- **BASIC-PLUS-TWO Task Names**

  BASIC-PLUS-TWO provides the FSS$ function for parsing a file specification. Output from FSS$ includes a two-word area containing the filename in RAD50 format. You can use MAP storage to hold the output from FSS$ and to refer to filename portion of the output. Figure 7-6 illustrates the use of FSS$.

  An alternative to using FSS$, which should also provide better performance, is to code your own routine in MACRO to perform the conversion.

7.4.3 Specifying Integer Arguments

All integer arguments in the subroutines are one-word values. In FORTRAN, declare integer arguments as INTEGER*2. In PASCAL, you declare one-word integers as INTEGER values. In BASIC-PLUS-2, integer arguments have the WORD data type.

7.4.4 GETADR Subroutine

Some subroutine calls include an argument described as an integer array. The integer array contains some values that are the addresses of other variables or arrays. Since some languages, such as FORTRAN, do not provide a means of assigning such an address as a value, you must use the GETADR subroutine. For example:

Calling Sequence:

```
CALL GETADR(ipm,[arg1],[arg2],...[argn])
```

ipm

An array of dimension n.

arg1,...argn

Arguments whose addresses are to be inserted in ipm. Arguments are inserted in the order specified. If a null argument is specified, then the corresponding entry in ipm is left unchanged. When the argument is an array name, the address of the first array element is inserted into ipm.

FORTRAN Example:

```
DIMENSION IBUF(80),IOSB(2),IPARAM(6)
INTEGER*2 IDSW,IREAD,IEFLAG,LUN
IREAD = 512
IEFLAG = 1
IPARAM(2)=80
   .
   .
   .
CALL GETADR (IPARAM(1),IBUF(1))
CALL WTQIO (IREAD,LUN,IEFLAG,,IOSB,IPARAM,IDSW)
   .
   .
   .
```

In this example, CALL GETADR enables you to specify a buffer address in the CALL QIO directive (see Section 8.46).

7.4.5 The Subroutine Calls

Chapter 8 describes the available high-level language calls for each directive.

For some directives, notably Mark Time (CALL MARK), both the standard high-level language subroutine call and the FORTRAN ISA standard call are provided. Other directives, however, are not available to high-level language tasks. Examples are Specify Floating Point Exception AST (SFPA$) and Specify SST Vector Table For Task (SVTK$).

Table 7-1 shows directives that are not available as high-level language subroutines:

Table 7-1: Directives Without High-Level Language Subroutines

Directive	Macro Call
AST Service Exit	ASTX$S
Connect To Interrupt Vector	CINT$
Specify Floating Point Exception AST	SFPA$
Specify Receive By Reference AST	SRRA$
Specify Receive Data AST	SRDA$
Specify SST Vector Table For Debugging Aid	SVDB$
Specify SST Vector Table For Tasks	SVTK$

7.4.6 Error Conditions

Each subroutine call includes an optional argument that specifies the integer to receive the Directive Status Word (ids). When you specify this argument, the subroutine returns a value that indicates whether the directive operation succeeded or failed.

If the directive failed, the value indicates the reason for the failure. The possible values are the same as those returned to the Directive Status Word (DSW) in MACRO-11 programs--except for the two ISA calls, CALL WAIT and CALL START. The ISA calls have positive numeric error codes.

7.4.7 AST Support for High-Level Languages

Each high-level language has restrictions on handling ASTs. Consult the documentation for your particular high-level language for details. Typical restrictions (FORTRAN) are described later in this section.

The following routines provide support for ASTs in high-level languages:

```
CALL CNCT

CALL SDRC

CALL SPAWN

CALL SREX
```

Whenever you specify a high-level language AST routine to one of the system library routines listed above, the AST routine is replaced by an internal routine that saves the general purpose registers and calls the specified routine using a coroutine call when the AST occurs.

After the high-level language routine completes, by way of a RETURN statement (or equivalent), the internal routine restores the general purpose registers and issues an ASTX$ directive.

Use extreme caution when coding an AST service routine in high-level language. For instance, you cannot perform the following operations while at AST state:

- I/O of any kind

  This includes FORTRAN ENCODE and DECODE statements and internal file I/O. Fortran I/O is not reentrant; therefore, the information in the impure data area can be destroyed.

- Floating-point operations

  The floating-point processor's context is not saved while in AST state. Since the scientific subroutines use floating-point operations, they can not be called at AST state.

- Traceback information in the generated code

  Use of traceback information corrupts the error recovery in the FORTRAN run time library. Any FORTRAN modules that will be called at AST state must be compiled without traceback. See your FORTRAN user's guide for more information.

- Virtual array operations

  Use of virtual arrays at AST state remaps the current array such that any operations at non-AST state will not be executed correctly.

- Subprograms can not be shared between AST processing and normal task processing.

- Do not invoke EXIT or STOP statements with files open.

  FORTRAN flushes the task's buffers, which could be in an intermediate state. Therefore, data might be lost if any output files are open when the EXIT or STOP is executed.

  You can EXIT or STOP at AST state if no output files are open.

  Since the message put out by STOP uses a different mechanism from the normal FORTRAN I/O routines, the act of putting out this message does not corrupt impure data in the run-time system. Therefore, you can issue a STOP statement at AST state unless there are output files open.

Note also the following:

- Any execution time error at AST state will corrupt the program.

- Use extreme care if your task contains overlays. Both the interface routine and the actual code of the FORTRAN AST routine must be located in the root segment. Any routines that are called at AST state must also be in the root segment.

7.5 RESTRICTIONS ON NONPRIVILEGED TASKS

When issued by a nonprivileged task, several Executive directives are limited in their uses. Table 7-2 lists these directives and explains the restrictions.

Table 7-2: Restricted Directives Issued by Nonprivileged Tasks

Directive	Macro Call	Restrictions
Abort Task	ABRT$	A nonprivileged task can only abort tasks with the same TI: as the task issuing the directive.

RESTRICTIONS ON NONPRIVILEGED TASKS

Directive	Macro Call	Restrictions
Alter Priority	ALTP$	A nonprivileged task can only alter its own priority to values less than or equal to the task's installed priority.
Cancel Time Based Initiation Requests	CSRQ$	A nonprivileged task can issue this directive only if the TI: of the specified task is the same that of the issuing task.
Switch State	SWST$	This directive cannot be issued by a nonprivileged task under any circumstances.

7.6 DIRECTIVE CATEGORIES

This section groups the directives by function into the following ten categories:

- Task execution control directives
- Task status control directives
- Informational directives
- Event-associated directives
- Trap-associated directives
- I/O- and intertask communications-related directives
- Memory management directives

7.6.1 Task Execution Control Directives

The task execution control directives deal principally with starting and stopping tasks. Each of these directives (except Extend Task) changes the task's state (unless the task is already in the state being requested). These directives include:

Macro	Directive Name
ABRT$	Abort Task
CSRQ$	Cancel Time Based Initiation Requests
EXIT$S	Task Exit ($S form recommended)
EXTK$	Extend Task
RQST$	Request Task
SPND$S	Suspend ($S form recommended)
SWST$	Switch State

7.6.2 Task Status Control Directives

Two task status control directives alter the checkpointable attribute of a task. A third directive changes the running priority of an active task. These directives include:

Macro	Directive Name
ALTP$	Alter Priority
DSCP$S	Disable Checkpointing ($S form recommended)
ENCP$S	Enable Checkpointing ($S form recommended)

7.6.3 Informational Directives

Several directives provide the issuing task with system information and parameters such as the time of day, the task parameters, the console switch settings, and partition or region parameters. These directives include:

Macro	Directive Name
GPRT$	Get Partition Parameters
GREG$	Get Region Parameters
GTIM$	Get Time Parameters
GTSK$	Get Task Parameters

7.6.4 Event-Associated Directives

The event and event flag directives provide inter- and intratask synchronization and signaling, and the means to set the system time. Use these directives carefully since software faults resulting from erroneous signaling and synchronization are often difficult to isolate. The directives include:

Macro	Directive Name
CLEF$	Clear Event Flag
CMKT$	Cancel Mark Time Requests
DECL$S	Declare Significant Event ($S form recommended)
EXIF$	Exit If
MRKT$	Mark Time
RDEF$	Read Single Event Flag
SETF$	Set Event Flag
STIM$	Set System Time
STLO$	Stop For Logical 'OR' of Event Flags
STOP$S	Stop ($S form recommended)
STSE$	Stop For Single Event Flag
USTP$	Unstop
WSIG$S	Wait For Significant Event ($S form recommended)
WTLO$	Wait For Logical OR Of Event Flags
WTSE$	Wait For Single Event Flag

7.6.5 Trap-Associated Directives

The trap-associated directives provide trap facilities that allow transfer of control (software interrupts) to the executing tasks. These directives include:

Macro	Directive Name
ASTX$S	AST Service Exit ($S form recommended)
DSAR$S	Disable AST Recognition ($S form recommended)
ENAR$S	Enable AST Recognition ($S form recommended)
IHAR$S	Inhibit AST Recognition ($S form recommended)
SFPA$	Specify Floating Point Processor Exception AST
SRDA$	Specify Receive Data AST
SREA$	Specify Requested Exit AST
SREX$	Specify Requested Exit AST (extended)
SRRA$	Specify Receive-By-Reference AST
SVDB$	Specify SST Vector Table For Debugging Aid
SVTK$	Specify SST Vector Table For Task

7.6.6 I/O- and Intertask Communication Related Directives

The I/O- and intertask communication related directives allow tasks to access I/O devices at the driver interface level or interrupt level, to communicate with other tasks in the system, and to retrieve the MCR command line used to start the task. These directives include:

Macro	Directive Name
ALUN$	Assign LUN
CINT$	Connect To Interrupt Vector
CLOG$	Create Logical Name String
DLOG$	Delete Logical Name String
GDIR$	Get Default Directory String
GLUN$	Get LUN Information
GMCR$	Get MCR Command Line
QIO$	Queue I/O Request
QIOW$	Queue I/O Request And Wait
RCVD$	Receive Data
RCVX$	Receive Data Or Exit
SDAT$	Send Data
SDIR$	Set Default Directory String
TLOG$	Translate Logical Name String
VRCD$	Variable Receive Data
VRCS$	Variable Receive Data Or Stop
VRCX$	Variable Receive Data Or Exit
VSDA$	Variable Send Data

7.6.7 Memory Management Directives

The memory management directives allow a task to manipulate its virtual and logical address space, and to set up and control dynamically the window-to-region mapping assignments. The directives also provide the means by which tasks can share and pass references to data and routines. These directives include:

Macro	Directive Name
ATRG$	Attach Region
CRAW$	Create Address Window
CRRG$	Create Region
DTRG$	Detach Region
ELAW$	Eliminate Address Window
GMCX$	Get Mapping Context
MAP$	Map Address Window
RREF$	Receive By Reference
SREF$	Send By Reference
UMAP$	Unmap Address Window

7.6.8 Parent/Offspring Tasking Directives

Parent/offspring tasking directives permit tasks to start other tasks, and to connect to other tasks in order to receive status information. These directives include:

Macro	Directive Name
CNCT$	Connect
EMST$	Emit Status
EXST$	Exit With Status
RPOI$	Request and Pass Offspring Information
SDRC$	Send, Request, And Connect
SDRP$	Send Data, Request and Pass OCB
SPWN$	Spawn
VSRC$	Variable Send, Request, and Connect

The scheme used for task naming in Executive-level dispatching works as follows:

A single copy of the multiuser task must be installed with a name of the form ...mmm. When a task issues a directive specifying a task name of the form ...mmm, the Executive first forms the task name mmmtnn, where t is the first character of the device name of the TI: of the issuing task, and nn is the unit number.

The Executive then attempts to perform the directive as if the task name mmmtnn had been specified. If the directive is one that could activate the task (Request, Spawn, or Send, Request And Connect), a TCB can be dynamically created and filled in from the ...mmm TCB. If the directive is a send user-type directive, and the TCB mmmtnn does not exist, the send packet is queued to the ...mmm TCB until mmmtnn is activated. At that time any send packets for mmmtnn that are queued to the ...mmm TCB are moved to the mmmtnn TCB.

This allows for the specification of a specific copy of a multiuser task in a directive whose TI: is different from that of the issuing task. If the TI: of the target task is known, the task's name can be calculated and explicitly specified in a directive.

7.7 DIRECTIVE CONVENTIONS

The following are conventions for using system directives:

- In MACRO-11 programs, unless a number is followed by a decimal point, the assembler assumes the number to be octal.

  In high-level language programs, use a one-word integer unless the directive description states otherwise.

- In MACRO-11 programs, task and partition names can be from one to six characters long and must be represented as two words in Radix-50 form.

  For high-level language programs, see Section 7.4.2 for information on representing task and partition names in Radix-50 format.

- Device names are two characters long and are represented by one word of ASCII code.

- Some directive descriptions state that a certain parameter must be provided even though the system ignores it. Such parameters are included either for future extension to the system or for compatibility with programs written to run on other operating systems.

- In the directive descriptions, square brackets ([ ]) enclose optional parameters or arguments. To omit optional items, either use an empty (null) field in the parameter list or omit a trailing optional parameter.

- Logical Unit Numbers (LUNs) can range from 1 through 255 (decimal).

- Event flag numbers range from 1 through 64 (decimal); however, numbers 25 through 32 and 57 through 64 are reserved.

  Event flag numbers from 1 through 32 denote local flags. Numbers from 33 through 64 denote common flags.

Note that the Executive preserves all task registers when a task issues a directive.

CHAPTER 8

DIRECTIVE DESCRIPTIONS

This chapter describes the system directives, giving each directive's function and use. For each directive, we show the names of the high-level language and macro calls, the associated parameters, and possible return values of the Directive Status Word (DSW).

The descriptions generally describe the $ form of the macro call, although the $C and $S forms are also valid forms. (For the QIO directive, we show the QIO$ form, although the QIO$S and QIO$C forms are also valid.) However, where the $S form of a macro requires less space and performs as fast as a DIR$ (due to a smaller Directive Parameter Block), we show the $S form of the macro expansion.

In addition to the directive macros themselves, you can use the DIR$ macro to execute a directive if the directive has a predefined Directive Parameter Block (DPB). See Sections 7.3.1.1 and 7.3.2 for further details.

8.1 FORMAT OF SYSTEM DIRECTIVE DESCRIPTIONS

Each directive description includes most or all of the following elements:

High-Level Language Call

The high-level language call for each directive shows the name of the subroutine and the definitions for each of its parameters, given in a FORTRAN-like CALL statement. Note, however, that not all high-level languages use the CALL statement to execute external routines.

See Section 7.4 for information on using the high-level language subroutines.

Macro Call

The macro call for each directive shows the actual name of each parameter, and gives the defaults for optional parameters in parentheses following the parameter's description. Since zero is supplied as the default for most parameters, only nonzero default values are shown.

See Section 7.3 for information on using the directive macros.

Macro Expansion

Most of the directive descriptions expand the $ form of the macro, except where the $S form is recommended instead. Section 7.3.5 illustrates expansions for all three forms and for the DIR$ macro.

Definition Block Parameters

Only the memory management directives include these parameters. For such directives, a table describes all the relevant input and output parameters in the Region or Window Definition Block. (See Section 5.5.)

Local Symbol Definitions

Macro expansions usually generate local symbol definitions with an assigned value equal to the byte offset from the start of the DPB to the corresponding DPB element. This section lists those symbols. The length in bytes of the element pointed to by the symbol appears in parentheses following the symbol's description:

```
A.BTTN Task name (4)
```

defines A.BTTN as pointing to a task name in the Abort Task DPB; the task name has a length of four bytes.

DSW Return Codes

This section lists all valid return codes.

Implemented Subfunctions

Some directives, notably WIMP$, perform slightly different operations depending on the subfunction you select. The section on implemented subfunctions shows you how to specify the subfunction in your system call, and what parameters are defined for each subfunction.

Notes

The notes presented with some directive descriptions expand on the function, use, and consequences of using the directive. Always read the notes carefully.

8.2 ABRT$ - ABORT TASK

The Abort Task directive instructs the system to terminate execution of the indicated task. ABRT$ is intended for use as an emergency or fault exit.

A task must be privileged to issue the Abort Task directive (unless it is aborting a task with the same TI:).

High-Level Language Call

```
CALL ABORT (tsk[,ids])
```

tsk	Name of the task to be aborted (RAD50)
ids	Directive status

Macro Call

```
ABRT$   tsk
```

tsk	Name of the task to be aborted (RAD50)

Macro Expansion

```
ABRT$   ALPHA
.BYTE   83.,3           ;ABRT$ MACRO DIC, DPB SIZE=3 WORDS
.RAD50  /ALPHA/         ;TASK "ALPHA"
```

Local Symbol Definitions

A.BTTN	Task name (4)

DSW Return Codes

IS.SUC	Successful completion.
IE.INS	Task not installed.
IE.ACT	Task not active.
IE.PRI	Issuing task is not privileged (multiuser protection systems only).
IE.ADP	Part of the DPB is out of the issuing task's address space.
IE.SDP	Directive Identification Code (DIC) or DPB size is invalid.

Notes

1. When a task is aborted, the Executive frees all the task's resources. In particular, the Executive:

   - Detaches all attached devices.
   - Flushes the AST queue and despecifies all specified ASTs.
   - Flushes the receive and receive-by-reference queue.
   - Flushes the clock queue for outstanding Mark Time requests for the task.
   - Closes all open files (files open for write access are locked).
   - Detaches all attached regions except in the case of a fixed task, where no detaching occurs.
   - Runs down the task's I/O.
   - Disconnects from interrupt vectors.
   - Breaks the connection with any offspring tasks.
   - Returns a severe error status (EX$SEV) to the parent task when a connected task is aborted.
   - Frees the task's memory if the aborted task was not fixed.

2. If the aborted task had a requested exit AST specified, the task will receive that AST instead of being aborted. No indication that this has occurred is returned to the task that issued the abort request.

3. When the aborted task actually exits, the Executive declares a significant event.

8.3 ACHN$ - ASSIGN CHANNEL

Assign channel and Assign Lun (ALUN$) are similar in that both assign a logical unit number to a device. For ACHN$, however, instead of supplying the device name as with ALUN$, you can supply a file specification or logical name. The ACHN$ directive expands the file specification or logical and then assigns the LUN you specify to the device resulting from that expansion.

High-Level Language Call

```
CALL ACHN ([mod],[tbmsk],lun,fsbuf,fssz,[idsw])
```

mod	Modifier for logical name table entries
tbmsk	Inhibit mask to prevent a logical table from being searched
fsbuf	Array containing file specification
fssz	Size of fsbuf in bytes
idsw	Integer to receive directive status word

Macro Call

```
ACHN$ [mod],tbmsk,lun,fsbuf,fssz
```

mod	Optional modifier for logical name table entries
tbmsk	A byte whose low 4 bits constitute an inhibit mask to prevent the system from searching a particular logical table. For each of the following bits, if the bit is set, the system does not search the corresponding table:

Logical Name Table	Bit of tbmsk	Octal
System table (LT.SYS)	0	1
User table (LT.USR)	2	4
Session table (LT.SES)	4	20

lun	The LUN to be assigned
fsbuf	Address of file specification buffer
fssz	Size of the file specification buffer in bytes

Macro Expansion

```
.MACRO    ACHN$       MOD,TBMSK,LUN,FSBUF,FSSZ
.BYTE     207.,5      ;ACHN$ MACRO DIC, DPB SIZE = 5 WORDS
.BYTE     6           ;SUBFUNCTION
.BYTE     MOD         ;MODIFIER
.BYTE     LUN         ;LOGICAL UNIT NUMBER
.BYTE     TBMSK       ;TABLE MASK
.WORD     FSBUF       ;FILE SPECIFICATION BUFFER ADDRESS
.WORD     FSSZ        ;SIZE OF FILE SPECIFICATION BUFFER
```

Local Symbol Definitions

A.LFUN	Subfunction code (1)
A.LMOD	Logical name modifier (1)
A.LLUN	LUN number (1)
A.LTBL	Table inhibit mask (1)
A.LSBF	Address of file specification buffer (2)
A.LSSZ	File specification buffer size in bytes (2)

DSW Return Codes

IS.SUC	Successful completion.
IE.ADP	Part of the DPB or user buffer is out of the issuing task's address space, or the user does not have proper access to that region.
IE.SDP	DIC or DPB size is invalid.
IE.IDU	Invalid device or unit.
IE.ILU	Invalid LUN.
IE.LNL	LUN usage is interlocked.

8.4 ALTP$ - ALTER PRIORITY

The Alter Priority directive instructs the system to change the running priority of a specified active task to either a new priority indicated in the directive call, or to the task's default (installed) priority if the call does not specify a new priority.

The specified task must be installed and active. The Executive resets the task's priority to its installed priority when the task exits.

If the directive call omits a task name, the Executive defaults to the issuing task.

The Executive reorders any outstanding I/O requests for the task in the I/O queue and reallocates the task's partition. The partition reallocation can cause the task to be checkpointed.

A nonprivileged task can issue ALTP$ only for itself, and only for a priority equal to or lower than its installed priority. A privileged task can change the priority of any task to any value less than 250.

High-Level Language Call

```
CALL ALTPRI ([tsk],[ipri][,ids])
```

tsk	Active task name
ipri	A one-word integer value equal to the new priority, a number from 1 through 250 (decimal)
ids	Directive status

Macro Call

```
ALTP$  [name][,pri]
```

name	Active task name
pri	New priority number, 1 through 250 (decimal)

Macro Expansion

```
ALTP$     ALPHA, 75.
.BYTE     9.,4          ;ALTP$ MACRO DIC, DPB SIZE=4 WORDS
.RAD50    /ALPHA/       ;TASK ALPHA
.WORD     75.           ;NEW PRIORITY
```

Local Symbol Definitions

A.LTTN	Task name (4)
A.LTPR	Priority (2)

DSW Return Codes

IS.SUC	Successful completion.
IE.INS	Task not installed.
IE.ACT	Task not active.
IE.PRI	Issuing task is not privileged.
IE.IPR	Invalid priority.
IE.ADP	Part of the DPB is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

8.5 ALUN$ - ASSIGN LUN

The Assign LUN directive instructs the system to assign a physical device unit to a logical unit number (LUN). It does not indicate that the task has attached itself to the device.

The actual physical device assigned to the logical unit is dependent on the logical assignment table. The Executive first searches the logical assignment table for a device name match. If it finds a match, the Executive assigns the physical device unit associated with the matching entry to the logical unit. Otherwise, the Executive searches the physical device tables and assigns the actual physical device unit named to the logical unit. The Executive does not search the logical assignment table for slaved tasks.

When a task reassigns a LUN from one device to another, the Executive cancels all I/O requests for the issuing task in the previous device queue.

High-Level Language Call

```
CALL ASNLUN (lun,dev,unt[,ids])
```

lun	Logical unit number
dev	Device name (format: 1A2)
unt	Device unit number
ids	Directive status

Macro Call

```
ALUN$ lun,da,du
```

lun	Logical unit number
da	Device name (two characters)
du	Device unit number

Macro Expansion

```
ALUN$     7,TT,0          ;ASSIGN LOGICAL UNIT NUMBER
.BYTE     7,4             ;ALUN$ MACRO DIC, DPB SIZE=4 WORDS
.WORD     7               ;LOGICAL UNIT NUMBER 7
.ASCII    /TT/            ;DEVICE NAME IS TT (TERMINAL)
.WORD     0               ;DEVICE UNIT NUMBER=0
```

Local Symbol Definitions

A.LULU	Logical unit number (2)
A.LUNA	Physical device name (2)
A.LUNU	Physical device unit number (2)

DSW Return Codes

IS.SUC	Successful completion.
IE.LNL	LUN usage is interlocked. (See Note 1 below.)
IE.IDU	Invalid device and/or unit.
IE.ILU	Invalid logical unit number.
IE.ADP	Part of the DPB is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

Note

A return code of IE.LNL indicates that the specified LUN cannot be assigned as directed. Either the LUN is already assigned to a device with a file open for that LUN, or the LUN is currently assigned to a device attached to the task, and the directive attempted to change the LUN assignment. If a task has a LUN assigned to a device and the task has attached the device, the LUN can be reassigned, provided that the task has another LUN assigned to the same device.

8.6 ASTX$S - AST SERVICE EXIT

The AST Service Exit directive instructs the system to terminate execution of an AST service routine.

If another AST is queued and ASTs are not disabled, then the Executive immediately effects the next AST. Otherwise, the Executive restores the task's pre-AST state. See Notes.

High-Level Language Call

Neither the FORTRAN language nor the ISA standard permits direct linking to system trapping mechanisms. (Refer to Section 7.4.7 for more information on this subject.) Therefore, this directive is not available to high-level language programmers.

Macro Call

```
ASTX$S  [err]
```

err	Error routine address

Macro Expansion

```
ASTX$S  ERR
MOV     (PC)+,-(SP)     ;PUSH DPB ONTO THE STACK
.BYTE   115.,1          ;ASTX$S MACRO DIC, DPB SIZE=1 WORD
EMT     377             ;TRAP TO THE EXECUTIVE
JSR     PC,ERR          ;CALL "ERR" IF UNSUCCESSFUL
```

Local Symbol Definitions

None

DSW Return Codes

IS.SUC	Successful completion.
IE.AST	Directive not issued from an AST service routine.
IE.ADP	Part of the DPB or stack is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

Notes

1. A return to the AST service routine occurs if, and only if, the directive is rejected. Therefore, no Branch On Carry Clear instruction is generated if an error routine address is given. (The return occurs only when the Carry bit is set.)

2. When an AST occurs, the Executive pushes, at minimum, the following information onto the task's stack:

   SP+06 Event flag mask word

   SP+04 PS of task prior to AST

   SP+02 PC of task prior to AST

   SP+00 DSW of task prior to AST

   The task stack must be in this state when the AST Service Exit directive is executed.

   In addition to the data parameters, the Executive pushes supplemental information onto the task stack for certain ASTs. For I/O completion, the stack contains the address of the I/O status block; for Mark Time, the stack contains the Event Flag Number; for a floating-point processor exception, the stack contains the exception code and address.

   These AST parameters must be removed from the task's stack prior to issuing an AST exit directive. The following example shows how to remove AST parameters when a task uses an AST routine on I/O completion:

```
;
; EXAMPLE PROGRAM
;
; LOCAL DATA
;

IOSB:       .BLKW           2 ;I/O STATUS DOUBLEWORD
BUFFER:     .BLKW           30. ;I/O BUFFER

;
; START OF MAIN PROGRAM
;

START:      .               ;PROCESS DATA
    .
    .
    QIOW$C  IO.WVB,2,1,,IOSB,ASTSER,<BUFFER,60.,40>
```

```
        .
        .                           ;PROCESS & WAIT
        .
        EXIT$S                      ;EXIT TO EXECUTIVE

;
; AST SERVICE ROUTINE
;

ASTSER:                              ;PROCESS AST
                .
                .
                .
                TST (SP)+           ;REMOVE IOSB ADDRESS
                ASTX$S               ;AST EXIT
```

3. The task can alter its return address by manipulating the information on its stack prior to executing an AST exit directive. For example, to return to task state at an address other than the pre-AST address indicated on the stack, the task can simply replace the PC word on the stack. This procedure can be useful in those cases in which error conditions are discovered in the AST routine; but you should use extreme caution when doing this alteration since AST service routine bugs are difficult to isolate.

4. Because this directive requires only a one-word DPB, the $S form of the macro is recommended. It requires less space and executes with the same speed as the DIR$ macro.

8.7 ATRG$ - ATTACH REGION

The Attach Region directive attaches the issuing task to a static common region or to a named dynamic region. (No other type of region can be attached to the task by means of this directive.) The Executive checks the desired access specified in the region status word against the owner UIC and the protection word of the region. If there is no protection violation, the Executive grants the desired access. If the region is successfully attached to the task, the Executive returns a 16-bit region ID (in R.GID), which the task uses in subsequent mapping directives.

You can also use the directive to determine the ID of a region already attached to the task. In this case, the task specifies the name of the attached region in R.GNAM and clears all four bits described below in the region status word R.GSTS. When the Executive processes the directive, it checks that the named region is attached. If the region is attached to the issuing task, the Executive returns the region ID, as well as the region size, for the task's first attachment to the region. You might want to use the Attach Region directive in this way to determine the region ID of a common block attached to the task at task-build time.

High-Level Language Call

```
CALL ATRG (irdb[,ids])
```

irdb	An eight-word integer array containing a Region Definition Block (see Section 5.5.1.2)
ids	Directive status

Macro Call

```
ATRG$   rdb
```

rdb	Region Definition Block (RDB) address

Macro Expansion

```
ATRG$   RDBADR
.BYTE   57.,2       ;ATRG$ MACRO DIC, DPB SIZE=2 WORDS
.WORD   RDBADR      ;RDB ADDRESS
```

Definition Block Parameters

Table 8-1 shows the Region Definition Block parameters for this directive.

Table 8-1: Region Definition Block Parameters for ATRG$

Array Element	Offset	Description
Input Parameters		
irdb(3)(4)	R.GNAM	Name of the region to be attached
irdb(7)	R.GSTS	Bit settings* in the region status word (specifying desired access to the region): **Bit** **Definition** RS.RED 1 if read access desired RS.WRT 1 if write access desired RS.EXT 1 if extend access desired RS.DEL 1 if delete access desired Clear all four bits to request the region ID and region size of the named region if it is already attached to the issuing task.
Output Parameters		
irdb(1)	R.GID	ID assigned to the region
irdb(2)	R.GSIZ	Size in 32-word blocks of the attached region

* If you are a high-level language programmer, see Section 5.5.1 to determine the bit values represented by the symbolic names described.

Local Symbol Definition

A.TRBA	Region Definition Block address (2)

DSW Return Codes

IS.SUC	Successful completion.
IE.UPN	An attachment descriptor cannot be allocated.
IE.PRI	Privilege violation.
IE.NVR	Invalid region ID.
IE.PNS	Specified region name does not exist.
IE.HWR	Region had parity error or load failure.
IE.ADP	Part of the DPB or RDB is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

8.8 CINT$ - CONNECT TO INTERRUPT VECTOR

The Connect to Interrupt Vector directive enables a task to process hardware interrupts through a specified vector. The Interrupt Service Routine is included in the task's own space. The issuing task must be privileged.

System overhead entails execution of approximately ten instructions before entry into the ISR, and ten instructions after exit from the ISR. The Executive provides a mechanism for transfer of control from the ISR to task-level code, using either an AST or a local event flag.

After a task has connected to an interrupt vector, it can process interrupts on three different levels: interrupt, fork, and task.

- **Interrupt Level**

  When an interrupt occurs, control is transferred, with the Interrupt Transfer Block (ITB) that has been allocated by the CINT$ directive, to the Executive subroutine $INTSC. From there, control goes to the ISR specified in the directive.

  The ISR processes the interrupt and either dismisses the interrupt directly or enters fork level through a call to the Executive routine $FORK2.

- **Fork Level**

  The fork-level routine executes at priority 0, the lowest processor priority. This allows interrupts and more time-dependent tasks to be serviced promptly. If required, the fork routine sets a local event flag for the task and/or queues an AST to an AST routine specified in the directive.

- **Task Level**

  At task level, entered as the result of a local event flag or an AST, the task does final interrupt processing and has access to Executive directives. The task level can be subdivided into AST level and non-AST level.

Typically, the ISR does the minimal processing required for an interrupt and stores information for the fork routine or task-level routine in a ring buffer. The fork routine is entered after a number of interrupts have occurred, as deemed necessary by the ISR, and further condenses the information.

Finally, the fork routine wakes up the task-level code for ultimate processing, which requires access to Executive directives. The fork level can, however, be a transient stage from ISR to task-level code without doing any processing.

A task must be built privileged in order to use the CINT$ directive. However, it is valid to use the /PR:0 switch to the Task Builder to have "unprivileged mapping," that is, up to 32K words of virtual address space available. This precludes use of the Executive subroutines from task-level code; however, the ISR and fork-level routines are always mapped to the Executive when they are executed. You should always include the Executive symbol table file (on P/OS floppy PRODCL2:[ZZPRIVDEV]POS.STB) as input to the Task Builder.

As described in the notes section, special considerations apply to the mapping of the ISR, fork routine, and enable/disable routine, as well as all task data buffers accessed by these routines.

High-Level Language Call

Not supported

Macro Call

```
CINT$   vec,base,isr,dsi,psw,ast
```

vec — Interrupt vector address--must be in the range 60 (octal) to 376 (octal) inclusive, and must be a multiple of 4.

base — Virtual base address for kernel APR 5 mapping of the ISR and enable/disable interrupt routines. This address is automatically truncated to a 32 (decimal) word boundary. The "base" argument is ignored in an unmapped system.

isr — Virtual address of the ISR, or 0 to disconnect from the interrupt vector.

dsi — Virtual address of the enable/disable interrupt routine.

psw — Initial priority at which the ISR is to execute. This is normally equal to the hardwired interrupt priority, and is expressed in the form n*40, where n is a number in the range 0-7. This form puts the value in bits 5-7 of psw.

We recommend that you make use of the symbols PR4, PR5, PR6, and PR7 for this purpose. These are implemented via the macro HWDDF$ found on the P/OS floppy PRODCL2: in [ZZPRIVDEV]EXEMC.MLB. Also, you should be sure to specify the correct value for this parameter. An incorrect initial priority (for example, specifying PR4 for a device that interrupts at PR5) can cause a system crash.

ast — Virtual address of an AST routine to be entered after the fork-level routine queues an AST.

To disconnect form interrupts on a vector, the argument isr is set to 0 and the arguments base, dsi, psw, and ast are ignored.

Macro Expansion

```
CINT$   420,BADR,TADR,EDADR,PR5,ASTADR
.BYTE   129.,7          ;CINT$ MACRO DIC, DPB SIZE=7 WORDS
.WORD   420.            ;INTERRUPT VECTOR ADDRESS = 420
.WORD   BADR            ;VIRTUAL BASE ADDRESS FOR KERNEL APR
.WORD   IADR            ;VIRTUAL ADDRESS OF THE INTERRUPT
                        ;SERVICE ROUTINE
.WORD   EDADR           ;VIRTUAL ADDRESS OF THE INTERRUPT
                        ;ENABLE/DISABLE ROUTINE
.BYTE   PR5,0           ;INITIAL INTERRUPT SERVICE ROUTINE
                        ;PRIORITY (LOW BYTE). HIGH BYTE = 0.
.WORD   ASTADR          ;VIRTUAL ADDRESS OF AST ROUTINE
```

Local Symbol Definitions

C.INVE — Vector address (2)

C.INBA — Base address (2)

C.INIS — ISR address (2)

C.INDI — Enable/disable interrupt routine address (2)

C.INPS — Priority (2)

C.INAS — AST address (2)

DSW Return Codes

IE.UPN — An ITB could not be allocated (no pool space).

IE.ITS — The function requested is "disconnect" and the task is not the owner of the vector.

IE.PRI	Issuing task is not privileged.
IE.RSU	The specified vector is already in use.
IE.ILV	The specified vector is invalid (lower than 60 octal or higher than 376 octal, or not a multiple of 4).
IE.MAP	ISR or enable/disable interrupt routine is not within 4K words from the value (base address & 177700).
IE.ADP	Part of the DPB is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

Notes

1. Checkpointable Tasks

Note the following points regarding checkpointable tasks:

- When a task connects to an interrupt vector, checkpointing of the task is automatically disabled.
- When a task disconnects from a vector and is not connected to any other vector, checkpointing of the task is automatically enabled, regardless of its state before the first connect or any change in state while the task was connected.

2. Mapping Considerations

The argument "base," after being truncated to a 32-word boundary, is the start of a 4K word area mapped in kernel APR 5. All code and data in the task that the routines use must fall within that area, or a fatal error will occur--probably resulting in a system crash.

Furthermore, the code and data must be either position independent or coded in such a way that the code can execute in APR 5 mapping. (See the *PDP-11 MACRO-11 Language Reference Manual* for more information on position-independent code.) When the routines execute, the processor is in kernel mode, and the virtual address space includes all of the Executive, the pool, and the I/O page.

References within the task image must be PC-relative or use a special offset defined below. References outside the task image must be absolute.

3. **ISR**

When the ISR is entered, R5 points to the fork block in the Interrupt Transfer Block (ITB), and R4 is saved and free to be used. If you use registers R0 through R3, you must save them. If one ISR services mulitple vectors, the interrupting vector can be identified by the vector address, which is stored at offset X.VEC in the ITB. The following example loads the vector address into R4:

```
MOV X.VEC-X.FORK(R5),R4
```

The ISR either dismisses the interrupt directly by an RTS PC instruction, or calls $FORK2 if the fork routine is to be entered. When calling $FORK2, R5 must point to the fork block in the ITB, and the stack must be in the same state as it was upon entry to the ISR. The call must use absolute addressing, as follows:

```
CALL @#$FORK2
```

NOTE

Do not put the ISR in a common. Commons can be checkpointed or shuffled independently from the task; the Executive disables checkpointing and shuffling only for the task region.

4. **Fork-Level Routine**

The fork-level routine starts immediately after the call to $FORK2. On entry, R4 and R5 are the same as when $FORK2 was called. All registers are free to be used. The first instruction of the fork routine must be CLR @R3, which declares the fork block to be free.

The fork-level routine should be entered if servicing the interrupt takes more than 500 microseconds. It must be entered if an AST is to be queued or an event flag is to be set. (Fork level is described in greater detail in the *Guide to Writing a P/OS I/O Driver* and *Advanced Programmer's Notes*.)

You can queue an AST by calling the subroutine $QASTC:

Input: R5 pointer to fork block in the ITB.

Output: if AST successfully queued, carry bit = 0
if AST was not specified by CINT$, carry bit = 1

Registers
Altered: R0, R1, R2, and R3

You can set an event flag by calling the subroutine $SETF:

Input: R0 Event flag number

R5 Task Control Block (TCB) address of task for which flag is to be set. This is usually, but not necessarily, the task that has connected to the vector. This task's TCB address is at offset X.TCB in the ITB.

Output: Specified event flag set

Registers
Altered: R1 and R2

Note that absolute addressing must be used when calling these routines (and any other Executive subroutines) from fork level:

```
CALL @#$QASTC

CALL @#$SETF
```

5. Enable/Disable Interrupt Routine

The purpose of the enable/disable interrupt routine, whose address is included in the directive call, is to allow the user to have a routine automatically called in the following three cases:

- When the directive is successfully executed to connect to an interrupt vecotr (argument isr is nonzero), the routine is called immediately before return to the task.
- When the directive is successfully executed to disconnect from an interupt vector (argument isr=0).

- When the task is aborted or exits with interrupt vectors still connected.

In the first case, the routine is called with the Carry bit cleared; in the other cases, it is called with the Carry bit set. In all three cases, R1 is a pointer to the Interrupt Transfer Block (ITB). Registers R0, R2, and R3 are free to be used; other registers must be returned unmodified. Return is accomplished by means of an RTC PC instruction.

Typically, the routine dispatches to one of two routines, depending on whether the Carry bit is cleared or set. One routine sets interrupt enable and performs any other necessary initialization; the other clears interrupt enable and cleans up.

Note that the ITB contains the vector address, in the event that common code is used for multiple vectors.

6. **AST Routine**

The fork routine can queue an AST for the task through a call to the Executive routine $QASTC. When the AST routine is entered (at task level), the top work of the stack contains the vector address and must be popped off the stack before AST exit (ASTX$S).

7. **ITB Structure**

The following offsets are defined relative to the start of the ITB:

X.LNK	Link word
X.JSR	Subroutine call to $intsc
X.PSW	PSW for ISR (low-order byte)
X.ISR	ISR address (relocated)
X.FORK	Start of fork block
X.REL	APR 5 relocation
X.DSI	Address of enable/disable interrupt routine (relocated)

X.TCB	TCB address of owning task
X.AST	Start of AST block
X.VEC	Vector address
X.VPC	Saved PC from vector
X.LEN	Length in bytes of ITB

The symbols X.LNK through X.TCB are defined locally by the macro ITBDF$, which is included on the P/OS floppy PRODCL2: in [ZZPRIVDEV]EXEMC.MLB. All symbols are defined globally by PRODCL2:[ZZPRIVDEV]EXELIB.OLB.

8.9 CLEF$ - CLEAR EVENT FLAG

The Clear Event Flag directive instructs the system to report an indicated event flag's polarity and then clear it.

High-Level Language Call

```
CALL CLREF (efn[,ids])
```

efn	Event flag number
ids	Directive status

Macro Call

```
CLEF$   efn
```

efn	Event flag number

Macro Expansion

```
CLEF$   52.
.BYTE   31.,2           ;CLEF$ MACRO DIC, DPB SIZE=2 WORDS
.WORD   52.             ;EVENT FLAG NUMBER 52.
```

Local Symbol Definitions

C.LEEF	Event flag number (2)

DSW Return Codes

IS.CLR	Successful completion; flag was already clear.
IS.SET	Successful completion; flag was set.
IE.IEF	Invalid event flag number (EFN<1 or EFN>64).
IE.ADP	Part of the DPB is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

8.10 CLOG$ - CREATE LOGICAL NAME STRING

The Create Logical Name String directive establishes the relationship between a logical name string and an equivalence value string. The maximum length for each string is 255 (decimal) characters. If you create a logical name string with the same name, modifier, and table as an existing logical name string, the new definition supersedes the old one.

High-Level Language Call

```
CALL CRELOG (mod,itbnum,lns,lnssz,iens,ienssz,idsw)
```

mod	The modifier of the logical name within a table See Section 2.2 for details.
itbnum	The logical name table number: system (LT.SYS) = 0 session (LT.SES) = 4 user (LT.USR) = 2
lns	Character array containing the logical name string
lnssz	Size (in bytes) of the logical name string
iens	Character array containing the equivalence name string
ienssz	Size (in bytes) of the equivalence name string
idsw	Integer to receive the Directive Status Word

Macro Call

```
CLOG$  mod,tbnum,lns,lnssz,ens,enssz
```

mod	The modifier of the logical name within a table. See Section 2.2 for details.
tbnum	The logical name table number: system (LT.SYS) = 0 session (LT.SES) = 4 user (LT.USR) = 2
lns	Character array containing the logical name string

lnssz	Size (in bytes) of the logical name string
iens	Character array containing the equivalence name string
ienssz	Size (in bytes) of the equivalence name string

Macro Expansion

```
CLOG$   MOD,TBNUM,LNS,LNSSZ,ENS,ENSSZ
.BYTE   207.,7          ;CLOG$ MACRO DIC, DPB SIZE = 7 WORDS
.BYTE   0               ;SUBFUNCTION
.BYTE   MOD             ;LOGICAL NAME MODIFIER
.BYTE   TBNUM           ;LOGICAL NAME TABLE NUMBER
.BYTE   0               ;RESERVED FOR FUTURE USE
.WORD   LNS             ;ADDRESS OF LOGICAL NAME BUFFER
.WORD   LNSSZ           ;BYTE COUNT OF LOGICAL NAME STRING
.WORD   ENS             ;ADDRESS OF EQUIVALENCE BUFFER
.WORD   ENSSZ           ;BYTE COUNT OF EQUIVALENCE STRING
```

Local Symbol Definitions

C.LENS	Address of Equivalence name string (2)
C.LESZ	Byte count of equivalence name string (2)
C.LFUN	Subfunction (1)
C.LLNS	Address of logical name string (2)
C.LLSZ	Byte count of logical name string (2)
C.LMOD	Logical name modifier (1)
C.LTBL	Logical table number (1)

DSW Return Codes

IS.SUC	Successful completion of service.
IS.SUP	Successful completion of service. A new equivalence name string superseded a previously specified name string.
IE.UPN	Insufficient dynamic storage is available to create the logical name.
IE.IBS	The length of the logical or equivalence string is invalid. Each string length must be greater than 0 but not greater than 255 (decimal) characters.

IE.ITN	Invalid table number specified.
IE.ADP	Part of the DPB or user buffer is out of the issuing task's address space, or the user does not have proper access to that region.
IE.SDP	DIC or DPB size is invalid.

8.11 CMKT$ - CANCEL MARK TIME REQUESTS

The Cancel Mark Time Requests directive instructs the system to cancel a specific Mark Time Request or all Mark Time requests that have been made by the issuing task.

High-Level Language Call

```
CALL CANMT ([efn][,ids])
```

efn — Event flag number

ids — Directive status

Macro Call

```
CMKT$   [efn,ast,err]
```

efn — Event flag number

ast — Mark time AST address

err — Error routine address

Macro Expansion

```
CMKT$   52.,MRKAST,ERR ;TWO ARGUMENTS ARE IGNORED
.BYTE   27.,3          ;CMKT$ MACRO DIC, DPB SIZE=3 WORDS
.WORD   52.            ;EVENT FLAG NUMBER 52.
.WORD   MRKAST         ;ADDRESS OF MRKT$ REQUEST AST ROUTINE
```

NOTE

The above example will cancel only the Mark Time requests that were specified with efn 52 or the AST address MRKAST. If no ast or efn parameters are specified, all Mark Time requests issued by the task are canceled and the DPB size will equal 1.

Local Symbol Definitions

C.MKEF — Event flag number (2)

C.MKAE — Mark Time Request AST routine address (2)

DSW Return Codes

IS.SUC	Successful completion.
IE.ADP	Part of the DPB is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

Notes

1. If neither the efn nor ast parameters are specified, all Mark Time requests issued by the task are canceled. In addition, the DPB size will be one word. (When either the efn and/or ast parameters are specified, the DPB size will be three words.)

2. If both efn and ast parameters are specified (and nonzero), only Mark Time Requests issued by the task specifying either that event flag or AST address are canceled.

3. If only one efn or ast parameter is specified (and nonzero), only Mark Time Requests issued by the task specifying the event flag or AST address are canceled.

8.12 CNCT$ - CONNECT

The Connect directive synchronizes the task issuing the directive with the exit or emit status of another task (offspring) that is already active. Execution of this directive queues an Offspring Control Block (OCB) to the offspring task, and increments the issuing task's rundown count (contained in the issuing task's Task Control Block). The rundown count is maintained to indicate the combined total number of tasks presently connected as offspring tasks and the total number of virtual terminals the task has created.

Except when using the high-level language call CNCTN, the exit AST routine is called when the offspring exits or emits status with the address of the associated exit status block on the stack.

For high-level languages, call CNCTN instead of CNCT when you do not use ASTs. Using CNCTN stops the system from bringing an additional impure area into your task root, thus saving virtual address space. The interface routines would normally use the additional impure area to save context during an AST.

High-Level Language Call

```
CALL CNCT (rtname,[iefn],[iast],[iesb],[iparm][,ids])

CALL CNCTN (rtname,[iefn],[iast],[iesb],[iparm][,ids])
```

rtname — A single-precision, floating-point variable containing the offspring task name in Radix-50 format.

iefn — Event flag to be set when the offspring task exits or emits status.

iast — Name of an AST routine to be called when the offspring task exits or emits status. This parameter is ignored when calling CNCTN.

NOTE

Refer to Section 7.4.7 for important guidelines on using high-level language AST service routines.

iesb — Name of an eight-word status block to be written when the offspring task exits or emits status.

Word 0 - Offspring task exit status

Word 1 - System abort code

Word 2-7 - Reserved

NOTE

The exit status block defaults to one word. To use the eight-word exit status block, you must specify the logical OR of the symbol SP.WX8 and the event flag number in the iefn parameter above.

iparm — Name of a word to receive the status block address when an AST occurs.

ids — Integer to receive the Directive Status Word.

Macro Call

```
CNCT$   tname,[efn],[east],[esb]
```

tname — Name (RAD50) of the offspring task to be connected

efn — The event flag to be cleared on issuance and set when the offspring task exits or emits status

east — Address of an AST routine to be called when the offspring task exits or emits status

esb — Address of an eight-word status block to be written when the offspring task exits or emit status

Word 0 - Offspring task exit status

Word 1 - System abort code

Word 2-7 - Reserved

NOTE

The exit status block defaults to one word. To use the eight-word exit status block, you must specify the logical OR of the symbol SP.WX8 and the event flag number in the efn parameter above.

Macro Expansion

```
CNCT$     ALPHA,1,CONAST,STBUF
.BYTE     143.,6          ;CNCT$ MACRO DIC, DPB SIZE=6 WORDS
.RAD50    ALPHA           ;OFFSPRING TASK NAME
.BYTE     1               ;EVENT FLAG NO = 1
.BYTE     16.             ;EXIT STATUS BLOCK CONSTANT
.WORD     CONAST          ;AST ROUTINE ADDRESS
.WORD     STBUF           ;EXIT STATUS BLOCK ADDRESS
```

Local Symbol Definitions

C.NCTN	Task name (4)
C.NCEF	Event flag (2)
C.NCEA	AST routine address (2)
C.NCES	Exit status block address (2)

DSW Return Codes

IS.SUC	Successful completion.
IE.UPN	Insufficient dynamic memory to allocate an offspring control block.
IE.INS	The specified task was a command line interpreter.
IE.ACT	The specified task was not active.
IE.IEF	Invalid event flag number (EFN<0 or EFN>64).
IE.ADP	Part of the DPB or exit status block is not in the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

Note

Do not change the virtual mapping of the exit status block while the connection is in effect. Doing so can cause obscure errors since the exit status block is always returned to the virtual address specified, regardless of the physical address to which it is mapped.

8.13 CRAW$ - CREATE ADDRESS WINDOW

The Create Address Window directive creates a new virtual address window by allocating a window block from the header of the issuing task and establishing its virtual address base and size. (Space for the window block has to be reserved at task-build time by means of the WNDWS keyword.) Execution of this directive unmaps and then eliminates any existing windows that overlap the specified range of virtual addresses. If the window is successfully created, the Executive returns an 8-bit window ID to the task.

The 8-bit window ID returned to the task is a number between 1 and 23, which is an index to the window block in the task's header. The window block describes the created address window.

If WS.MAP in the window status word is set, the Executive proceeds to map the window according to the Window Definition Block (WDB) input parameters.

A task can specify any length for the mapping assignment that is less than or equal to both the window size specified when the window was created, and the length remaining between the specified offset within the region and the end of the region.

If W.NLEN is set to 0, the length defaults to either the window size or the length remaining in the region, whichever is smaller. (Because the Executive returns the actual length mapped as an output parameter, the task must clear that offset before issuing the directive each time it wants to default the length of the map.)

The values that can be assigned to W.NOFF depend on the setting of bit WS.64B in the window status word (W.NSTS):

- If WS.64B = 0, the offset specified in W.NOFF must represent a multiple of 256 words (512 bytes). Because the value of W.NOFF is expressed in units of 32-word blocks, the value must be a multiple of 8.

- If WS.64B = 1, the task can align on 32-word boundaries. The programmer can therefore specify any offset within the region.

High-Level Language Call

```
CALL CRAW (iwdb[,ids])
```

iwdb	An eight-word integer array containing a Window Definition Block. (See Section 5.5.2.2.)

ids Directive status

Macro Call

```
CRAW$   wdb
```

wdb Window Definition Block address

Macro Expansion

```
CRAW$   WDBADR
.BYTE   117.,2    ;CRAW$ MACRO DIC, DPB SIZE=2 WORDS
.WORD   WDBADR    ;WDB ADDRESS
```

Definition Block Parameters

Table 8-2 shows the Window Definition Block parameters for this directive.

Table 8-2: Window Definition Block Parameters for CRAW$

Array Element	Offset	Description
Input Parameters		
iwdb(1) bits 8-15	W.NAPR	Base APR of the address window to be created.
iwdb(3)	W.NSIZ	Desired size, in 32-word blocks, of the address window.
iwdb(4)	W.NRID	ID of the region to which the new window is to be mapped, or 0 for task region (to be specified only if WS.MAP=1).
iwdb(5)	W.NOFF	Offset in 32-word blocks from the start of the region at which the window is to start mapping (to be specified only if WS.MAP=1). **NOTE** If WS.64B in the window status word equals 0, the value specified must be a multiple of 8.

Array Element	Offset	Description
iwdb(6)	W.NLEN	Length in 32-word blocks to be mapped, or 0 if the length is to default to either the size of the window or the space remaining in the region, whichever is smaller (to be specified only if WS.MAP=1).
iwdb(7)	W.NSTS	Bit settings* in the window status word:

Bit	Definition
WS.MAP	1 if the new window is to be mapped
WS.WRT	1 if the mapping assignment is to occur with write access
WS.64B	0 for 256-word (512-byte) alignment, or 1 for 32-word (64-byte) alignment

Output Parameters

iwdb(1) bits 0-7	W.NID	ID assigned to the window.
iwdb(2)	W.NBAS	Virtual address base of the new window.
iwdb(6)	W.NLEN	Length, in 32-word blocks, actually mapped by the window.
iwdb(7)	W.NSTS	Bit settings* in the window status word:

* If you are a high-level language programmer, see Section 5.5.2 to determine the bit values represented by the symbolic names described.

Array Element	Offset	Description	
		Bit	**Definition (if bit=1)**
		WS.CRW	Address window successfully created
		WS.UNM	At least one window was unmapped
		WS.ELW	At least one window was eliminated
		WS.RRF	Reference was successfully received
		WS.RES	Map only if resident
		WS.NAT	Create attachment descriptor only if necessary (for Send By Reference directives)
		Bit	**Definition (if bit=1)**
		WS.64B	Define the task's permitted alignment boundaries: - 0 for 256-word (512-byte) alignment, 1 for 32-word (64-byte) alignment
		WS.MAP	Window is to be mapped
		WS.RCX	Exit if no references to receive
		WS.DEL	Send with delete access
		WS.EXT	Send with extend access
		WS.WRT	Send with write access or map with write access
		WS.RED	Send with read access

Local Symbol Definitions

C.RABA	Window Definition Block address (2)

DSW Return Codes

IS.SUC	Successful completion.
IE.PRI	Requested access denied at mapping stage.
IE.NVR	Invalid region ID.
IE.ALG	Task specified either an invalid base APR and window size combination, or an invalid region offset and length combination in the mapping assignment; or WS.64B = 0 and the value of W.NOFF is not a multiple of 8.
IE.WOV	No window blocks available in task's header.
IE.ADP	Part of the DPB or WDB is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

8.14 CRRG$ - CREATE REGION

The Create Region directive creates a dynamic region in a system-controlled partition and optionally attaches it to the issuing task.

If RS.ATT is set in the region status word, the Executive attempts to attach the task to the newly created region. If no region name has been specified, the user's program must set RS.ATT. (See the description of the Attach Region directive.)

By default, the Executive marks a dynamically created region for deletion when the last task detaches from it. To override this default condition, set RS.NDL in the region status word as an input parameter. Be careful in deciding to override the delete-on-last-detach option. An error within a program can cause the system to lock by leaving no free space in a system-controlled partition.

If the region is not given a name, the Executive ignores the state of RS.NDL. All unnamed regions are deleted when the last task detaches from them.

Named regions are put in the Common Block Directory (CBD). However, memory is not allocated until the Executive maps a task to the region.

The Executive returns an error if there is not enough space to accommodate the region in the specified partition. (See Notes.)

High-Level Language Call

```
CALL CRRG (irdb[,ids])
```

irdb	An eight-word integer array containing a Region Definition Block (see Section 7.5.1.2)
ids	Directive status

Macro Call

```
CRRG$   rdb
```

rdb	Region Definition Block address

Macro Expansion

```
CRRG$   RDBADR
.BYTE   55.,2       ;CRRG$ MACRO DIC, DPB SIZE = 2 WORDS
.WORD   RDBADR      ;RDB ADDRESS
```

Definition Block Parameters

Table 8-3 shows the Region Definition Block parameters for this directive.

Table 8-3: Region Definition Block Parameters for CRRG$

Array Element	Offset	Description
Input Parameters		
irdb(2)	R.GSIZ	Size, in 32-word blocks, of the region to be created.
irdb(3)(4)	R.GNAM	Name of the region to be created, or 0 for no name.
irdb(5)(6)	R.GPAR	Name of the system-controlled partition in which the region is to be allocated, or 0 for the partition in which the task is running.
irdb(7)	R.GSTS	Bit settings* in the region status word:

* If you are a high-level language programmer, see Section 5.5.1 to determine the bit values represented by the symbolic names described.

Array Element	Offset	Description
		Bit **Definition (if bit=1)**
		RS.CRR Region was successfully created.
		RS.UNM At least one window was unmapped on a detach.
		RS.MDL Mark region for deletion on last detach.
		RS.NDL The region should not be deleted on last detach.
		Bit **Definition (if bit=1)**
		RS.ATT Created region should be attached.
		RS.NEX Created region cannot be extended.
		RS.RED Read access is desired on attach.
		RS.WRT Write access is desired on attach.
		RS.EXT Extend access is desired on attach.
		RS.DEL Delete access is desired on attach.
irdb(8)	R.GPRO	Protection word for the region (DEWR,DEWR,DEWR,DEWR).
Output Parameters		
irdb(1)	R.GID	ID assigned to the created region (returned if RS.ATT=1).
irdb(2)	R.GSIZ	Size in 32-word blocks of the attached region (returned if RS.ATT=1).
irdb(7)	R.GSTS	Bit settings in the region status word:

Array Element	Offset	Description	
		Bit	**Definition**
		RS.CRR	1 if the region was successfully created.

Local Symbol Definitions

C.RRBA	Region Definition Block address (2)

DSW Return Codes

IS.SUC	Successful completion.
IE.UPN	A Partition Control Block (PCB) or an attachment descriptor could not be allocated, or the partition was not large enough to accommodate the region, or there is currently not enough contiguous space in the partition to accommodate the region.
IE.HWR	The directive failed in the attachment stage because a region parity error was detected.
IE.PRI	Attach failed because desired access was not allowed.
IE.PNS	Specified partition in which the region was to be allocated does not exist; or no region name was specified and RS.ATT = 0.
IE.ADP	Part of the DPB or RDB is out of issuing task's address space.
IE.SDP	DIC or RDB size is invalid.

Notes

1. The Executive does not return an error if the named region already exists. In this case, the Executive clears the RS.CRR bit in the status word R.GSTS. If RS.ATT has been set, the Executive attempts to attach the already existing named region to the issuing task.

2. The protection word (see R.GPRO above) has the same format as that of the file system protection word. There are four categories, and the access for each category is coded into four bits. From low order to high order, the categories follow this order: system, owner, group, world. The access code bits within each category are arranged (from low order to high order) as follows: read, write, extend, delete. A bit that is set indicates that the corresponding access is denied.

   The issuing task's UIC is the created region's owner UIC.

   In order to prevent creation of common blocks that are not easily deleted, the system and owner categories are always forced to have delete access, regardless of the value actually specified in the protection word.

8.15 CRVT$ - CREATE VIRTUAL TERMINAL

The Create Virtual Terminal directive creates a virtual terminal for use by a parent task in communicating with its offspring tasks. When the offspring task issues a read or write to its TI: terminal, the request is sent to the parent task through the virtual terminal.

This directive creates a Device Control Block (DCB) and a Unit Control Block (UCB) for each virtual terminal unit, and links the unit to the device list. Each newly created virtual terminal unit is assigned the lowest available virtual terminal unit number.

Only one copy of the Status Control Block (SCB) is required. The data structure for Virtual Terminal Unit 0 (VT0:) is used as a template for these dynamically created data structures. Thus, VT0: is never assigned as a virtual terminal unit number.

On successful completion of this directive, the assigned VT: unit number is returned in the DSW with the Carry bit clear. The task must save this number if this virtual terminal is referred to in another directive.

A rundown count is maintained in the issuing task's TCB to indicate the total (current) number of virtual terminals the task has created and the number of connected offspring tasks. This count is reduced when an Eliminate Virtual Terminal directive is issued specifying this VT: unit.

The input and output AST routines for the virtual terminal unit are entered with the following three words on the stack:

SP+04	Third parameter word (VFC) of offspring request
SP+02	Byte count of offspring request
SP+00	Virtual terminal unit number (low byte); I/O subfunction code of offspring request (high byte)

The attach and detach AST routine is entered with the following three words on the stack:

SP+04	Second word of offspring task name (0 if detach AST)
SP+02	First word of offspring task name (0 if detach AST)
SP+00	Virtual terminal unit number (low byte); I/O subfunction code of offspring request (high byte)

Note that the detach AST routine is entered with 0 in both task name words on the stack. The AST routine must remove the three words from the stack before it issues an AST Service Exit directive.

Parent tasks can service each offspring input or output request with a corresponding output or input request to the correct virtual device unit. For example, where MACRO-11 has been activated as an offspring task with a TI: of VT3:, the following apply:

- MACRO-11 issues an IO.RVB or IO.RLB to TI: for its first input line. The virtual terminal driver queues the read request internally and effects an AST in the parent at the virtual address "iast" with the unit number 3 and the byte count from MACRO-11's I/O request on the stack.

- In its AST routine, the parent answers the offspring's read request in a QIO directive to a LUN assigned to VT3: with an IO.WVB or IO.WLB function, a byte count of the line, and the status to be returned (such as IS.CR).

- The virtual terminal driver reads the line from the parent's buffer, writes the line to MACRO-11's buffer, and then signals I/O completion for both I/O requests.

- Similarly, if MACRO-11 needs to print an error message, it does so with an IO.WVB or IO.WLB to TI:. The virtual terminal driver queues the write request internally and effects an AST in the parent at the virtual address "oast" with the unit number 3, the byte count, and the VFC from MACRO-11's I/O request on the stack.

- In its output AST routine, the parent issues an IO.RVB or IO.RLB to retrieve the line by means of the virtual terminal driver. The parent may then output this line to a log file, for example. The third word on the AST stack in the parent's output AST routine is the vertical format character, telling the parent what type of carriage control is expected for the output line. This word would be ignored in the input AST routine.

The virtual terminal driver does not interpret or modify transferred bytes, I/O subfunction codes, or vertical format characters. However, this driver does automatically truncate offspring I/O requests to the maximum byte count specified in the "mlen" parameter, without notifying either the parent or offspring task. The actual number of bytes transferred on each request is equal to the smaller of the byte counts specified in the offspring and parent I/O requests.

The total number of bytes transferred is returned in the corresponding I/O status blocks. Note that offspring tasks can receive "mlen" in the fourth characteristics word when a Get LUN Information directive is issued.

Intermediate buffering in secondary pool, when enabled by the parent task, is performed on offspring input and output requests when the offspring task is checkpointable. Offspring tasks, therefore, can be stopped and checkpointed.

If the parent task is stopped and checkpointed when the offspring task issues an I/O request, the resulting AST brings the parent task to an unstopped state from which it can return to memory to service the I/O request. Upon exit from the AST routine, the parent task is again stopped. This mode of operation allows the parent and offspring tasks to share the same physical memory, even while the parent task services the terminal I/O requests for the offspring task.

Whenever the virtual terminal driver determines that it should not use intermediate buffering, offspring tasks are locked in memory when I/O requests are issued, and transfers occur directly between parent and offspring buffers.

The intermediate buffering of offspring I/O requests can normally be enabled and disabled by the parent task with the IO.STC function, as described below. An exception exists for virtual terminals created with a "mlen" parameter greater than the system-wide maximum.

If a Create Virtual Terminal directive is specified with a "mlen" parameter greater than the system-wide maximum, the parameter is accepted, but intermediate buffering for the created virtual terminal unit is automatically disabled. Furthermore, intermediate buffering for that unit cannot be enabled by the parent task with the IO.STC function.

Parent tasks specify the first word of the I/O completion status for the offspring request in the third word of the QIO DPB parameter list. For example, consider an offspring input request for 10 characters or more that is honored with a write logical of 10 characters and IS.CR in the third parameter word. The second word of the I/O status would be set to 10, and 10 characters would be transferred.

Another example is when a parent task issues a read request to satisfy a write request issued by the offspring task. To notify the offspring task that its write request was satisfied, the parent task would specify IS.SUC in the third parameter word.

A special I/O function, IO.STC, returns status to an offspring task without a data transfer. The parameter word format for the IO.STC function is as follows:

- Word 0 with bit 0 set indicates that status is being returned.
- Word 0 with bit 1 clear, if the virtual terminal is in full-duplex mode, indicates that status is being returned for an offspring read request.
- Word 0 with bit 1 set, if the virtual terminal is in full-duplex mode, indicates that status is being returned for an offspring write request.
- Word 1 is the second word of I/O return status.
- Word 2 is the first word of I/O return status.

NOTE

If the virtual terminal is in half-duplex mode, bit 1 of word 0 is ignored.

The status words are reversed in order to be similar to the format in which status must be passed back in a parent read or write function to an offspring task. The IO.STC function must be used to return status when no transfer is desired, because a byte count of 0 is not allowed in an IO.RLB or IO.WLB (read logical block and write logical block operations, respectively). For example, IE.EOF (write end-of-file tape mark) is normally returned with IO.STC.

Note that it is important to specify an I/O completion status for all parent read and write requests that satisfy corresponding requests from the offspring task. If a return status is not specified, it defaults to zero. A zero indicates that the I/O is still pending (IS.PND). This causes the offspring task to hang if it examines the I/O status block to determine whether the I/O is completed.

In addition to returning status, the IO.STC function has an additional purpose. It can enable or disable intermediate buffering of I/O requests. (Note that a task cannot perform both IO.STC functions in the same I/O request.) If bit 0 of the first parameter word in IO.STC is clear, bit 1 in this word is interpreted as a disable buffering flag:

- If bit 0 is clear and bit 1 is set, intermediate buffering of offspring I/O is disabled.

- If bit 0 is clear and bit 1 is clear, buffering is enabled.

Buffering cannot be enabled on a virtual terminal unit that has been created with an "mlen" parameter greater than the system-wide maximum. An attempt to do both results in an error return of IE.IFC.

The only tasks that can assign LUNs to a virtual terminal unit are:

- The task that created the virtual terminal unit

- That task's offspring task, whose TI: is the virtual terminal unit

Attachment of a virtual terminal unit by an offspring task prevents the dequeuing of I/O requests to that unit from other offspring tasks. Parent I/O requests are always serviced.

Both parent and offspring tasks can specify the I/O functions IO.GTS, SF.GMC, and SF.SMC. However, SF.GMC and SF.SMC support only a limited number of terminal characteristics for virtual terminals. Please refer to Chapter 12 for a list of valid characteristics.

Note that the parent task is not notified when the offspring issues any of the above directives.

When an offspring task issues a read-with-prompt request (IO.RPR), the virtual terminal driver separates the request into an IO.WLB request and an IO.RLB request. The parent task cannot issue an IO.RPR.

When a virtual terminal is in half-duplex mode, the virtual terminal driver handles only one offspring request at a time. For example, if the offspring task issues a read request and then issues a write request without waiting for the read to be completed, the driver queues the write request to be processed when the read is completed.

The parent task can issue an SF.SMC function to set the virtual terminal to full-duplex mode. In full-duplex mode, the write request in the previous example would be processed even if the previous read was not yet completed. If the parent task is at AST state, it will not receive notification of the I/O request.

Both parent and offspring tasks can issue an SF.GMC request to determine the mode of the virtual terminal. However, only the parent task can change the mode (using SF.SMC).

High-Level Language Call

```
CALL CRVT ([iiast],[ioast],[iaast],[imlen],iparm[,ids])
```

iiast — AST address where input requests from offspring tasks are serviced

ioast — AST address where output requests from offspring tasks are serviced

iaast — AST address where the parent task can be notified of the completion of successful offspring attach and detach requests to the virtual terminal unit

NOTE

At least one of the above optional parameters should be specified. Otherwise, the virtual terminal created is treated as the null device.

imlen — Maximum buffer length allowed for offspring I/O requests

iparm — Address of three-word buffer to receive information from the stack when an AST occurs

ids — Integer to receive the Directive Status Word containing the virtual terminal number

Macro Call

```
CRVT$   [iast],[oast],[aast],[mlen]
```

iast — AST address at which input requests from offspring tasks are serviced. If iast=0, offspring input requests are rejected with IE.IFC returned.

oast — AST address at which output requests from offspring tasks are serviced. If oast=0, offspring output requests are rejected with IE.IFC returned.

aast — AST address at which the parent task can be notified of the completion of successful offspring attach and detach requests to the virtual terminal unit. If aast=0, no notification of offspring attach/detach is returned to the parent task.

NOTE

At least one of the above optional parameters should be specified. Otherwise, the virtual terminal created is treated as the null device.

mlen — Maximum buffer length (in bytes) allowed for offspring I/O requests.

NOTE

If the value of the mlen parameter is -1, the system creates a special kind of null virtual terminal. See Section 12.6 for details.

Macro Expansion

```
CRVT$   IASTRU,OASTRU,PAST,20.
.BYTE   149.,5      ;CRVT$ MACRO DIC, DPB SIZE = 5 WORDS
.WORD   IASTRU      ;INPUT REQUEST AST ROUTINE ADDRESS
.WORD   OASTRU      ;OUTPUT REQUEST AST ROUTINE ADDRESS
.WORD   PAST        ;SUCCESSFUL VT ATTACH NOTIFICATION
                    ;AST ROUTINE ADDRESS
.WORD   20.         ;MAXIMUM BUFFER LENGTH = 20 (DECIMAL)
                    ;BYTES
```

Local Symbol Definitions

C.RVIA — Input request AST routine address (2)

C.RVOA — Output request AST routine address (2)

C.RVAA — VT attach notification AST routine address (2)

C.RVML — Maximum buffer length (2)

DSW Return Codes

unit	Successful completion results in the return of the unit number of the created virtual terminal unit with the C bit clear.
IE.UPN	Insufficient dynamic memory to allocate the virtual terminal device unit data structures.
IE.HWR	Virtual terminal device driver not resident.
IE.ADP	Part of the DPB is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

8.16 CSRQ$ - CANCEL TIME-BASED INITIATION REQUESTS

The Cancel Time-Based Initiation Requests directive instructs the system to cancel all time-synchronized initiation requests for a specified task, regardless of the source of each request. These requests result from a Run directive.

High-Level Language Call

```
CALL CANALL (tsk[,ids])
```

tsk	Task name
ids	Directive status

Macro Call

```
CSRQ$   tt
```

tt	Scheduled (target) task name

Macro Expansion

```
CSRQ$   ALPHA
.BYTE   25.,3           ;CSRQ$ MACRO DIC, DPB SIZE=3 WORDS
.RAD50  /ALPHA/         ;TASK "ALPHA"
```

Local Symbol Definitions

C.SRTN	Target task name (4)

DSW Return Codes

IS.SUC	Successful completion.
IE.INS	Task is not installed.
IE.PRI	The issuing task is not privileged and is attempting to cancel requests made by another task.
IE.ADP	Part of the DPB is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

Note

If you specify an error routine address when using the $C or $S macro form, you must include a null argument. For example:

```
        CSRQ$S      #TNAME,,ERR     ;CANCEL REQUESTS FOR "ALPHA"
          .           .
          .           .
          .           .
        TNAME: .RAD50  /ALPHA/
```

8.17 DECL$S - DECLARE SIGNIFICANT EVENT

The Declare Significant Event directive instructs the system to declare a significant event.

Declaration of a significant event causes the Executive to scan the Active Task List from the beginning, searching for the highest priority task that is ready to run. Use this directive with discretion to avoid excessive scanning overhead.

The $S form of the macro is recommended because this directive requires only a one-word DPB.

High-Level Language Call

```
CALL DECLAR ([,ids])
```

ids	Directive status

Macro Call

```
DECL$S  [,err]
```

err	Error routine address

Macro Expansion

```
DECL$S  ,ERR            ;ONE ARGUMENT IGNORED
MOV     (PC)+,-(SP)     ;PUSH DPB ONTO THE STACK
.BYTE   35.,1           ;DECL$S MACRO DIC, DPB SIZE=1 WORD
EMT     377             ;TRAP TO THE EXECUTIVE
BCC     .+6             ;BRANCH IF DIRECTIVE SUCCESSFUL
JSR     PC,ERR          ;OTHERWISE, CALL ROUTINE "ERR"
```

Local Symbol Definitions

None

DSW Return Codes

IS.SUC	Successful completion.
IE.ADP	Part of the DPB is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

8.18 DLOG$ - DELETE LOGICAL NAME

The Delete Logical Name directive deletes a logical name from the logical name table and returns to the system the resources used by that logical name. You should delete logical names when they are no longer needed. If you do not specify the the logical name string buffer address, DLOG$ deletes all of the logical names with the specified modifier in the specified logical name table.

High-Level Language Call

```
CALL DELLOG  (mod,itbnum,lns,lnssz,idsw)
```

mod — The modifier of the logical name within a table. See Section 2.2 for details.

itbnum — The logical name table number:

```
system (LT.SYS) = 0
session (LT.SES) = 4
user (LT.USR) = 2
```

lns — Character array containing the logical name string

lnssz — Size (in bytes) of the logical name string. If lnssz equals 0, then the operation deletes all logicals in the specified table with the specified modifier.

idsw — Integer to receive the Directive Status Word

Macro Call

```
DLOG$  mod,tbnum,lns,lnssz
```

mod — The modifier of the logical name within a table. See Section 2.2 for details.

tbnum — The logical name table number:

```
system (LT.SYS) = 0
session (LT.SES) = 4
user (LT.USR) = 2
```

lns — Character array containing the logical name string

lnssz — Size (in bytes) of the logical name string. If lnssz equals 0, then the operation deletes all

logicals in the specified table with the specified modifier.

Macro Expansion

```
DLOG$   mod,tbnum,lns,lnssz
.BYTE   207.,5          ;DLOG$ MACRO DIC, DPB SIZE = 5 WORDS
.BYTE   2               ;SUBFUNCTION CODE FOR DELETION
.BYTE   MOD             ;LOGICAL NAME MODIFIER
.BYTE   TBNUM           ;LOGICAL NAME TABLE NUMBER
.BYTE   0               ;RESERVED FOR FUTURE USE
.WORD   LNS             ;ADDRESS OF THE LOGICAL NAME BUFFER
.WORD   LNSSZ           ;BYTE COUNT, LOGICAL NAME STRING
```

Local Symbol Definitions

D.LFUN	Subfunction (1)
D.LLNS	Address of logical name string (2)
D.LLSZ	Byte count of logical name string (2)
D.LMOD	Logical name modifier (1)
D.LTBL	Logical table number (1)

DSW Return Codes

IS.SUC	Successful completion.
IE.LNF	The specified logical name string was not found.
IE.IBS	The length of the logical or equivalence string is invalid. Each string length must be greater than 0 but not greater than 255 (decimal) characters.
IE.ITN	Invalid table number specified.
IE.ADP	Part of the DPB or user buffer is out of the issuing task's address space, or the user does not have proper access to that region.
IE.SDP	DIC or DPB size is invalid.

8.19 DSAR$S/IHAR$S - DISABLE/INHIBIT AST RECOGNITION

The Disable (or Inhibit) AST Recognition directive instructs the system to disable recognition of ASTs for the issuing task. The ASTs are queued as they occur and are effected when the task reenables AST recognition. There is an implied disable AST recognition directive whenever an AST service routine is executing. When a task's execution is started, AST recognition is enabled. (See Notes.)

High-Level Language Call

```
CALL DSASTR  [(ids)]
   or
CALL INASTR  [(ids)]
```

ids	Directive status

Macro Call

```
DSAR$S  [err]
```

err	Error routine address

Macro Expansion

```
DSAR$S  ERR
MOV     (PC)+,-(SP)     ;PUSH DPB ONTO THE STACK
.BYTE   99.,1           ;DSAR$S MACRO DIC, DPB SIZE=1 WORD
EMT     377             ;TRAP TO THE EXECUTIVE
BCC     .+6             ;BRANCH IF DIRECTIVE SUCCESSFUL
JSR     PC,ERR          ;OTHERWISE, CALL ROUTINE "ERR"
```

Local Symbol Definitions

None

DSW Return Codes

IS.SUC	Successful completion.
IE.ITS	AST recognition is already disabled.
IE.ADP	Part of the DPB is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

Notes

1. This directive disables only the recognition of ASTs; the Executive still queues the ASTs. They are queued FIFO and will occur in that order when the task reenables AST recognition.

2. Because this directive requires only a one-word DPB, the $S form of the macro is recommended. It requires less space than, and executes with the same speed as, the DIR$ macro.

8.20 DSCP$S - DISABLE CHECKPOINTING

The Disable Checkpointing directive instructs the system to disable checkpointing for a task that has been installed as a checkpointable task. Only the affected task can issue this directive. A task cannot disable the ability of another task to be checkpointed.

High-Level Language Call

```
CALL DISCKP [(ids)]
```

ids — Directive status

Macro Call

```
DSCP$S  [err]
```

err — Error routine address

Macro Expansion

```
DSCP$S  ERR
MOV     (PC)+,-(SP)     ;PUSH DPB ONTO THE STACK
.BYTE   95.,1           ;DSCP$S MACRO DIC, DPB SIZE=1 WORD
EMT     377             ;TRAP TO THE EXECUTIVE
BCC     .+6             ;BRANCH IF DIRECTIVE SUCCESSFUL
JSR     PC,ERR          ;OTHERWISE, CALL ROUTINE "ERR"
```

Local Symbol Definitions

None

DSW Return Codes

IS.SUC	Successful completion.
IE.ITS	Task checkpointing is already disabled.
IE.CKP	Issuing task is not checkpointable.
IE.ADP	Part of the DPB is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

Notes

1. When a checkpointable task's execution is started, checkpointing is enabled (that is, the task can be checkpointed).

2. Because this directive requires only a one-word DPB, the $S form of the macro is recommended. It requires less space than, and executes with the same speed as, the DIR$ macro.

8.21 DTRG$ - DETACH REGION

The Detach Region directive detaches the issuing task from a specified, previously attached region. Any of the task's windows that are currently mapped to the region are automatically unmapped.

If RS.MDL is set in the region status word when the directive is issued, the task marks the region for deletion on the last detach. A task must be attached with delete access to mark a region for deletion.

High-Level Language Call

```
CALL DTRG (irdb[,ids])
```

irdb	An eight-word integer array containing a Region Definition Block (see Section 5.5.1.2)
ids	Directive status

Macro Call

```
DTRG$     rdb
```

rdb	Region Definition Block address

Macro Expansion

```
DTRG$   RDBADR
.BYTE   59.,2     ;DTRG$ MACRO DIC, DPB SIZE=2 WORDS
.WORD   RDBADR    ;RDB ADDRESS
```

Definition Block Parameters

Table 8-4 shows the Region Definition Block parameters for this directive.

Local Symbol Definitions

D.TRBA	Region Definition Block address (2)

Table 8-4: Region Definition Block Parameters for DTRG$

Array Element	Offset	Description
Input Parameters		
irdb(1)	R.GID	ID of the region to be detached
irdb(7)	R.GSTS	Bit settings* in the region status word: **Bit** **Definition** RS.MDL 1 if the region should be marked for deletion when the last task detaches from it
Output Parameters		
irdb(7)	R.GSTS	Bit settings* in the region status word: **Bit** **Definition** RS.UNM 1 if any windows were unmapped

DSW Return Codes

IS.SUC	Successful completion.
IE.PRI	The task, which is not attached with delete access, has attempted to mark the region for deletion on the last detach, or the task has outstanding I/O.
IE.NVR	The task specified an invalid region ID or attempted to detach region 0 (its own task region).

* If you are a high-level language programmer, see Section 5.5.1 to determine the bit values represented by the symbolic names described.

IE.ADP	Part of the DPD or RDB is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

8.22 ELAW$ - ELIMINATE ADDRESS WINDOW

The Eliminate Address Window directive deletes an existing address window, unmapping it first if necessary. Subsequent use of the eliminated window's ID is invalid.

High-Level Language Call

```
CALL ELAW (iwdb[,ids])
```

iwdb	An eight-word integer array containing a Window Definition Block (see Section 5.5.2.2)
ids	Directive status

Macro Call

```
ELAW$   wdb
```

wdb	Window Definition Block address

Macro Expansion

```
ELAW$   WDBADR
.BYTE   119.,2          ;ELAW$ MACRO DIC, DPB SIZE=2 WORDS
.WORD   WDBADR          ;WDB ADDRESS
```

Definition Block Parameters

Table 8-5 shows the Window Definition Block parameters for this directive.

Local Symbol Definitions

E.LABA	Window Definition Block address (2)

DSW Return Codes

IS.SUC	Successful completion.
IE.NVW	Invalid address window ID.
IE.ADP	Part of the DPB or WDB is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

Table 8-5: Window Definition Block Parameters for ELAW$

Array Element	Offset	Description
Input Parameters		
iwdb(1) bits 0-7	W.NID	ID of the address window to be eliminated
Output Parameters		
iwdb(7)	W.NSTS	Bit settings* in the window status word: **Bit** — **Definition** WS.ELW — 1 if the address window was successfully eliminated WS.UNM — 1 if the address window was unmapped

* If you are a high-level language programmer, see Section 5.5.2 to determine the bit values represented by the symbolic names described.

8.23 ELVT$ - ELIMINATE VIRTUAL TERMINAL

The Eliminate Virtual Terminal directive causes the specified virtual terminal unit data structures to be marked for deallocation, and eventually, to be unlinked from the device list and deallocated. This directive can only be issued by the task that created the virtual terminal device unit. All active nonprivileged tasks are aborted whose TI: device units are the virtual terminal being deallocated. TKTN messages reporting the abortion of these tasks in this instance are directed to CO:. All LUNs assigned by the issuing task, or by any offspring task being aborted, are deassigned.

A rundown count is maintained in the TCB of each parent task. This count reflects the total number of outstanding virtual terminal units the task has created, plus the number of connected (offspring) tasks. A series of ELVT$ directives are issued when a parent task, which has not eliminated virtual terminals it has created, exits. The virtual terminal data structures continue to exist until the last task whose TI: is the virtual terminal unit exits and until all CLI commands for that unit have been processed.

High-Level Language Call

```
CALL ELVT (iunum[,ids])
```

iunum	Virtual terminal unit number
ids	Integer to receive the Directive Status Word

Macro Call

```
ELVT$   unum
```

unum	Unit number of the virtual terminal to be eliminated. The task must provide this parameter after the virtual terminal is created. (See Note.)

Macro Expansion

```
ELVT$   0
.BYTE   151.,2              ;ELVT$ MACRO DIC, DPB SIZE = 2
WORDS
.WORD   0                   ;VIRTUAL TERMINAL UNIT NUMBER
```

Local Symbol Definitions

E.LVNM	VT unit number (2)

DSW Return Codes

IS.SUC	Successful completion.
IE.IDU	The specified virtual terminal unit does not exist or it was not created by the issuing task.
IE.ADP	Part of the DPB is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

Note

The actual virtual terminal unit number is not known until after the virtual terminal is actually created (that is, until after successful completion of a Create Virtual Terminal directive). The Create Virtual Terminal directive DSW contains the actual virtual terminal unit number for use in the Eliminate Virtual Terminal directive. Thus, the task must save DSWs for all virtual terminals it creates, and later eliminate them using the Eliminate Virtual Terminal directive.

8.24 EMST$ - EMIT STATUS

The Emit Status directive returns the specified 16-bit quantity to the specified connected task. It possibly sets an event flag or declares an AST if previously specified by the connected task in a Send, Request And Connect, a Spawn, or a Connect directive. If the specified task is multiply connected to the task issuing this directive, the first (oldest) Offspring Control Block (OCB) in the queue is used to return status. If no task name is specified, this action is taken for all tasks that are connected to the issuing task at that time. In any case, whenever status is emitted to one or more tasks, those tasks no longer remain connected to the task issuing the Emit Status directive.

High-Level Language Call

```
CALL EMST ([rtname],status[,ids])
```

rtname	Name of a task connected to the issuing task to which the status is emitted
status	A 16-bit quantity returned to the connected task
ids	Integer to receive the Directive Status Word

Macro Call

```
EMST$   [tname],status
```

tname	Name of a task connected to the issuing task to which the status is emitted
status	16-bit quantity returned to the connected task

Macro Expansion

```
EMST$     ALPHA,STWD
.BYTE     147.,4        ;EMST$ MACRO DIC, DPB SIZE=4 WORDS
.RAD50    ALPHA         ;NAME OF CONNECTED TASK TO RECEIVE
STATUS
.WORD     STWD          ;VALUE OF STATUS TO BE RETURNED
```

Local Symbol Definitions

```
E.MSTN      Task name (4)

E.MSST      Status to be returned (2)
```

DSW Return Codes

IS.SUC	Successful completion.
IE.ITS	The specified task is not connected to the issuing task.
IE.ADP	Part of the DPB is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

8.25 ENAR$S - ENABLE AST RECOGNITION

The Enable AST Recognition directive instructs the system to recognize ASTs for the issuing task; that is, the directive nullifies a Disable AST Recognition directive. ASTs that were queued while recognition was disabled are effected at issuance. When a task's execution is started, AST recognition is enabled.

High-Level Language Call

```
CALL ENASTR  [(ids)]
```

ids Directive status

Macro Call

```
ENAR$S  [err]
```

err Error routine address

Macro Expansion

```
ENAR$S  ERR
MOV     (PC)+,-(SP)     ;PUSH DPB ONTO THE STACK
.BYTE   101.,1          ;ENAR$S MACRO DIC, DPB SIZE=1 WORD
EMT     377             ;TRAP TO THE EXECUTIVE
BCC     .+6             ;BRANCH IF DIRECTIVE SUCCESSFUL
JSR     PC,ERR          ;OTHERWISE, CALL ROUTINE "ERR"
```

Local Symbol Definitions

None

DSW Return Codes

IS.SUC Successful completion.

IE.ITS AST recognition is not disabled.

IE.ADP Part of the DPB is out of the issuing task's address space.

IE.SDP DIC or DPB size is invalid.

Note

Because this directive requires only a one-word DPB, the $S form of the macro is recommended. It requires less space and executes with the same speed as that of the DIR$ macro.

8.26 ENCP$S - ENABLE CHECKPOINTING

The Enable Checkpointing directive instructs the system to make the issuing task checkpointable after its checkpointing has been disabled for that task; that is, the directive nullifies a DSCP$S directive. This directive cannot be used to enable checkpointing of a task that was built with checkpointing disabled.

High-Level Language Call

```
CALL ENACKP [(ids)]
```

ids Directive status

Macro Call

```
ENCP$S  [err]
```

err Error routine address

Macro Expansion

```
ENCP$S  ERR
MOV     (PC)+,-(SP)     ;PUSH DPB ONTO THE STACK
.BYTE   97.,1           ;ENCP$S MACRO DIC, DPB SIZE=1 WORD
EMT     377             ;TRAP TO THE EXECUTIVE
BCC     .+6             ;BRANCH IF DIRECTIVE SUCCESSFUL
JSR     PC,ERR          ;OTHERWISE, CALL ROUTINE "ERR"
```

Local Symbol Definitions

None

DSW Return Codes

IS.SUC	Successful completion.
IE.ITS	Checkpointing is not disabled or task is connected to an interrupt vector.
IE.ADP	Part of the DPB is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

Note

Because this directive requires only a one-word DPB, the $S form of the macro is recommended. It requires less space than, and executes with the same speed as, the DIR$ macro.

8.27 EXIF$ - EXIT IF

The Exit If directive instructs the system to terminate the execution of the issuing task if, and only if, an indicated event flag is not set. The Executive returns control to the issuing task if the specified event flag is set. See Notes.

High-Level Language Call

```
CALL EXITIF (efn[,ids])
```

efn — Event flag number

ids — Directive status

Macro Call

```
EXIF$   efn
```

efn — Event flag number

Macro Expansion

```
EXIF$   52.
.BYTE   53.,2           ;EXIF$ MACRO DIC, DPB SIZE=2 WORDS
.WORD   52.             ;EVENT FLAG NUMBER 52.
```

Local Symbol Definitions

E.XFEF — Event flag number (2)

DSW Return Codes

IS.SET — Indicated EFN set; task did not exit.

IE.IEF — Invalid event flag number (EFN<1 or EFN>64).

IE.ADP — Part of the DPB is out of the issuing task's address space.

IE.SDP — DIC or DPB size is invalid.

Notes

1. The Exit If directive is useful to avoid a possible race condition that can occur between two tasks communicating by means of the Send and Receive directives. The race condition occurs when one task executes a Receive directive and finds its receive queue empty; but before the task can exit, the other task sends it a message. The message is lost because

the Executive flushed the receiver task's receive queue when it decided to exit. This condition can be avoided if the sending task specifies a common event flag in the Send directive and the receiving task executes an Exit If specifying the same common event flag. If the event flag is set, the Exit If directive returns control to the issuing task, signaling that something has been sent.

2. A high-level language program that issues the Exit If call must first close all files by issuing Close calls. To avoid the time overhead involved in closing and reopening files, the task should first issue the appropriate test or Clear Event Flag directive. If the Directive Status Word indicates that the flag was not set, then the task can close all files and issue the call to Exit If.

3. On Exit, the Executive frees task resources. In particular, the Executive does the following:

   - Detaches all attached devices
   - Flushes the AST queue and despecifies all specified ASTs
   - Flushes the receive and receive-by-reference queues
   - Flushes the clock queue for any outstanding Mark Time requests for the task
   - Closes all open files (files open for write access are locked)
   - Detaches all attached regions, except in the case of a fixed task
   - Runs down the task's I/O
   - Disconnects from interrupt vectors
   - Breaks the connection with any offspring tasks
   - Returns a success status (EX$SUC) to any parent tasks
   - Frees the task's memory if the exiting task was not fixed

4. If the task exits, the Executive declares a significant event.

8.28 EXIT$S - TASK EXIT

The Task Exit directive instructs the system to terminate the execution of the issuing task.

High-Level Language Call

See Note 5 below.

Macro Call

```
EXIT$S  [err]
```

err Error routine address

Macro Expansion

```
EXIT$S  ERR
MOV     (PC)+,-(SP)     ;PUSH DPB ONTO THE STACK
.BYTE   51.,1           ;EXIT$S MACRO DIC, DPB SIZE=1 WORD
EMT     377             ;TRAP TO THE EXECUTIVE
JSR     PC,ERR          ;CALL ROUTINE "ERR"
```

Local Symbol Definitions

None

DSW Return Codes

IE.ADP	Part of the DPB is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

Notes

1. A return to the task occurs if, and only if, the directive is rejected. Therefore, no Branch on Carry Clear instruction is generated if an error routine address is given, since the return will only occur with carry set.

2. Exit causes a significant event to be declared.

3. On Exit, the Executive frees task resources. In particular, the Executive:

   - Detaches all attached devices

- Flushes the AST queue and despecifies all specified ASTs
- Flushes the receive and receive-by-reference queues
- Flushes the clock queue for all outstanding Mark Time requests for the task
- Closes all open files (files open for write access are locked)
- Detaches all attached regions, except in the case of a fixed task, where no detaching occurs
- Runs down the task's I/O
- Disconnects from interrupt vectors
- Breaks the connection with any offspring tasks
- Returns a success code (EX$SUC) to any parent task
- Frees the task's memory if the exiting task was not fixed

4. Because this directive requires only a one-word DPB, the $S form of the macro is recommended. It requires less space than, and executes with the same speed as, the DIR$ macro.

5. You can terminate FORTRAN tasks with the STOP statement or with CALL EXIT. CALL EXIT is a FORTRAN OTS routine that closes open files and performs other cleanup before it issues an EXIT$S directive (or an EXST$ directive in FORTRAN-77). FORTRAN tasks that terminate with the STOP statement result in a message being displayed on the task's TI:. This message includes task name (as it appears in the Active Task List), the statement causing the task to stop, and an optional character string specified in the STOP statement. Tasks that terminate with CALL EXIT do not display a termination message.

   For example, a FORTRAN task containing the following statement:

```
20        STOP 'THIS FORTRAN TASK'
```

   exits with the following message displayed on the task's TI: (TT0 in this example):

```
TT0       STOP THIS FORTRAN TASK
```

8.29 EXST$ - EXIT WITH STATUS

The Exit With Status directive causes the issuing task to exit, passing a 16-bit status back to all tasks connected (by the Spawn, Connect, or Send, Request And Connect directive). If the issuing task has no connected tasks, then the directive simply performs a Task Exit.

No format of the status word is enforced by the Executive; format conventions are a function of the cooperation between parent and offspring tasks. However, if an offspring task aborts for any reason, a status of EX$SEV is returned to the parent task. This value is interpreted as a "severe error" by batch processors. Furthermore, if a task performs a normal exit with other tasks connected to it, a status of EX$SUC (successful completion) is returned to all connected tasks.

High-Level Language Call

```
CALL EXST (istat)
```

istat	A 16-bit status value to be returned to parent task

Macro Call

```
EXST$   sts
```

sts	A 16-bit status value to be returned to parent task

Macro Expansion

```
EXST$     STWD
.BYTE     29.,2          ;EXST$ MACRO DIC, DPB SIZE=2 WORDS
.WORD     STWD           ;VALUE OF STATUS TO BE RETURNED
```

Local Symbol Definitions

E.XSTS	Value of status to be returned (2)

DSW Return Codes

No status is returned if the directive is successfully completed, since the directive causes the issuing task to exit.

IE.ADP	Part of the DPB is out of the issuing task's address space.

IE.SDP DIC or DPB size is invalid.

Notes

1. The Executive does the following to free a task's resources on Exit:

   - Detaches all attached devices
   - Flushes the AST queue and despecifies all specified ASTs
   - Flushes the Receive and Receive-by-reference queues
   - Flushes the clock queue for any outstanding Mark Time requests for the task
   - Closes all open files. (Files open for write access are locked.)
   - Detaches all attached regions except in the case of a fixed task
   - Runs down the task's I/O
   - Disconnects from interrupt vectors
   - Breaks the connection with any offspring tasks
   - Returns the specified exit status to any parent tasks
   - Frees the task's memory if the exiting task was not fixed

2. If the task exits, the Executive declares a significant event.

8.30 EXTK$ - EXTEND TASK

The Extend Task directive instructs the system to modify the size of the issuing task by a positive or negative increment of 32-word blocks. If the directive does not specify an increment value or specifies an increment value of zero, the Executive makes the issuing task's size equal to its installed size. The issuing task cannot have any outstanding I/O when it issues the directive. The task also must be checkpointable to increase its size; if necessary, the Executive checkpoints the task, and then returns the task to memory with its size modified as directed.

The Executive does not change any current mapping assignments if the task has memory-resident overlays. However, if the task does not have memory-resident overlays, the Executive attempts to modify, by the specified number of 32-word blocks, the mapping of the task to its task region.

If the issuing task is checkpointable but has no preallocated checkpoint space available, a positive increment can require dynamic memory and extra space in a checkpoint file sufficient to contain the task.

There are several constraints on the size to which a task can extend itself using the Extend directive:

- A task that does not have memory-resident overlays cannot extend itself beyond 32K minus 32 words.
- A task that has preallocated checkpoint space in its task image file cannot extend itself beyond its installed size.
- A task that has memory-resident overlays cannot reduce its size below the highest window in the task partition.

High-Level Language Call

```
CALL EXTTSK ([inc][,ids])
```

inc	A positive or negative number equal to the number of 32-word blocks by which the task size is extended or reduced
ids	Directive status

Macro Call

```
EXTK$   [inc]
```

inc — A positive or negative number equal to the number of 32-word blocks by which the task size is to be extended or reduced

Macro Expansion

```
EXTK$   40
.BYTE   89.,3           ;EXTK$ MACRO DIC, DPB SIZE=3 WORDS
.WORD   40              ;EXTEND INCREMENT, 40 (OCTAL) BLOCKS
(1K

;WORDS)
.WORD   0               ;RESERVED WORD
```

Local Symbol Definitions

E.XTIN — Extend increment (2)

DSW Return Codes

IS.SUC — Successful completion.

IE.UPN — Insufficient dynamic memory, or insufficient space in a checkpoint file.

IE.ITS — The issuing task is not running in a system controlled partition.

IE.ALG — The issuing task attempted to reduce its size to less than the size of its task header; or the task tried to increase its size beyond 32K words; or the task tried to increase its size to the extent that one virtual address window would overlap another; or the task has memory-resident overlays and it attempted to reduce its size below the highest window mapped to the task partition.

IE.RSU — Other tasks are attached to this task partition.

IE.IOP — I/O is in progress for this task partition.

IE.CKP — The issuing task is not checkpointable and specified a positive integer.

IE.NSW — Attempt to extend task size beyond installed size when checkpoint space is allocated in the task.

IE.ADP	Part of the DPB is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

8.31 FEAT$ - TEST FOR SPECIFIED SYSTEM FEATURE

The Features directive tests for the presence of a specific system software or hardware option (such as floating point support, or the presence of the Commercial Instruction Set).

High-Level Language Call

```
CALL FEAT (isym,ids)
```

isym	Symbol for the specified system feature. See Table 8-6 for a list of system feature symbols.
ids	Directive status

Macro Call

```
FEAT$   feat
```

feat	Symbol for the specified system feature. See Table 8-6 for a list of system feature symbols.

Macro Expansion

```
FEAT$   FE$POS
.BYTE   177.,2          ;FEAT$ MACRO DIC, DPB SIZE=2 WORDS
.WORD   FE$POS          ;FEATURE IDENTIFIER
```

Local Symbol Definitions

F.EAF	Feature identifier (2)

DSW Return Codes

IS.CLR	Successful completion; feature not present.
IS.SET	Successful completion; feature present.
IE.ADP	Part of the DPB is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

Note

If neither of the feature symbols HF$WS or HF$FS is set, then the system is stand-alone.

Table 8-6: System Feature Symbols

Symbol	Meaning
FE$ACN	System supports CPU accounting.
FE$AHR	Alternate header refresh area support.
FE$AST	System has AST support.
FE$CAL	Dynamic checkpoint space allocation.
FE$CEX	COM executive is loaded.
FE$CLI	Multiple CLI support.
FE$CRA	System spontaneously crashed; 1=yes (bit 33 decimal).
FE$CXD	Comm exec is deallocated, non-I/D only (bit 49 decimal).
FE$DAS	Kernel data space supported (bit 17 decimal).
FE$DPR	System has a separate directive partition.
FE$DRV	Loadable driver support.
FE$DYM	Dynamic memory allocation supported.
FE$EIS	System requires extended instruction set.
FE$EVT	System supports event trace feature.
FE$EXP	Extend task directive support.
FE$EXT	22-bit extended memory support (bit 1).
FE$EXV	Executive is supported to 20K.
FE$FDT	Full-duplex terminal driver support.
FE$GGF	Group global event flag support.
FE$IRR	INSTALL, RUN, and REMOVE support.
FE$LIB	Supervisor mode libraries support.

Symbol	Meaning
FE$LSI	Processor is an LSI-11.
FE$MP	System supports multiprocessing.
FE$MUP	Multiuser protection support.
FE$MXT	MCR exit after each command mode.
FE$NLG	Logins disabled--multiuser support.
FE$OFF	Parent/offspring tasking support.
FE$PKT	Preallocation of I/O packets.
FE$PLA	PLAS support.
FE$PMN	System supports pool monitoring.
FE$POL	System supports secondary pools.
FE$POS	System is a P/OS system; 1=yes (equivalent to FE$XT).
FE$PRO	System supports secondary pool prototype TCBs.
FE$RAS	Receive/send data packet support.
FE$RBN	Round robin scheduling support.
FE$RLK	System supports RMS record locking.
FE$SDW	System supports shadow recording.
FE$SHF	System supports shuffler task.
FE$STM	System has set system time directive.
FE$STP	Event flag mask is in the TCB; 1=yes.
FE$SWP	Executive level disk swapping support.
FE$TCM	System has separate terminal driver pool.
FE$UDS	System supports user data space.
FE$WAT	System has watchdog timer support.
FE$WND	System supports secondary pool file windows.

Symbol	Meaning
FE$X25	X.25 CEX is loaded.
FE$XCR	System crashed from XDT; 1=yes.
FE$XHR	System supports external task headers.
FE$XT	System is a P/OS system; 1=yes (equivalent to FE$POS).
FE$11S	RSX-11S system.
HF$BRG	P/OS bridge module present.
HF$CIS	Processor supports commercial instruction set.
HF$CLK	P/OS clock is present.
HF$EIS	Processor has extended instruction set.
HF$FPP	Processor has no floating point unit; 1=yes.
HF$FS	System is a file server on a P/OS Server system.
HF$INV	Nonvolatile RAM present; 1=yes.
HF$ITF	Invalid time format in nonvolatile RAM.
HF$NVR	XT nonvolatile RAM present; 1=yes (bit 17 decimal).
HF$UBM	Processor has unibus map; 1=yes (bit 1 decimal).
HF$WS	System is a workstation on a P/OS Server system.

8.32 FSS$ - FILE SPECIFICATION SCAN

The File Specification Scan directive returns the location and length of the parts of a file specification. It indicates which parts of a file specification are present and absent, as well as which parts contain wildcards.

The directive accepts as input parameters the address and length of the file specification (a string) and of a parse block. A *parse block* is the block of memory into which the FSS$ directive places a series of one-word values that describe the file specification. FSS$ first clears the parse block, then enters a descriptor for each of the fields in the file specification.

Table 8-7 shows the format of the FSS$ Parse Block.

High-Level Language Call

```
CALL FSS (fsbuf,fssz,prsblk,prssz,[reserv],[idsw])
```

fsbuf	Array containing file specification
fssz	Size of fsbuf in bytes
prsblk	Array to contain the parse block
prssz	Size of the parse block in bytes
reserv	Reserved parameter, must not be specified
idsw	Integer to receive directive status word

Macro Call

```
FSS$ fsbuf,fssz,prsblk,prssz,reserv
```

fsbuf	Address of file specification buffer
fssz	Size of the file specification buffer in bytes
prsblk	Address of the parse block
prssz	Size of the parse block in bytes
reserv	Reserved parameter - must be blank

Macro Expansion

```
.MACRO    FSS$        FSBUF,FSSZ,PRSBLK,PRSSZ,RESERV
.BYTE     207.,7 ;FSS$ MACRO DIC, DPB SIZE = 7 WORDS
.BYTE     5            ;SUBFUNCTION
.BYTE     0
.WORD     0
.WORD     FSBUF        ;FILE SPECIFICATION BUFFER ADDRESS
.WORD     FSSZ         ;FILE SPECIFICATION BUFFER SIZE
.WORD     PRSBLK       ;PARSE BLOCK ADDRESS
.WORD     PRSSZ        ;PARSE BLOCK SIZE
```

Local Symbol Definitions

L.LFUN	Subfunction code (1)
F.LSBF	Address of file specification buffer (2)
F.LSSZ	File specification buffer size in bytes (2)
F.LPBK	Address of the parse block (2)
F.LPBZ	Size of the parse block in bytes (2)

DSW Return Codes

IS.SUC	Successful completion.
IE.ADP	Part of the DPB or user buffer is out of the issuing task's address space, or the user does not have the proper access to that region.
IE.SDP	DIC or DPB size is invalid.

Table 8-7: Format of the FSS$ Parse Block

Offset (Decimal Byte)	Symbolic	Description
0	O$STAT	Status of the operation. This field can contain the following values:

Offset (Decimal Byte)	Symbolic	Description
		SU$SUC - Success ER$NOD - Error in node name (or imbalanced nodes for $RENAME) ER$DEV - Bad device, or inappropriate device type ER$DIR - Error in directory name ER$FNM - Error in filename ER$TYP - Error in file type extension ER$VER - Error in version number ER$ESS - Expanded string area too short ER$XTR - Extraneous field detected during parse
2	O$FLAG	A series of flags indicating what FSS$ found in the file specification. For each component present in the file specification, FSS$ sets the appropriate bit: FS$NOD - Node present FS$DEV - Device present FS$DIR - Directory FS$QUO - Quoted filename present FS$NAM - Filename present

Offset (Decimal Byte)	Symbolic	Description
		FS$TYP - File type present
		FS$VER - File version present
		FS$WCH - Wildcard character present
		FS$WDI - Wildcard in directory
		FS$WNA - Wildcard in filename
		FS$WTY - Wildcard in file type
		FS$WVE - Wildcard in file version
4	O$NODS	Length of the node specification
6	O$NODA	Address of the node specification
8	O$DEVS	Length of the device specification
10	O$DEVA	Address of the device specification
12	O$DIRS	Length of the directory specification
14	O$DIRA	Address of the directory specification
16	O$NAMS	Length of the filename specification
18	O$NAMA	Address of the filename specification
20	O$TYPS	Length of the type specification
22	O$TYPA	Address of the type specification
24	O$VERS	Length of the version specification
26	O$VERA	Address of the version specification
28	O$TRLS	Length of the trailing string
30	O$TRLA	Address of the trailing string. Note that this field is always filled, even when the length is zero.
32	O$ACCS	Length of the access control specification

Offset (Decimal Byte)	Symbolic	Description
34	O$ACCA	Address of the access control specification. Note that this field is always filled, even when the length is zero.
36	O$LTYP	This byte indicates the logical types that the file specification might contain. O$LTYP contains the following flags: P.LNON - No logical name present P.LNAM - The filename might be a logical name P.LDEV - The device name might be a logical name P.LNOD - The node specification might be a logical name
N/A	N/A	Reserved
38	O$PLEN	Length of the parse block

Definitions of each of the fields in the file specification follow.

- **Node**

  The node field contains all nodes specified, including those obtained through forced routing. The value in the O$NODS field includes the access control string size. If a node is present, then FSS$ sets the FS$NOD bit in the parse block's O$FLAG word.

- **Access Control**

  The access control string consists of the beginning quote, username and password, the ending quote, and the double colon. FSS$ returns the access control string address as the terminator of the initial node name in the O$ACCA field. However, if the initial node name does not contain an access control string, FSS$ sets the fields O$ACCS and 0$ACCA to zero.

- **Device**

  The device string consists of the name of the device terminated by a single colon. If a device is present, then FSS$ sets the FS$DEV bit in the parse block's O$FLAG word.

- **Directory**

  The directory is the string bounded either by square brackets ([]) or by angle brackets (<>). The directory string includes any characters that are valid in a directory specification, including wildcards and directory hierarchies. Note that FSS$ does not check the syntax of the directory for its validity within the context of the operation, such as would be required in network access operations.

  If a directory is present, then FSS$ sets the FS$DIR bit in the parse block's O$FLAG word. In addition, if wildcards are present, FSS$ sets the FS$WDI bit. (Wildcards are % and *.)

- **Filename**

  The filename is the string terminated by a period, a semicolon, or the end of the file specification.

  If a filename is present, then FSS$ sets the FS$NAM bit in the parse block's O$FLAG word. In addition, if wildcards are present, FSS$ sets the FS$WNA bit. (Wildcards are % and *.)

  If the file specification is a quoted string, FSS$ sets the FS$QUO bit in the O$FLAG word. In this case, the file specification represents either a foreign file (if being passed to another system) or an ANSI file. In the case of a foreign file, FSS$ allows a version field, but does not supply a default value. In the the case of an ANSI file, FSS$ allows a version field but not a type field.

- **Type**

  The type is the string terminated by a period or semicolon, or the end of the file specification. The type string always includes a leading period. Note that if the filename is a quoted string, then the type field must be null.

  If a type is present, then FSS$ sets the FS$TYP bit in the parse block's O$FLAG word. In addition, if wildcards are present, FSS$ sets the FS$WTY bit. (Wildcards are % and *.)

- **Version**

  The version is the string introduced by a period or the semicolon and terminated by the end of the string. The version string must contain the digits between zero and nine (optionally preceded by a minus sign), or a * wildcard.

  If a version is present, then FSS$ sets the FS$VER bit in the parse block's O$FLAG word. In addition, if wildcards are present, FSS$ sets the FS$WVE bit. (Wildcards are % and *.)

- **Trailing**

  The trailing string is that part of the input string that FSS$ could not successfully and completely parse. If FSS$ detects an error in the directory specification, for example, the trailing string includes the erroneous directory specification.

  This feature allows you to use FSS$ in command line parsing. Any character not part of a file specification terminates the scan and causes FSS$ to return all information obtained up to the point at which the unusual character occurred.

8.33 GDIR$ - GET DEFAULT DIRECTORY

The Get Default Directory directive retrieves the default directory string, returning it and the string length to a user-specified buffer.

High-Level Language Call

```
CALL GETDDS  (mod,iens,ienssz,[irsize],[idsw])
```

mod	The modifier of the logical name within a table. See Section 2.2 for details.
iens	Character array containing the equivalence name string
ienssz	Size (in bytes) of the equivalence name string
irsize	Buffer address of the returned equivalence string size
idsw	Integer to receive the Directive Status Word

Macro Call

```
GDIR$  mod,ens,enssz,rsize
```

mod	The modifier of the logical name within a table. See Section 2.2 for details.
ens	Buffer address of the equivalence name string
enssz	Size (in bytes) of the equivalence name string
rsize	Buffer address to which the size of the equivalence name string is returned

Macro Expansion

```
GDIR$   mod,ens,enssz,rsize
.BYTE   207.,6      ;GDIR$ MACRO DIC AND DPB SIZE
.BYTE   4           ;SUBFUNCTION CODE FOR GET DEFAULT
                    ;DIRECTORY
.BYTE   MOD         ;LOGICAL NAME MODIFIER
.WORD   0           ;RESERVED
.WORD   ENS         ;BUFFER ADDRESS OF EQUIVALENCE NAME
.WORD   ENSSZ       ;BYTES COUNT OF EQUIVALENCE STRING
.WORD   RSIZE       ;BUFFER ADDRRESS FOR RETURNED EQUIVALENCE
                    ;STRING
```

Local Symbol Definitions

G.DENS	Address of equivalence name buffer (2)
G.DESZ	Byte count of equivalence string (2)
G.DFUN	Subfunction code (1)
G.DMOD	Logical name modifier (1)
G.DRSZ	Buffer address for returned equivalence string (2)

DSW Return Codes

IS.SUC	Successful completion of service.
IE.RBS	The resulting equivalence name string is too large for the buffer to receive it.
IE.LNF	The specified logical name string was not found.
IE.IBS	The length of the logical or equivalence string is invalid. Each string length must be greater than 0 but not greater than 255 (decimal) characters.
IE.ADP	Part of the DPB or user buffer is out of the issuing task's address space, or the user does not have proper access to that region.
IE.SDP	DIC or DPB size is invalid.

High-Level Language Example

The following PASCAL program illustrates a call to the getdds subroutine.

```
program gdir(input,output);

(* this sample program makes a call to the system
directive GDIR to return the default directory. *)

type
 dirarray  = packed array [1..32] of char;
 (* hold for return dir name *)
var

p32   : dirarray;  (* directory name string *)
imod  : integer;   (* imod is an integer that we will set to 0 *)
```

```
siz   : integer;   (* size of the array that we are passing *)
bufad : integer;   (* size of returned string *)
dsw   : integer;   (* directive status word *)

[ external (getdds)]
 procedure getdir ( var  int    : integer;
                    var  str    : dirarray;
                    var  strsze : integer;
                    var  bufadd : integer;
                    var  drswd  : integer);  seq11;


                          (* MAIN *)
begin

imod   := 0;                                (* set the value = 0 *)
siz    := 32;                             (* ini stringsize to 32 *)
bufad  := -1;    (* this return contains the size of the string *)
dsw    := 0;                                    (* ini dsw to 0 *)


getdir(imod,p32,siz,bufad,dsw);  (* make the call *)

writeln('Default directory name : ',p32);       (* directory name *)
writeln('Directory name size    : ',bufad:2);   (* size of string *)
writeln('DSW                    : ',dsw:2);     (* dsw status     *)
end.
```

8.34 GLUN$ - GET LUN INFORMATION

The Get LUN Information directive instructs the system to fill a six-word buffer with information about a physical device unit to which a LUN is assigned. If requests to the physical device unit have been redirected to another unit, the information returned will describe the effective assignment.

High-Level Language Call

```
CALL GETLUN (lun,dat[,ids])
```

lun	Logical unit number
dat	A six-word integer array to receive LUN information
ids	Directive status

Macro Call

```
GLUN$   lun,bufa
```

lun	Logical unit number
bufa	Address of six-word buffer that will receive the LUN information

Buffer Format

Word 0	Name of assigned device
Word 1	Unit number of assigned device and flags byte (flags byte equals 200 if the device driver is resident, or 0 if the driver is not loaded)
Word 2	First device characteristics word:
Bit 0	Record-oriented device (DV.REC,1=yes)[FD.REC]*
Bit 1	Carriage-control device (DV.CCL,1=yes)[FD.CCL]

---

* Bits with associated symbols have the symbols shown in square brackets. These symbols can be defined for use by a task by means of the FCSBT$ macro.

Bit 2	Terminal device (DV.TTY,1=yes)[FD.TTY]
Bit 3	Directory (file-structured) device (DV.DIR,1=yes)[FD.DIR]
Bit 4	Reserved
Bit 5	Sequential device (DV.SQD,1=yes)[FD.SQD]
Bit 6	Mass storage device (DV.MSD,1=yes)
Bit 7	Reserved for future use.
Bit 8	Reserved for future use.
Bit 9	Unit software write-locked (DV.SWL,1=yes)
Bit 10	Reserved for future use.
Bit 11	Reserved for future use.
Bit 12	Pseudo device (DV.PSE,1=yes)
Bit 13	Device mountable as a communications channel (DV.COM,1=yes)
Bit 14	Device mountable as a FILES-11 device (DV.F11,1=yes)
Bit 15	Device mountable (DV.MNT,1=yes)

Word 3 Second device characteristics word

Word 4 Third device characteristics word (words 3 and 4 are device driver specific)

Word 5 Fourth device characteristics word (normally buffer-size)

Macro Expansion

```
GLUN$   7,LUNBUF
.BYTE   5,3             ;GLUN$ MACRO DIC, DPB SIZE=3 WORDS
.WORD   7               ;LOGICAL UNIT NUMBER 7
.WORD   LUNBUF          ;ADDRESS OF SIX-WORD BUFFER
```

Local Symbol Definitions

G.LULU Logical unit number (2)

G.LUBA Buffer address (2)

The following offsets are assigned relative to the start of the LUN information buffer:

G.LUNA Device name (2)

G.LUNU Device unit number (1)

G.LUFB Flags byte (1)

G.LUCW Four device characteristics words (8)

DSW Return Codes

IS.SUC Successful completion.

IE.ULN Unassigned LUN.

IE.ILU Invalid logical unit number.

IE.ADP Part of the DPB or buffer is out of the issuing task's address space.

IE.SDP DIC or DPB size is invalid.

8.35 GMCR$ - GET COMMAND LINE

The Get Command Line directive instructs the system to transfer an 80-byte command line to the issuing task.

High-Level Language Call

```
CALL GETMCR (buf[,ids])
```

buf	An 80-byte array to receive command line
ids	Directive status

Macro Call

```
GMCR$
```

Macro Expansion

```
GMCR$
.BYTE   127.,41.        ;GMCR$ MACRO DIC,   DPB   SIZE=41.
.BLKW   40.             ;80.  BYTE MCR COMMAND LINE BUFFER
```

Local Symbol Definitions

G.MCRB	Command line buffer (80)

DSW Return Codes

+n	Successful completion; n is the number of data bytes transferred (excluding the termination character). The termination character is, however, in the buffer.
IE.AST	No command line exists for the issuing task, or the task has already issued one or more Get Command Line directives and has retrieved the entire command line.
IE.ADP	Part of the DPB is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

Notes

1. The GMCR$S form of the macro is not supplied, since the DPB receives the actual command line.

2. The system processes all lines to:

   - o Convert tabs to a single space
   - o Convert multiple spaces to a single space
   - o Convert lowercase to uppercase
   - o Remove all trailing blanks

   The terminator (<RETURN> or <ESC>) is the last character in the line.

3. If the character before the terminator is a hyphen, there is at least one continuation line present. Therefore, you must issue another GMCR$ directive to obtain the rest of the command line.

8.36 GMCX$ - GET MAPPING CONTEXT

The Get Mapping Context directive causes the Executive to return a description of the current window-to-region mapping assignments. The returned description is in a form that enables the user to restore the mapping context through a series of Create Address Window directives. The macro argument specifies the address of a vector that contains one Window Definition Block (WDB) for each window block allocated in the task's header, plus a terminator word.

For each window block in the task's header, the Executive sets up a WDB in the vector as follows:

1. If the window block is unused (that is, if it does not correspond to an existing address window), the Executive does not record any information about that block in a WDB. Instead, the Executive uses the WDB to record information about the first block encountered that corresponds to an existing window. In this way, unused window blocks are ignored in the mapping context description returned by the Executive.

2. If a window block describes an existing unmapped address window, the Executive fills in the offsets W.NID, W.NAPR, W.NBAS, and W.NSIZ with information sufficient to re-create the window. The window status word W.NSTS is cleared.

3. If a window block describes an existing mapped window, the Executive fills in the offsets W.NAPR, W.NBAS, W.NSIZ, W.NRID, W.NOFF, W.NLEN, and W.NSTS with information sufficient to create and map the address window. WS.MAP is set in the status word (W.NSTS) and, if the window is mapped with write access, the bit WS.WRT is set as well.

Note that in no case does the Executive modify W.NSRB.

The terminator word, which follows the last WDB filled in, is a word equal to the negative of the total number of window blocks in the task's header. It is thereby possible to issue a TST or TSTB instruction to detect the last WDB used in the vector. The terminating word can also be used to determine the number of window blocks built into the task's header.

When Create Address Window directives are used to restore the mapping context, there is no guarantee that the same address window IDs will be used. The user must therefore be careful to use the latest window IDs returned from the Create Address Window directives.

High-Level Language Call

```
CALL GMCX (imcx[,ids])
```

imcx — An integer array to receive the mapping context. The size of the array is (8*n)+1 where n is the number of window blocks in the task's header. The maximum size is (8*24)+1=193 words.

ids — Directive status.

Macro Call

```
GMCX$   wvec
```

wvec — The address of a vector of n Window Definition Blocks, followed by a terminator word; n is the number of window blocks in the task's header.

Macro Expansion

```
GMCX$   VECADR
.BYTE   113.,2          ;GMCX$ MACRO DIC, DPB SIZE=2 WORDS
.WORD   VECADR          ;WDB VECTOR ADDRESS
```

Definition Block Parameters

Table 8-8 shows the Window Definition Block parameters for this directive.

Table 8-8: Window Definition Block Parameters for GMCX$

Array Element	Offset	Description
Input Parameters		
None		
Output Parameters		
iwdb(1) bits 0-7	W.NID	ID of address window
iwdb(1) bits 8-15	W.NAPR	Base APR of the window

Array Element	Offset	Description
iwdb(2)	W.NBAS	Base virtual address of the window
iwdb(3)	W.NSIZ	Size, in 32-word blocks, of the window
iwdb(4)	W.NRID	ID of the mapped region, or no change if the window is unmapped
iwdb(5)	W.NOFF	Offset, in 32-word blocks, from the start of the region at which mapping begins, or no change if the window is unmapped
iwdb(6)	W.NLEN	Length, in 32-word blocks, of the area currently mapped within the region, or no change if the window is unmapped
iwdb(7)	W.NSTS	Bit settings* in the window status word (all 0 if the window is not mapped):
		Bit **Definition**
		WS.MAP 1 if the window is mapped
		WS.WRT 1 if the window is mapped with write access

NOTE

The length mapped (W.NLEN) can be less than the size of the window (W.NSIZ) if the area from W.NOFF to the end of the partition is smaller than the window size.

Local Symbol Definitions

G.MCVA Address of the vector (wvec) containing the window definition blocks and terminator word (2)

* If you are a high-level language programmer, see Section 5.5.2 to determine the bit values represented by the symbolic names described.

DSW Return Codes

IS.SUC	Successful completion.
IE.ADP	Address check of the DPB or the vector (wvec) failed.
IE.SDP	DIC or DPB size is invalid.

8.37 GPRT$ - GET PARTITION PARAMETERS

The Get Partition Parameters directive instructs the system to fill an indicated three-word buffer with partition parameters. If a partition is not specified, the partition of the issuing task is assumed.

High-Level Language Call

```
CALL GETPAR ([prt],buf[,ids])
```

prt	Partition name
buf	A three-word integer array to receive partition parameters
ids	Directive status

Macro Call

```
GPRT$   [prt],buf
```

prt	Partition name
buf	Address of a three-word buffer

Buffer Format

Word 0	Partition physical base address expressed as a multiple of 32 words (partitions are always aligned on 32-word boundaries). Therefore, a partition starting at 40000 (octal) will have 400 (octal) returned in this word.
Word 1	Partition size expressed as a multiple of 32 words.
Word 2	Partition flags word. This word is returned equal to 0 to indicate a system-controlled partition, or equal to 1 to indicate a user-controlled partition.

Macro Expansion

```
GPRT$   ALPHA,DATBUF
.BYTE   65.,4           ;GPRT$ DIC, DPB SIZE=4 WORDS
.RAD50  /ALPHA/         ;PARTITION "ALPHA"
.WORD   DATBUF          ;ADDRESS OF THREE-WORD BUFFER
```

Local Symbol Definitions

G.PRPN Partition name (4)

G.PRBA Buffer address (2)

The following offsets are assigned relative to the start of the partition parameters buffer:

G.PRPB Partition physical base address expressed as an absolute 32-word block number (2)

G.PRPS Partition size expressed as a multiple of 32-word blocks (2)

G.PRFW Partition flags word (2)

DSW Return Codes

Successful completion is indicated by a cleared Carry bit, and the starting address of the partition is returned in the DSW. The returned address is virtual and is always zero if it is not the task partition. Unsuccessful completion is indicated by a set Carry bit and one of the following codes in the DSW:

IE.INS Specified partition not in system.

IE.ADP Part of the DPB or buffer is out of the issuing task's address space.

IE.SDP DIC or DPB size is invalid.

Notes

1. A variation of this directive exists called Get Region Parameters. When the first word of the two-word partition name is 0, the Executive interprets the second word of the partition name as a region ID. If the two-word name is 0,0, it refers to the task region of the issuing task.

2. Omission of the partition-name argument returns parameters for the issuing task's unnamed subpartition, not for the system-controlled partition.

8.38 GREG$ - GET REGION PARAMETERS

The Get Region Parameters directive instructs the Executive to fill an indicated three-word buffer with region parameters. If a region is not specified, the task region of the issuing task is assumed.

This directive is a variation of the Get Partition Parameters directive.

High-Level Language Call

```
CALL GETREG ([rid],buf[,ids])
```

rid — Region id

buf — A three-word integer array to receive region parameters

ids — Directive status

Macro Call

```
GREG$   [rid],buf
```

rid — Region ID

buf — Address of a three-word buffer

Buffer Format

Word 0 — Region base address expressed as a multiple of 32 words. (Regions are always aligned on 32-word boundaries.) Thus, a region starting at 1000 (octal) will have 10 (octal) returned in this word.

Word 1 — Region size expressed as a multiple of 32 words.

Word 2 — Region flags word. This word is returned equal to 0 if the region resides in a system-controlled partition, or equal to 1 if the region resides in a user-controlled partition.

Macro Expansion

```
GREG$   RID,DATBUF
.BYTE   65.,4           ;GREG$ MACRO DIC, DPB SIZE=4 WORDS
.WORD   0               ;WORD THAT DISTINGUISHES GREG$

;FROM GPRT$
.WORD   RID             ;REGION ID
.WORD   DATBUF          ;ADDRESS OF THREE-WORD BUFFER
```

Local Symbol Definitions

G.RGID Region ID (2)

G.RGBA Buffer address

The following offsets are assigned relative to the start of the region parameters buffer:

G.RGRB Region base address expressed as an absolute 32-word block number (2)

G.RGRS Region size expressed as a multiple of 32-word blocks (2)

G.RGFW Region flags word (2)

DSW Return Codes

Successful completion is indicated by carry clear, and the starting address of the region is returned in the DSW. The returned address is virtual and is always zero if it is not the task region. Unsuccessful completion is indicated by carry set and one of the following codes in the DSW:

IE.NVR Invalid region ID.

IE.ADP Part of the DPB or buffer is out of the issuing task's address space.

IE.SDP DIC or DPB size is invalid.

8.39 GTIM$ - GET TIME PARAMETERS

The Get Time Parameters directive instructs the system to fill an indicated eight-word buffer with the current time parameters. All time parameters are delivered as binary numbers. The value ranges (in decimal) are shown below in the buffer format.

High-Level Language Call

```
CALL GETTIM (ibfp[,ids])
```

ibfp An eight-word integer array

Macro Call

```
GTIM$   bufa
```

bufa Address of eight-word buffer

Buffer Format

Word	0	Year (since 1900)
Word	1	Month (1-12)
Word	2	Day (1-31)
Word	3	Hour (0-23)
Word	4	Minute (0-59)
Word	5	Second (0-59)
Word	6	Tick of second (fixed rate of 64 decimal)
Word	7	Ticks per second (fixed rate of 64 decimal)

Macro Expansion

```
GTIM$   DATBUF
.BYTE   61.,2           ;GTIM$ DIC, DPB SIZE=2 WORDS
.WORD   DATBUF          ;ADDRESS OF 8.-WORD BUFFER
```

Local Symbol Definitions

G.TIBA Buffer address (2)

The following offsets are assigned relative to the start of the time parameters buffer:

G.TIYR	Year (2)
G.TIMO	Month (2)
G.TIDA	Day (2)
G.TIHR	Hour (2)
G.TIMI	Minute (2)
G.TISC	Second (2)
G.TICT	Clock tick of second (2)
G.TICP	Clock ticks per second (2)

DSW Return Codes

IS.SUC	Successful completion.
IE.ADP	Part of the DPB or buffer is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

Note

The format of the time buffer is compatible with that of the buffers used with the Set System Time directive.

8.40 GTSK$ - GET TASK PARAMETERS

The Get Task Parameters directive instructs the system to fill an indicated 16-word buffer with parameters relating to the issuing task.

High-Level Language Call

```
CALL GETTSK (buf[,ids])
```

buf	A 16-word integer array to receive the task parameters
ids	Directive status

Macro Call

```
GTSK$  bufa
```

bufa	Address of a 16-word buffer

Buffer Format

Word 0	Issuing task's name in Radix-50 (first half)
Word 1	Issuing task's name in Radix-50 (second half)
Word 2	Partition name in Radix-50 (first half)
Word 3	Partition name in Radix-50 (second half)
Word 4	Undefined by the system
Word 5	Undefined by the system
Word 6	Run priority
Word 7	User Identification Code (UIC) of issuing task (the task's default UIC)*
Word 10	Number of logical I/O units (LUNs)
Word 11	Undefined by the system
Word 12	Undefined by the system

---

* See note in RQST$ description on contents of words 7 and 17.

Word 13 (Address of task SST vector tables)*

Word 14 (Size of task SST vector table in words)*

Word 15 Size (in bytes) either of task's address window 0 in mapped systems, or of task's partition in unmapped system (equivalent to partition size)

Word 16 System on which task is running:

11 The system always returns this code

Word 17 Protection UIC**

Macro Expansion

```
GTSK$   DATBUF
.BYTE   63.,2           ;GTSK$ MACRO DIC, DPB SIZE = 2 WORDS
.WORD   DATBUF          ;ADDRESS OF 16-WORD BUFFER
```

Local Symbol Definitions

G.TSBA Buffer address (2)

The following offsets are assigned relative to the task parameter buffer:

G.TSTN Task name (4)

G.TSPN Partition name (4)

G.TSPR Priority (2)

G.TSGC UIC group code (1)

G.TSPC UIC member code (1)

G.TSNL Number of logical units (2)

G.TSVA Task's SST vector address (2)

G.TSVL Task's SST vector length in words (2)

---

* Words 13 and 14 will contain valid data if word 14 is not zero. If word 14 is zero, the contents of word 13 are meaningless.

** See note in RQST$ description on contents of words 7 and 17.

G.TSTS	Task size (2)
G.TSSY	System on which task is running (2)
G.TSDU	Protection UIC (2)

DSW Return Codes

IS.SUC	Successful completion.
IE.ADP	Part of the DPB or buffer is out of the issuing task's address space.
IE.SDP	DIC or DPB is invalid.

8.41 MAP$ - MAP ADDRESS WINDOW

The Map Address Window directive maps an existing window to an attached region. The mapping begins at a specified offset from the start of the region. If the window is already mapped elsewhere, the Executive unmaps it before carrying out the mapping assignment described in the directive.

For the mapping assignment, a task can specify any length that is less than or equal to both:

- The window size specified when the window was created
- The length remaining between the specified offset within the region and the end of the region

A task must be attached with write access to a region in order to map to it with write access. To map to a region with read-only access, the task must be attached with either read or write access.

If W.NLEN is set to 0, the length defaults to either the window size or the length remaining in the region, whichever is smaller. (Since the Executive returns the actual length mapped as an output parameter, the task must clear that parameter in the WDB before issuing the directive each time it wants to default the length of the map.)

The values that can be assigned to W.NOFF depend on the setting of bit WS.64B in the window status word (W.NSTS):

- If WS.64B = 0, the offset specified in W.NOFF must represent a multiple of 256 words (512 bytes). Because the value of W.NOFF is expressed in units of 32-word blocks, the value must be a multiple of 8.
- If WS.64B = 1, the task can align on 32-word boundaries; the programmer can therefore specify any offset within the region.

High-Level Language Call

```
CALL MAP (iwdb[,ids])
```

iwdb	An eight-word integer array containing a Window Definition Block (see Section 5.5.2.2)
ids	Directive status

Macro Call

```
MAP$   wdb
```

wdb Window Definition Block address

Macro Expansion

```
MAP$    WDBADR
.BYTE   121.,2          ;MAP$ MACRO DIC, DPB SIZE=2 WORDS
.WORD   WDBADR          ;WDB ADDRESS
```

Definition Block Parameters

Table 8-9 shows the Window Definition Block parameters for this directive.

Local Symbol Definitions

M.APBA	Window Definition Block address (2)

DSW Return Codes

IS.SUC	Successful completion.
IE.PRI	Privilege violation.
IE.NVR	Invalid region ID.
IE.NVW	Invalid address window ID.
IE.ALG	Task specified an invalid region offset and length combination in the Window Definition Block parameters; or WS.64B = 0 and the value of W.NOFF is not a multiple of 8.
IE.HWR	Region had a parity error or a load failure.
IE.ITS	WS.RES was set, and region is not resident.
IE.ADP	Part of the DPB or WDB is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

Table 8-9: Window Definition Block Parameters for MAP$

Array Element	Offset	Description
Input Parameters		
iwdb(1) bits 0-7	W.NID	ID of the window to be mapped
iwdb(4)	W.NRID	ID of the region to which the window is to be mapped, or 0 if the task region is to be mapped
iwdb(5)	W.NOFF	Offset, in 32-word blocks, within the region at which mapping is to begin. Note that if WS.64B in the window status word equals 0, the value specified must be a multiple of 8
iwdb(6)	W.NLEN	Length, in 32-word blocks, within the region to be mapped, or 0 if the length is to default to either the size of the window or the space remaining in the region from the specified offset, whichever is smaller
iwdb(7)	W.NSTS	Bit settings* in the window status word: **Bit** — **Definition** WS.WRT — 1 if write access is desired WS.64B — 0 for 256-word (512-byte) alignment, or 1 for 32-word (64-byte) alignment
Output Parameters		
iwdb(6)	W.NLEN	Length of the area within the region actually mapped by the window

* If you are a high-level language programmer, see Section 5.5.2 to determine the bit values represented by the symbolic names described.

Array Element	Offset	Description
iwdb(7)	W.NSTS	Bit settings* in the window status word:

Bit	Definition
WS.UNM	1 if the window was unmapped first

Notes

1. When the Map Address Window directive is issued, the task can be blocked until the region is loaded.

2. Bit WS.RES in word W.NSTS of the Window Definition Block, when set, specifies that the region should be mapped only if the region is resident.

3. See Chapter 5 for complete information on using the memory management features of the Professional.

4. A fast remapping feature is available for frequently mapped regions; see Section 5.7 for details.

---

* If you are a high-level language programmer, see Section 5.5.2 to determine the bit values represented by the symbolic names described.

8.42 MRKT$ - MARK TIME

The Mark Time directive instructs the system to declare a significant event after an indicated time interval. The interval begins when the task issues the directive; however, task execution continues during the interval.

If an event flag is specified, the flag is cleared when the directive is issued, and set when the significant event occurs.

If an AST entry point address is specified, an AST occurs at the time of the significant event. When the AST occurs, the task's PS, PC, directive status, Wait For mask words, and the event flag number specified in the directive, are pushed onto the issuing task's stack.

If neither an event flag number nor an AST service entry point is specified, the significant event still occurs after the indicated time interval. (See Notes.)

High-Level Language Call

```
CALL MARK (efn,tmg,tnt[,ids])
```

efn — Event flag number

tmg — Time interval magnitude (see Note 5)

tnt — Time interval unit (see Note 5)

ids — Directive status

The ISA standard call for delaying a task for a specified time interval is also provided:

```
CALL WAIT (tmg,tnt[,ids])
```

tmg — Time interval magnitude (see Note 5)

tnt — Time interval unit (see Note 5)

ids — Directive status

Macro Call

```
MRKT$   [efn],tm,tu[,ast]
```

efn — Event flag number

tm — Time interval magnitude (see Note 6)

tu Time interval unit (see Note 6)

ast AST entry point address

Macro Expansion

```
MRKT$   52.,30.,2,MRKAST
.BYTE   23.,5           ;MRKT$ MACRO DIC, DPB SIZE=5 WORDS
.WORD   52.             ;EVENT FLAG NUMBER 52.
.WORD   30.             ;TIME MAGNITUDE=30.
.WORD   2               ;TIME UNIT=SECONDS
.WORD   MRKAST          ;ADDRESS OF MARK TIME AST ROUTINE
```

Local Symbol Definitions

M.KTEF Event flag (2)

M.KTMG Time magnitude (2)

M.KTUN Time unit (2)

M.KTAE AST entry point address (2)

DSW Return Codes

For CALL MARK and MRKT$:

IS.SUC Successful completion.

IE.UPN Insufficient dynamic memory.

IE.ITI Invalid time parameter.

IE.IEF Invalid event flag number (EFN<0 or EFN>64).

IE.ADP Part of the DPB is out of the issuing task's address space.

IE.SDP DIC or DPB size is invalid.

For CALL WAIT:

The system provides the following positive error codes to be returned for ISA calls:

1 Successful completion

2 Insufficient dynamic storage

3 Specified task not installed

94	Invalid time parameters
98	Invalid event flag number
99	Part of DPB out of task's range
100	DIC or DPB size invalid

Notes

1. Mark Time requires dynamic memory for the clock queue entry.

2. If an AST entry point address is specified, the AST service routine is entered with the task's stack in the following state:

SP+10	Event flag mask word*
SP+06	PS of task prior to AST
SP+04	PC of task prior to AST
SP+02	DSW of task prior to AST
SP+00	Event flag number or zero (if none was specified in the Mark Time directive)

   The event flag number must be removed from the task's stack before an AST Service Exit directive is executed.

3. If the directive is rejected, the specified event flag is not guaranteed to be cleared or set. Consequently, if the task indiscriminately executes a Wait For directive and the Mark Time directive is rejected, the task can wait indefinitely. Care should always be taken to ensure that the directive was successfully completed.

4. If a task issues a Mark Time directive that specifies a common event flag and then exits before the indicated time has elapsed, the event flag is not set.

5. The Executive returns the code IE.ITI (or 94) in the Directive Status Word if the directive specifies an invalid time parameter. The time parameter consists of

---

* The event flag mask word preserves the Wait For conditions of a task prior to AST entry. A task can, after an AST, return to a Wait For state. Because these flags and the other stack data are in the user task, they can be modified. Such modification is strongly discouraged, however, since the task can easily fault on obscure conditions. For example, clearing the mask word results in a permanent Wait For state.

two components: the time interval magnitude and the time interval unit, represented by the arguments tm and tu, respectively.

A legal magnitude value (tm) is related to the value assigned to the time interval unit (tu). The unit values are encoded as follows:

For an ISA FORTRAN call (CALL WAIT):

0 Ticks. A tick occurs for each clock interrupt at a rate of 64 (decimal) ticks per second.

1 Milliseconds. The subroutine converts the specified magnitude to the equivalent number of system clock ticks. On systems with line frequency clocks, millisecond Mark Time requests can only be approximations.

For all other high-level language and macro calls:

1 Ticks. A tick occurs for each clock interrupt at a rate of 64 (decimal) ticks per second.

For all calls:

2 Seconds

3 Minutes

4 Hours

The magnitude (tm) is the number of units to be clocked. The following list describes the magnitude values that are valid for each type of unit. In no case can the value of tm exceed 24 hours. The list applies to both high-level language and macro calls.

If tu = 0, 1, or 2, tm can be any positive value with a maximum of 15 bits.

If tu = 3, tm can have a maximum value of 1440 (decimal).

If tu = 4, tm can have a maximum value of 24 (decimal).

6. The minimum time interval is one tick. If you specify a time interval of zero, it will be converted to one tick.

8.43 PFCS$ - PARSE FCS SPECIFICATION

The Parse FCS Specification separates the parts of an FCS file specification, expands any logical name references, and merges default name block information into the specification. The result is a fully expanded FCS file specification.

NOTE

The result cannot contain a node specification.

This directive provides the full capabilites of the ACHN$ and FSS$ directives, in that it allows you to perform a LUN assignment (like ACHN$) and it returns a parse block (like FSS$). See the descriptions of those directives for full information.

For information on parsing an RMS file specification, see Section 8.44.

High-Level Language Call

```
CALL PRSFCS([mod],[tbmsk],[lun],prbuf,prsz,rsbuf,rssz,
     [rslen],[prsblk,prssz],[dfnbk,dfnsz],[rsmsk],[idsw])
```

mod	Modifier for logical name table entries
tbmsk	Inhibit mask to prevent a logical table from being searched
prbuf	Array containing the primary file specification
prsz	Length of prbuf in bytes
rsbuf	Array to receive resultant file specification
rssz	Length of rsbuf in bytes
rslen	Integer to receive length of resultant string
prsblk	Parse block
prssz	Length of prsblk in bytes
dfnbk	Array containing default name block
dfnsz	Length of dfnbk in bytes
rsmsk	A mask of the fields to suppress in the resultant string

idsw Integer to receive directive status word

All commas are required, unless followed by a right parenthesis.

Macro Call

```
PFCS$ [mod],tbmsk,lun,prbuf,prsz,rsbuf,rssz,rslen,
      prsblk,prssz,dfnbk,dfnsz,rsmsk
```

mod — Optional modifier for duplicate logical name table entries

tbmsk — A byte whose low 4 bits constitute an inhibit mask to prevent the system from searching a particular logical table. For each of the following bits, if the bit is set, the system does not search the corresponding table:

Logical Name Table	Bit of tbmsk	Octal
System table (LT.SYS)	0	1
User table (LT.USR)	2	4
Session table (LT.SES)	4	20

lun — The LUN to be assigned

prbuf — Address of the primary file specification buffer

prsz — Size of the primary file specification buffer in bytes

rsbuf — Address of the resultant file specification buffer

rssz — Size of the resultant file specification buffer in bytes

rslen — Address of a word to receive the resultant string size

prsblk — Address of the parse block; see Section 8.32 for a description of the parse block

prssz — Size of the parse block in bytes

dfnbk — Address of the default name block

dfnsz — Size of the default name block

rsmsk Mask that allows you to suppress fields in the resultant string.

Macro Expansion

```
.MACRO  PFCS$     MOD,TBMSK,LUN,PRBUF,PRSZ,RSBUF,RSSZ,
                       RSLEN,PRSBLK,PRSSZ,DFNBK,DFNSZ,RSMSK
.BYTE   207.,13.  ;PFCS$ MACRO DIC, DPB SIZE = 13. WORDS
.BYTE   8.        ;SUBFUNCTION
.BYTE   MOD       ;MODIFIER
.BYTE   LUN       ;LOGICAL UNIT NUMBER
.BYTE   TBMSK     ;TABLE MASK
.WORD   PRBUF     ;FILE SPECIFICATION BUFFER ADDRESS
.WORD   PRSZ      ;FILE SPECIFICATION BUFFER SIZE
.WORD   RSBUF     ;RESULTANT STRING BUFFER (RSB) ADDRESS
.WORD   RSSZ      ;RESULTANT STRING BUFFER SIZE
.WORD   RSLEN     ;ADDRESS OF WORD TO RECEIVE RSB LENGTH
.WORD   PRSBLK    ;PARSE BLOCK ADDRESS
.WORD   PRSSZ     ;PARSE BLOCK SIZE
.WORD   DFNBK     ;DEFAULT NAME BLOCK ADDRESS
.WORD   DFNSZ     ;DEFAULT NAME BLOCK SIZE
.WORD   RSMSK     ;INHIBIT MASK
```

Local Symbol Definitions

F.LFUN Subfunction code (1)

F.LMOD Logical name modifier (1)

F.LLUN LUN number (1)

F.LTBL Table inhibit mask (1)

F.LPBF Address of the primary file specification buffer (2)

F.LPSZ Size of the primary file specification buffer in bytes (2)

F.LRBF Address of the resultant file specification buffer (2)

F.LRSZ Size of the resultant file specification buffer in bytes (2)

F.LRLN Length of the resultant file specification (2)

F.LPRS Parse block address (2)

F.LPRZ Size of the parse block in bytes (2)

F.LDBF Address of the default name block (2)

F.LDSZ Size of the default name block (2)

F.LMSK Inhibit mask (2)

DSW Return Codes

IS.SUC Successful completion.

IE.ADP Part of the DPB or user buffer is out of the issuing task's address space, or the user does not have the proper access to that region.

IE.SDP DIC or DPB size is invalid.

IE.IDU Invalid device or unit.

IE.ILU Invalid LUN.

IE.LNL LUN usage is interlocked.

8.44 PRMS$ - PARSE RMS SPECIFICATION

Parse RMS Specification separates the parts of an RMS primary file specification and expands any logical name references, then performs a merge operation. Then, PRMS$ processes the default file specification and merges this information with the previous result, including the current device, default directory, and node name if necessary. The result is a fully expanded RMS file specification.

This directive provides the full capabilites of the ACHN$ and FSS$ directives, in that it allows you to perform a LUN assignment (like ACHN$), and it returns a parse block (like FSS$). See the descriptions of those directives for full information.

For information on parsing an FCS file specification, see Section 8.43.

High-Level Language Call

```
CALL PRSRMS([mod],[tbmsk],[lun],prbuf,prsz,rsbuf,rssz,
        [rslen],[prsblk,prssz],[dfbuf,dfsz],[rsmsk],[idsw])
```

mod	Modifier for logical name table entries.
tbmsk	Inhibit mask to prevent a logical table from being searched.
prbuf	Array containing the primary file specification.
prsz	Length of prbuf in bytes.
rsbuf	Array to receive resultant file specification.
rssz	Length of rsbuf in bytes.
rslen	Integer to receive length of resultant string.
prsblk	Parse block.
prssz	Length of prsblk in bytes.
dfbuf	Array containing default file specification.
dfsz	Length of dfbuf in bytes.
rsmsk	A mask of the fields to suppress in the resultant string.

idsw Integer to receive directive status word.

All commas are required, unless they are followed by a right parenthesis.

Macro Call

```
PRMS$ [mod],tbmsk,lun,prbuf,prsz,rsbuf,rssz,rslen,
          prsblk,prssz,dfbuf,dfsz,rsmsk
```

mod Optional modifier for duplicate logical name table entries

tbmsk A byte whose low 4 bits constitute an inhibit mask to prevent the system from searching a particular logical table. For each of the following bits, if the bit is set, the system does not search the corresponding table:

Logical Name Table	Bit of tbmsk	Octal
System table (LT.SYS)	0	1
User table (LT.USR)	2	4
Session table (LT.SES)	4	20

lun The LUN to be assigned

prbuf Address of the primary file specification buffer

prsz Size of the primary file specification buffer in bytes

rsbuf Address of the resultant file specification buffer

rssz Size of the resultant file specification buffer in bytes

rslen Address of a word to receive the resultant string size

prsblk Address of the parse block. (See Section 8.32 for a description of the parse block.)

prssz Size of the parse block in bytes

dfbuf Address of the default file specification buffer

dfsz	Size of the default file specification buffer in bytes
rsmsk	Mask that allows you to suppress fields in the resultant string

Local Symbol Definitions

R.LFUN	Subfunction code (1)
R.LMOD	Logical name modifier (1)
R.LLUN	LUN number (1)
R.LTBL	Table inhibit mask (1)
R.LPBF	Address of the primary file specification buffer (2)
R.LPSZ	Size of the primary file specification buffer in bytes (2)
R.LRBF	Address of the resultant file specification buffer (2)
R.LRSZ	Resultant file specification buffer size in bytes (2)
R.LRLN	Length of the resultant file specification (2)
R.LPRS	Parse block address (2)
R.LPRZ	Parse block size (2)
R.LDBF	Address of the default file specification buffer (2)
R.LDSZ	Size of the default file specification buffer in bytes (2)
R.LMSK	Inhibit mask (2)

Macro Expansion

```
.MACRO  PRMS$       MOD,TBMSK,LUN,PRBUF,PRSZ,RSBUF,RSSZ,
                         RSLEN,PRSBLK,PRSSZ,DFBUF,DFSZ,RSMSK
.BYTE   207.,13.    ;PRMS$ MACRO DIC, DPB SIZE = 13.  WORDS
.BYTE   7           ;SUBFUNCTION
.BYTE   MOD         ;MODIFIER
.BYTE   LUN         ;LOGICAL UNIT NUMBER
.BYTE   TBMSK       ;TABLE MASK
```

```
.WORD   PRBUF       ;PRIMARY STRING
.WORD   PRSZ        ;PRIMARY STRING SIZE
.WORD   RSBUF       ;RESULTANT STRING BUFFER (RSB) ADDRESS
.WORD   RSSZ        ;RESULTANT STRING BUFFER SIZE
.WORD   RSLEN       ;ADDRESS OF WORD TO RECEIVE RSB SIZE
.WORD   PRSBLK      ;PARSE BLOCK ADDRESS
.WORD   PRSSZ       ;PARSE BLOCK SIZE
.WORD   DFBUF       ;DEFAULT FILE SPECIFICATION ADDRESS
.WORD   DFSZ        ;DEFAULT FILE SPECIFICATION SIZE
.WORD   RSMSK       ;INHIBIT MASK
```

DSW Return Codes

IS.SUC — Successful completion.

IE.ADP — Part of the DPB or user buffer is out of the issuing task's address space, or the user does not have the proper access to that region.

IE.SDP — DIC or DPB size is invalid.

IE.IDU — Invalid device or unit.

IE.ILU — Invalid LUN.

IE.LNL — LUN usage is interlocked.

8.45 QIO$ - QUEUE I/O REQUEST

The Queue I/O Request directive instructs the system to place an I/O request for an indicated physical device unit into a queue of priority-ordered requests for that device unit. The physical device unit is specified as a logical unit number (LUN) assigned to the device.

The Executive declares a significant event when the I/O transfer completes. If the directive call specifies an event flag, the Executive clears the flag when the request is queued and sets the flag when the significant event occurs.

The I/O status block is also cleared when the request is queued and is set to the final I/O status when the I/O request is complete. If an AST service routine entry point address is specified, the AST occurs upon I/O completion, and the task's Wait For mask word, PS, PC, DSW, and the address of the I/O status block, are pushed onto the task's stack.

The description below deals solely with the Executive directive. (See Notes.)

High-Level Language Call

```
CALL QIO (fnc,lun,[efn],[pri],[isb],[prl][,ids])
```

fnc	I/O function code (see Appendix A)
lun	Logical unit number
efn	Event flag number
pri	Priority; ignored, but must be present
isb	A two-word integer array to receive final I/O status
prl	A six-word integer array containing device-dependent parameters to be placed in parameter words 1 through 6 of the DPB. Fill in this array by using the GETADR routine (see Section 7.4.4).
ids	Directive status

QIO$ - QUEUE I/O REQUEST

Macro Call

```
QIO$   fnc,lun,[efn],[pri],[iost],[ast],[prmlst]
```

fnc	I/O function code (see Appendix A)
lun	Logical unit number
efn	Event flag number
pri	Priority; ignored, but must be present
iost	Address of I/O status block
ast	Address of entry point of AST service routine
prmlst	Parameter list of the form <P1,...P6>

Macro Expansion

```
QIO$    IO.RVB,7,52.,,IOSTAT,IOAST,<IOBUFR,512.>
.BYTE   1,12.       ;QIO$ MACRO DIC, DPB SIZE=12
.WORD   IO.RVB      ;FUNCTION=READ VIRTUAL BLOCK
.WORD   7           ;LOGICAL UNIT NUMBER 7
.BYTE   52.,0       ;EFN 52., PRIORITY IGNORED
.WORD   IOSTAT      ;ADDRESS OF TWO-WORD I/O STATUS BLOCK
.WORD   IOAST       ;ADDRESS OF I/O AST ROUTINE
.WORD   IOBUFR      ;ADDRESS OF DATA BUFFER
.WORD   512.        ;BYTE COUNT=512.
.WORD   0           ;ADDITIONAL PARAMETERS...
.WORD   0           ;...NOT USED IN...
.WORD   0           ;...THIS PARTICULAR...
.WORD   0           ;...INVOCATION OF QUEUE I/O
```

Local Symbol Definitions

Q.IOFN	I/O function code (2)
Q.IOLU	Logical unit number (2)
Q.IOEF	Event flag number (1)
Q.IOPR	Priority (1)
Q.IOSB	Address of I/O status block (2)
Q.IOAE	Address of I/O done AST entry point (2)
Q.IOPL	Parameter list (6 words) (12)

DSW Return Codes

IS.SUC	Successful completion.
IE.UPN	Insufficient dynamic memory.
IE.ULN	Unassigned LUN.
IE.HWR	Device driver not loaded.
IE.PRI	Task other than despooler attempted a write logical block operation.
IE.ILU	Invalid LUN.
IE.IEF	Invalid event flag number (EFN<0 or EFN>64).
IE.ADP	Part of the DPB or I/O status block is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

Notes

1. If the directive call specifies an AST entry point address, the task enters the AST service routine with its stack in the following state:

SP+10	Event flag mask word
SP+06	PS of task prior to AST
SP+04	PC of task prior to AST
SP+02	DSW of task prior to AST
SP+00	Address of I/O status block, or zero if none was specified in the QIO directive

   The address of the I/O status block, which is a trap-dependent parameter, must be removed from the task's stack before an AST Service Exit directive is executed.

2. If the directive is rejected, the specified event flag is not guaranteed to be cleared or set. Consequently, if the task indiscriminately executes a Wait For or Stop For directive and the QIO directive is rejected, the task can wait indefinitely. Care should always be taken to ensure that the directive was successfully completed.

3. Tasks or regions cannot normally be checkpointed with I/O outstanding for two reasons:

   - If the QIO directive results in a data transfer, the data transfers directly to or from the user-specified buffer.

   - If an I/O status block address is specified, the directive status is returned directly to the I/O status block.

   The Executive waits until a task has no outstanding I/O before initiating checkpointing in all cases except the one described below.

   Drivers that buffer I/O check for the following conditions for a task:

   - That the task is checkpointable

   - That checkpointing is enabled

   If those two conditions are met, the driver or the Executive buffers the I/O request internally, and the task is checkpointable with this outstanding I/O. If the task also entered a Wait For state when the I/O was issued (see the QIOW$ directive) or subsequently enters a Wait For state, the task is stopped. Any competing task waiting to be loaded into the partition can checkpoint the stopped task, regardless of priority. If the stopped task is checkpointed, the executive does not bring it back into memory until the stopped state is terminated by completion of buffered I/O or satisfaction of the Wait For condition.

   Not all drivers buffer I/O requests. The terminal driver is an example of one that does.

4. Any task that is linked to a common (read-only) area can issue QIO write requests from that area.

8.46 QIOW$ - QUEUE I/O REQUEST AND WAIT

The Queue I/O Request And Wait directive is identical to the Queue I/O Request in all respects but one. If the Wait variation of the directive specifies an event flag, the Executive automatically effects a Wait For Single Event Flag directive. If an event flag is not specified, however, the Executive treats the directive as if it were a simple Queue I/O Request.

The following description lists the high-level language and macro calls with associated parameters, as well as the macro expansion. Consult the description of Queue I/O Request for a definition of the parameters, the local symbol definitions, the DSW return codes, and explanatory notes.

High-Level Language Call

```
CALL WTQIO (fnc,lun,[efn],[pri],[isb],[prl][,ids])
```

fnc	I/O function code (see Appendix A)
lun	Logical unit number
efn	Event flag number
pri	Priority; ignored, but must be present
isb	A two-word integer array to receive final I/O status
prl	A six-word integer array containing device-dependent parameters to be placed in parameter words 1 through 6 of the DPB
ids	Directive status

Macro Call

```
QIOW$   fnc,lun,[efn],[pri],[iost],[ast][,prmlst]
```

fnc	I/O function code (see Appendix A)
lun	Logical unit number
efn	Event flag number
pri	Priority; ignored, but must be present
iost	Address of I/O status block

ast Address of entry point of AST service routine

prmlst Parameter list of the form <P1,...P6>

Macro Expansion

```
QIOW$  IO.RVB,7,52.,,IOSTAT,IOAST,<IOBUFR,512.>
.BYTE  3,12.          ;QIO$ MACRO DIC, DPB SIZE=12.
.WORD  IO.RVB         ;FUNCTION=READ VIRTUAL BLOCK
.WORD  7              ;LOGICAL UNIT NUMBER 7
.BYTE  52.,0          ;EFN 52., PRIORITY IGNORED
.WORD  IOSTAT         ;ADDRESS OF TWO-WORD I/O STATUS BLOCK
.WORD  IOAST          ;ADDRESS OF I/O AST ROUTINE
.WORD  IOBUFR         ;ADDRESS OF DATA BUFFER
.WORD  512.           ;BYTE COUNT=512.
.WORD  0              ;ADDITIONAL PARAMETERS...
.WORD  0              ;...NOT USED IN...
.WORD  0              ;...THIS PARTICULAR...
.WORD  0              ;...INVOCATION OF QUEUE I/O
```

8.47 RCST$ - RECEIVE DATA OR STOP

The Receive Data Or Stop directive instructs the system to dequeue a 13-word data block for the issuing task; the data block was queued for the task with a Send Data Directive or a Send, Request And Connect directive.

A 2-word task name of the sender (in Radix-50 format) and the 13-word data block are returned in an indicated 15-word buffer. The task name is contained in the first two words of the buffer.

If no data has been sent, the issuing task is stopped. In this case, the sender task is expected to issue an Unstop directive after sending data. A success status code of IS.SUC indicates that a packet has been received. A success status code of IS.SET indicates that the task was stopped and has been unstopped. The directive must then be reissued to retrieve the packet.

When a slave task issues the Receive Data or Stop directive, it assumes the UIC and TI: terminal of the task that sent the data.

High-Level Language Call

```
CALL RCST ([rtname],ibuf[,ids])
```

rtname	Sender task name (if not specified, data can be received from any task)
ibuf	Address of 15-word buffer to receive the sender task name and data
ids	Integer to receive the Directive Status Word

Macro Call

```
RCST$   [tname],buf
```

tname	Sender task name (if not specified, data can be received from any task)
buf	Address of 15-word buffer to receive the sender task name and data

Macro Expansion

```
RCST$     ALPHA,TSKBUF
.BYTE     139.,4          ;RCST$ MACRO DIC, DPB SIZE=4 WORDS
.RAD50    ALPHA           ;DATA SENDER TASK NAME
.WORD     TSKBUF          ;BUFFER ADDRESS
```

Local Symbol Definitions

R.CSTN	Task name (4)
R.CSBF	Buffer address (2)

DSW Return Codes

IS.SUC	Successful completion.
IS.SET	No data was received and task was stopped. (Note that the task must be Unstopped before it can see this status.)
IE.AST	The issuing task is at AST state.
IE.ADP	Part of the DPB is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

Note

The Receive Data Or Stop directive is treated as a 13 (decimal) word Variable Receive Data Or Stop directive.

8.48 RCVD$ - RECEIVE DATA

The Receive Data directive instructs the system to dequeue a 13-word data block for the issuing task. The data block has been queued (FIFO) for the task by a Send Data Directive.

A 2-word sender task name (in Radix-50 form) and the 13-word data block are returned in an indicated 15-word buffer, with the task name in the first two words.

When a slave task issues the Receive Data directive, it assumes the UIC and TI: terminal of the task that sent the data.

High-Level Language Call

```
CALL RECEIV ([tsk],buf[,,ids])
```

tsk — Sender task name. (If not specified, data can be received from any task.)

buf — A 15-word integer array for received data

ids — Directive status

Macro Call

```
RCVD$   [tn],ba
```

tn — Sender task name. (If not specified, data can be received from any task.)

ba — Address of 15-word buffer

Macro Expansion

```
RCVD$     ALPHA,DATBUF    ;TASK NAME AND BUFFER ADDRESS
.BYTE     75.,4           ;RCVD$ MACRO DIC, DPB SIZE=4 WORDS
.RAD50    /ALPHA/         ;SENDER TASK NAME
.WORD     DATBUF          ;ADDRESS OF 15.-WORD BUFFER
```

Local Symbol Definitions

R.VDTN — Sender task name (4)

R.VDBA — Buffer address (2)

DSW Return Codes

IS.SUC	Successful completion.
IE.ITS	No data currently queued.
IE.ADP	Part of the DPB or buffer is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

Notes

1. The Receive Data directive is treated as a 13 (decimal) word Variable Receive Data directive.

2. If the sending task specifies a common event flag in the Send Data directive, the receiving task can use that event flag for synchronization. However, between the time that the receiver issues this directive and the time the receiver issues its next instruction, the sender can send data and set the event flag. If the next instruction is an Exit directive, any data sent during this time will be lost because the Executive flushes the task's receive list as part of exit processing. Therefore, use the Exit If directive or the Receive Data or Exit directive in order to avoid the race condition.

8.49 RCVX$ - RECEIVE DATA OR EXIT

The Receive Data Or Exit directive instructs the system to dequeue a 13-word data block for the issuing task; the data block has been queued (FIFO) for the task by a Send Data directive.

A two-word sender task name (in Radix-50 form) and the 13-word data block are returned in an indicated 15-word buffer, with the task name in the first two words.

If no data has been sent, a task exit occurs. To prevent the possible loss of Send packets, the user should not rely on I/O rundown to take care of any outstanding I/O or open files; the task should assume this responsibility.

When a slave task issues the Receive Data Or Exit directive, it assumes the UIC and TI: of the task that sent the data. (See Notes.)

High-Level Language Call

```
CALL RECOEX ([tsk],buf[,,ids])
```

tsk	Sender task name (If not specified, data can be received from any task.)
buf	A 15-word integer array for received data
ids	Directive status

Macro Call

```
RCVX$   [tn],ba
```

tn	Sender task name (If not specified, data can be received from any task.)
ba	Address of 15-word buffer

Macro Expansion

```
RCVX$       ALPHA,DATBUF    ;TASK NAME AND BUFFER ADDRESS
.BYTE       77.,4           ;RCVX$ MACRO DIC, DPB SIZE=4 WORDS
.RAD50      /ALPHA/         ;SENDER TASK NAME
.WORD       DATBUF          ;ADDRESS OF 15.-WORD BUFFER
```

Local Symbol Definitions

R.VXTN	Sender task name (4)
R.VXBA	Buffer address (2)

DSW Return Codes

IS.SUC	Successful completion.
IE.ADP	Part of the DPB or buffer is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

Notes

1. A high-level language program that issues the RECOEX call must first close all files (for example, by issuing CLOSE calls in FORTRAN).

   To avoid the time overhead involved in the closing and reopening of files, the task should first issue the RECEIV call. If the directive status indicates that no data were received, then the task can close all files and issue the call to RECOEX. The following example illustrates the same overhead saving in MACRO:

```
RCVBUF:   .BLKW   15.           ; Receive buffer

START:    RCVX$C  ,RCVBUF       ; Attempt to receive message
          CALL    OPEN          ; call user subroutine to
                                ; open files.
PROC:               .
                    .
          Process packet of data
                    .
                    .
          RCVD$C  ,RCVBUF       ; Attempt to receive another
                                ; message
          BCC     PROC          ; If CC successful receive
          CALL    CLOSE         ; call user subroutine to
                                ; close files
                                ; and prepare for possible
                                ; task exit
          JMP     START         ; Make one last attempt
                                ; at receiving
```

2. If no data have been sent, that is, if no Send Data directive has been issued, the task exits. Send packets can be lost if a task exits with outstanding I/O or open files. (See third paragraph of this section.)

3. The Receive Data Or Exit directive is useful for avoiding a possible race condition that can occur between two tasks communicating by the Send and Receive directives. The race condition occurs when one task executes a Receive directive and finds its receive queue empty; but before the task can exit, the other task sends it a message. The message is lost because the Executive flushes the receiving task's receive queue when it exits. This condition can be avoided by the receiving task's executing a Receive Data Or Exit directive. If the receive queue is found to be empty, a task exit occurs before the other task can send any data; thus, no loss of data can occur.

4. On Exit, the Executive frees task resources. In particular, the Executive:

   - Detaches all attached devices.
   - Flushes the AST queue and despecifies all specified ASTs.
   - Flushes the receive and receive-by-reference queues.
   - Flushes the clock queue for outstanding Mark Time requests for the task.
   - Closes all open files. (Files open for write access are locked.)
   - Detaches all attached regions except in the case of a fixed task, where no detaching occurs.
   - Runs down the task's I/O.
   - Disconnects from interrupt vectors.
   - Returns a success status (EX$SUC) to any parent tasks.
   - Breaks the connection with any offspring tasks.
   - Frees the task's memory if the exiting task was not fixed.

5. If the task exits, the Executive declares a significant event.

6. The Receive Data Or Exit directive is treated as a 13-word Variable Receive Data Or Exit directive.

8.50 RDAF$ - READ ALL EVENT FLAGS

The Read All Event Flags directive instructs the system to read all 64 event flags for the issuing task and record their polarity in a 64-bit (4-word) buffer.

High-Level Language Call

A high-level language task can read only one event flag. The call is:

```
CALL READEF (efn[,ids])
```

efn — Event flag number

ids — Directive status

The Executive returns the status codes IS.SET (+02) and IS.CLR (00) for high-level language calls in order to report event flag polarity.

Macro Call

```
RDAF$   ba
```

ba — The address of a four-word buffer with the following format:

Word 0 Task local flags 1-16

Word 1 Task local flags 17-32

Word 2 Task common flags 33-48

Word 3 Task common flags 49-64

Macro Expansion

```
RDAF$     FLGBUF
.BYTE     39.,2          ;RDAF$ MACRO DIC, DPB SIZE=2 WORDS
.WORD     FLGBUF         ;ADDRESS OF FOUR-WORD BUFFER
```

Local Symbol Definitions

```
R.DABA     Buffer address (2)
```

DSW Return Codes

IS.SUC	Successful completion.
IE.ADP	Part of the DPB or buffer is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

8.51 RDEF$ - READ EVENT FLAG

The Read Event Flag directive tests an indicated event flag and reports its polarity in the DSW.

High-Level Language Call

```
CALL READEF (iefn[,ids])
```

iefn	Integer containing an Event Flag Number
ids	Integer variable to receive the Directive Status Word

Macro Call

```
RDEF$ efn
```

efn	Event flag number

Macro Expansion

```
RDEF$  6
.BYTE  37.,2
.WORD  6
```

Local Symbol Definitions

The following symbol is locally defined with its assigned value equal to the byte offset from the start of the DPB to the DPB element:

R.DEEF - Event flag number (length of 2 bytes)

DSW Return Codes

IS.CLR	Flag was clear.
IS.SET	Flag was set.
IE.IEF	Invalid event flag number (event flag number <1 or >64).
IE.ADP	Part of DPB is out of issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

8.52 RDXF$ - READ EXTENDED EVENT FLAGS

The Read Extended Event Flags directive instructs the system to read all local and common event flags for the issuing task and record their polarity in a 96-bit (6-word) buffer.

High-Level Language Call

A high-level language task can read only one event flag. The call is:

```
CALL READEF (efn[,ids])
```

efn Event flag number

ids Directive status

The Executive returns the status codes IS.SET (+02) and IS.CLR (00) for high-level language calls to report event flag polarity.

Macro Call

```
RDXF$   ba
```

ba Address of a six-word buffer with the following format:

Word	0	Task local flags 1-16
Word	1	Task local flags 17-32
Word	2	Task common flags 33-48
Word	3	Task common flags 49-64
Word	4	Reserved
Word	5	Reserved

Macro Expansion

```
RDXF$     FLGBUF
.BYTE     39.,3          ;RDXF$ MACRO DIC, DPB SIZE=3 WORDS
.WORD     FLGBUF         ;ADDRESS OF SIX-WORD BUFFER
```

Local Symbol Definitions

R.DABA	Buffer address (2)

DSW Return Codes

IS.SUC	Successful completion.
IS.CLR	Group global event flags do not exist. Words 4 and 5 of the buffer contain 0.
IE.ADP	Part of the DPB or buffer is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

8.53 RPOI$ - REQUEST AND PASS OFFSPRING INFORMATION

The Request and Pass Offspring Information directive instructs the system to request the specified task, and to chain to it by passing any or all of the parent connections from the issuing task to the requested task. Optionally, the directive can pass a command line to the requested task. Only a privileged task can specify the UIC and TI: of the requested task.

High-Level Language Call

```
CALL RPOI (tname,[iugc],[iumc],[iparen],[ibuf],[ibfl],[isc],
           [idnam],[iunit],[itask],[ocbad][,ids])
```

tname — Name of an array containing the actual name (in Radix-50) of the task to be requested and optionally chained to.

iugc — Name of an integer containing the group code number for the UIC of the requested target chain task.

iumc — Name of the integer containing the member code number for the UIC of the requested target chain task.

iparen — Name of an array (or I*4 integer) containing the Radix-50 name of the parent task. This name is returned in the information buffer of the GTCMCI subroutine.

ibuf — Name of an array that contains the command line text for the chained task.

ibfl — Name of an integer that contains the number of bytes in the command in the ibuf array.

isc — Flag byte controlling the actions of this directive request when executed. The bit definitions of this byte are as follows (only the low order byte of the integer specified in the call is ever used):

RP.OEX = 128. — Force this task to exit on successful execution of the RPOI directive.

RP.OAL = 1 — Pass all of this task's connections to the requested task. (The default is none.)

RP.ONX = 2 Pass the first connection in the queue if there is one.

idnam Name of an integer that contains the ASCII device name of the requested task's TI:

iunit Name of an integer that contains the unit number of the requested task's TI: device.

itask Name of an array which contains the Radix-50 name the requested task is to run under.

Any task can specify a new name for the requested task. However, the requested task (specified in the tname parameter) must be installed with the ...tsk format.

ocbad Reserved for future use.

ids Name of an integer to receive the Directive Status Word.

Macro Call

```
RPOI$   tname,pn,pr,[ugc],[umc],[parent],[bufadr],[buflen]
        ,[sc],[dnam],[unit],[task],[ocbad]
```

tname Name of task to be chained to

pn Partition name--ignored, but must be present

pr Priority--ignored, but must be present

ugc Group code for UIC of the requested task

umc Member code for UIC of the requested task

parent Name of issuing task's parent task whose connection is passed. If not specified, all connections are passed.

bufadr Address of buffer to be given to the requested task

buflen Length of buffer to be given to requested task

sc Flags byte:

RP.OEX (200) Force issuing task to exit.

RP.OAL	(1)	Pass all connections. (The default is none.)
RP.ONX	(2)	Pass the first connection in the queue, if there is one.

dnam — ASCII device name for TI:.

unit — Unit number of task TI:.

task — Radix-50 name under which the requested task is to run.

Any task can specify a new name for the requested task. However, the requested task (specified in the tname parameter) must be installed with the ...tsk format.

ocbad — Reserved for future use.

Local Symbol Definitions

R.POTK	Radix-50 name of installed task to be chained to (2)
(reserved)	Six unused words (6)
R.POUM	UIC member code (1)
R.POUG	UIC group code
R.POPT	Name of parent whose OCB should be passed (4)
R.POOA	Reserved for future use (2)
R.POBF	Address of command buffer (2)
R.POBL	Length of command (2)
R.POUN	Unit number of task TI: (1)
R.POSC	Flags byte (1)
R.PODV	ASCII device name for TI: (2)
R.POTN	Radix-50 name of task to be started (4)

Macro Expansion

```
RPOI$   tname,,,ugc,umc,ptsk,buf,buflen,sc,dev,unit,task,ocbad
.BYTE   11,16       ;DIC 11 DPB SIZE = 16. words
.RAD50  /tname/     ;NAME OF TASK TO CHAIN TO
.BLKW   3           ;RESERVED
.BYTE   umc         ;UIC MEMBER CODE
.BYTE   ugc         ;UIC GROUP CODE
.RAD50  ptsk        :NAME OF TASK WHOSE OCB SHOULD BE PASSED
.WORD   ocbad       ;ADDRESS OF OCB
.WORD   buf         ;ADDRESS OF BUFFER TO SEND
.WORD   buflen      ;LENGTH OF BUFFER
.ASCII  /dev/       ;ASCII NAME OF TI: OF REQUESTED TASK
.BYTE   unit        ;UNIT NUMBER OF TI: DEVICE
.BYTE   sc          ;PASS BUFFER AS SEND PACKET OR COMMAND CODE
.RAD50 /task/       ;NEW NAME OF TASK TO START
```

DSW Return Codes

IE.UPN — There is insufficient dynamic memory to allocate an Offspring Control Block, command line buffer, Task Control Block, or Partition Control Block.

IE.INS — The specified task was not installed, or it was a CLI but no command line was specified.

IE.ACT — The specified task was already active and it was not a command line interpreter.

IE.IDU — The specified virtual terminal unit does not exist, or was not created by the issuing task.

IE.NVR — There is no Offspring Control Block from the specified parent task.

IE.ALG — Either a parent name or an offspring control block address was specified, and the pass all connections flag or the pass next connection flag was set.

IE.PNS — The Task Control Block cannot be created in the same partition as its prototype.

IE.ADP — Part of the DPB, exit status block, or command line is out of the issuing task's address space.

IE.SDP — DIC or DPB size is invalid.

8.54 RQST$ - REQUEST TASK

The Request Task directive instructs the system to activate a task. The task is activated and runs contingent upon priority and memory availability. The Request Task directive is the basic mechanism used by running tasks to initiate other installed (dormant) tasks. The Request Task directive is a frequently used subset of the Run directive. See Notes.

High-Level Language Call

```
CALL REQUES (tsk,[opt][,ids])
```

tsk	Task name
opt	A four-word integer array
opt(1)	First half of partition name; ignored, but must be present
opt(2)	Second half of partition name; ignored, but must be present
opt(3)	Priority; ignored, but must be present
opt(4)	User Identification Code
ids	Directive status

Macro Call

```
RQST$   tn,[pn],[pr][,gc,p]
```

tn	Task name
pn	Partition name; ignored, but must be present
pr	Priority; ignored, but must be present
gc	UIC group code
p	UIC member code

Macro Expansion

```
RQST$     ALPHA,,,20,10
.BYTE     11.,7           ;RQST$ MACRO DIC, DPB SIZE=7 WORDS
.RAD50    /ALPHA/         ;TASK "ALPHA"
.WORD     0,0             ;PARTITION IGNORED
.WORD     0               ;PRIORITY IGNORED
```

```
.BYTE     10,20          ;UIC UNDER WHICH TO RUN TASK
```

Local Symbol Definitions

R.QSTN	Task name (4)
R.QSPN	Partition name (4)
R.QSPR	Priority (2)
R.QSPC	UIC member (1)
R.QSGC	UIC group (1)

DSW Return Codes

IS.SUC	Successful completion.
IE.UPN	Insufficient dynamic memory.
IE.INS	Task is not installed.
IE.ACT	Task is already active.
IE.ADP	Part of the DPB is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

Notes

1. The requested task must be installed in the system.

2. If the partition in which a requested task is to run is already occupied, the Executive places the task in a queue of tasks waiting for that partition. The requested task then runs, depending on priority and resource availability when the partition is free.

   If the current occupant of the partition is checkpointable, has checkpointing enabled, and is of lower priority than the requested task, it is written to disk when its current outstanding I/O completes; the requested task is then read into the partition.

3. Successful completion means that the task has been declared active, not that the task is actually running.

4. The requested task acquires the same TI: assignment as that of the requesting task.

5. The requested task always runs at the priority specified in its task header.

6. A task that executes in a system-controlled partition requires dynamic memory for the Partition Control Block used to describe its memory requirements.

7. Each active task has two UICs: a protection UIC and a default UIC. These are both returned when a task issues a Get Task Parameters directive (GTSK$). The UICs are used in the following ways:

   - The protection UIC determines the task's access rights for opening files and attaching to regions. When a task attempts to open a file, the system compares the task's protection UIC against the protection mask of the specified UFD. The comparison determines whether the task is to be considered for system, owner, group, or world access.

   - The default UIC is used by the Record Management Services (RMS) to determine the default UFD when a file-open operation does not specify a UIC. (The default UIC has no significance when a task attaches to a region.)

8.55 RREF$ - RECEIVE BY REFERENCE

The Receive By Reference directive requests the Executive to dequeue the next packet in the receive-by-reference queue of the issuing (receiver) task. If successful, the directive declares a significant event.

Optionally, the task exits if there are no packets in the queue. The directive can also specify that the Executive proceed to map the referred region.

Each reference in the task's receive-by-reference queue represents a separate attachment to a region. If a task has multiple references to a given region, it is attached to that region the corresponding number of times. Because region attachment requires system dynamic memory, the receiver task should detach from any region that it was already attached to in order to prevent depletion of the memory pool. That is, the task needs to be attached to a given region only once.

If the Executive does not find a packet in the queue, and the task has set WS.RCX in the window status word (W.NSTS), the task exits. If WS.RCX is not set, the Executive returns the DSW code IE.ITS.

If the Executive finds a packet, it writes the information provided to the corresponding words in the Window Definition Block. This information provides sufficient information to map the reference to a previously created address window, according to the sender task's specifications.

If the address of a ten-word receive buffer has been specified (W.NSRB in the Window Definition Block), then the sender task name and the eight additional words passed by the sender task (if any) are placed in the specified buffer. If the sender task did not pass any additional information, the Executive writes the sender task name and eight words of zero.

If the WS.MAP bit in the window status word has been set to 1, the Executive transfers control to the Map Address Window directive to attempt to map the reference.

When a task that has unreceived packets in its receive-by-reference queue exits or is removed, the Executive removes the packets from the queue and deallocates them. Related flags are not set.

High-Level Language Call

```
CALL RREF (iwdb,[isrb][,ids])
```

iwdb	An eight-word integer array that contains a Window Definition Block (see Section 5.5.2.2)
isrb	A ten-word integer array to be used as the receive buffer. If the call omits this parameter, the contents of iwdb(8) are unchanged.
ids	Directive status

Macro Call

```
RREF$   wdb
```

wdb	Window Definition Block address

Macro Expansion

```
RREF$    WDBADR
.BYTE    81.,2       ;RREF$ MACRO DIC, DPB SIZE=2 WORDS
.WORD    WDBADR      ;WDB ADDRESS
```

Definition Block Parameters

Table 8-10 describes the Window Definition Block parameters for this directive.

Local Symbol Definitions

R.REBA	Window Definition Block address (2)

DSW Return Codes

IS.SUC	Successful completion.
IS.HWR	Region has incurred a parity error.
IE.ITS	No packet found in the receive-by-reference queue.
IE.ADP	Address check of the DPB, WDB, or the receive buffer (W.NSRB) failed.
IE.SDP	DIC or DPB size is invalid.

Table 8-10: Window Definition Block Parameters, RREF$ and RRST$

Array Element	Offset	Description
Input Parameters		
iwdb(1) bits 0-7	W.NID	ID of an existing window if region is to be mapped
iwdb(7)	W.NSTS	Bit settings* in the window status word: **Bit** — **Definition** WS.MAP — 1 if received reference is to be mapped WS.RCX — 1 if task exit desired when no packet is found in the queue
iwdb(8)	W.NSRB	Optional address of a 10-word buffer to contain the sender task name and additional information
Output Parameters		
iwbd(4)	W.NRID	Region ID (pointer to attachment description)
iwdb(5)	W.NOFF	Offset word specified by sender task
iwdb(6)	W.NLEN	Length word specified by sender task

* If you are a high-level language programmer, see Section 5.5.2 to determine the bit values represented by the symbolic names described.

Array Element	Offset	Description
iwdb(7)	W.NSTS	Bit settings* in the window status word: **Bit** **Definition** WS.RED 1 if attached with read access WS.WRT 1 if attached with write access WS.EXT 1 if attached with extend access WS.DEL 1 if attached with delete access WS.RRF 1 if receive was successful The Executive clears the remaining bits.

8.56 RRST$ - RECEIVE BY REFERENCE OR STOP

The Receive By Reference or Stop directive requests the Executive to dequeue the next packet in the receive-by-reference queue of the issuing (receiver) task. The task stops if there are no packets in the queue. The directive can also specify that the Executive proceed to map the referenced region.

If successful, the directive declares a significant event.

Each reference in the task's receive-by-reference queue represents a separate attachment to a region. If a task has multiple references to a given region, it is attached to that region the corresponding number of times. Because region attachment requires system dynamic memory, the receiver task should detach from any region that it was already attached to in order to prevent depletion of the memory pool. That is, the task needs to be attached to a given region only once.

If the Executive finds a packet, it writes the information provided to the corresponding words in the Window Definition Block. This information provides sufficient information to map the reference, according to the sender task's specifications, to a previously created address window.

If the address of a ten-word receive buffer has been specified (W.NSRB in the Window Definition Block), then the sender task name and the eight additional words passed by the sender task (if any) are placed in the specified buffer. If the sender task did not pass any additional information, the Executive writes the sender task name and eight words of zero.

If the WS.MAP bit in the window status word has been set to 1, the Executive transfers control to the Map Address Window directive to attempt to map the reference.

When a task that has unreceived packets in its receive-by-reference queue exits or is removed, the Executive removes the packets from the queue and deallocates them. Any related flags are not set.

High-Level Language Call

```
CALL RRST (iwdb,[isrb][,ids])
```

iwdb — An eight-word integer array that contains a Window Definition Block (see Section 5.5.2.2)

isrb — A ten-word integer array to be used as the receive buffer. If the call omits this parameter, the contents of iwdb(8) are unchanged.

ids — Directive status

Macro Call

```
RRST$  wdb
```

wdb — Window Definition Block address

Macro Expansion

```
RRST$    WDBADR
.BYTE    213.,2      ;RRST$ MACRO DIC, DPB SIZE=2 WORDS
.WORD    WDBADR      ;WDB ADDRESS
```

Definition Block Parameters

Table 8-10 describes the Window Definition Block parameters for this directive.

Local Symbol Definitions

R.REBA — Window Definition Block address (2)

DSW Return Codes

IS.SUC — Successful completion.

IS.HWR — Region has incurred a parity error.

IE.ITS — No packet found in the receive-by-reference queue.

IE.ADP — Address check of the DPB, WDB, or the receive buffer (W.NSRB) failed.

IE.SDP — DIC or DPB size is invalid.

8.57 RSUM$ - RESUME TASK

The Resume Task directive instructs the system to resume the execution of a task that has issued a Suspend directive.

High-Level Language Call

```
CALL RESUME (tsk[,ids])
```

tsk Task name

ids Directive status

Macro Call

```
RSUM$ tn
```

tn Task name

Macro Expansion

```
RSUM$    ALPHA
.BYTE    47.,3          ;RSUM$ MACRO DIC, DPB SIZE=3 WORDS
.RAD50   /ALPHA/        ;TASK "ALPHA"
```

Local Symbol Definitions

R.SUTN Task name (4)

DSW Return Codes

IS.SUC Successful completion.

IE.INS Task is not installed.

IE.ACT Task is not active.

IE.ITS Task is not suspended.

IE.ADP Part of the DPB is out of the issuing task's address space.

IE.SDP DIC or DPB size is invalid.

8.58 RUN$ - RUN TASK

The Run Task directive causes a task to be requested at a specified time, and optionally to be requested periodically. The scheduled time is specified in terms of delta time from issuance. If the sm, rm, and ru parameters are omitted, Run is the same as Request Task, except that:

1. Run causes the task to become active one clock tick after the directive is issued.

2. The system always sets the TI: device for the requested task to CO:.

See Notes.

High-Level Language Call

```
CALL RUN (tsk,[opt],[sm],su,[rm],[ru][,ids])
```

tsk	Task name
opt	A four-word integer array
opt(1)	First half of partition name; ignored, but must be present
opt(2)	Second half of partition name; ignored, but must be present
opt(3)	Priority; ignored, but must be present
opt(4)	User Identification Code
sm	Schedule delta magnitude
su	Schedule delta unit (either 1, 2, 3, or 4)
rm	Reschedule interval magnitude
ru	Reschedule interval unit
ids	Directive status

The ISA standard call for initiating a task is also provided:

```
CALL START (tsk,sm,su[,ids])
```

tsk	Task name

sm	Schedule delta magnitude
su	Schedule delta unit (either 0, 1, 2, 3, or 4)
ids	Directive status

Macro Call

RUN$ tn,[pn],[pr],[gc],[p],[sm],su[,rm,ru]

tn	Task name
pn	Partition name; ignored, but must be present
pr	Priority; ignored, but must be present
gc	UIC group code
p	UIC member code
sm	Schedule delta magnitude
su	Schedule delta unit (either 1, 2, 3, or 4)
rm	Reschedule interval magnitude
ru	Reschedule interval unit

Macro Expansion

```
RUN$     ALPHA,,,20,10,20.,3,10.,3
BYTE     17.,11.     ;RUN$ MACRO DIC, DPB SIZE=11.  WORDS
.RAD50   /ALPHA/     ;TASK "ALPHA"
.WORD    0,0         ;PARTITION IGNORED
.WORD    0           ;PRIORITY IGNORED
.BYTE    10,20       ;UIC TO RUN TASK UNDER
.WORD    20.         ;SCHEDULE MAGNITUDE=20
.WORD    3           ;SCH. DELTA TIME UNIT=MINUTE (=3)
.WORD    10.         ;RESCH. INTERVAL MAGNITUDE=10.
.WORD    3           ;RESCH. INTERVAL UNIT=MINUTE (=3)
```

Local Symbol Definitions

R.UNTN	Task name (4)
R.UNPN	Partition name (4)
R.UNPR	Priority (2)
R.UNGC	UIC group code (1)

R.UNPC	UIC member code (1)
R.UNSM	Schedule magnitude (2)
R.UNSU	Schedule unit (2)
R.UNRM	Reschedule magnitude (2)
R.UNRU	Reschedule unit (2)

DSW Return Codes

For CALL RUN and RUN$:

IS.SUC	Successful completion.
IE.UPN	Insufficient dynamic memory.
IE.ACT	Multiuser task name specified.
IE.INS	Task is not installed.
IE.PRI	Nonprivileged task specified a UIC other than its own.
IE.ITI	Invalid time parameter.
IE.ADP	Part of the DPB is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

For CALL START:

The system provides the following positive error codes to be returned for ISA calls:

2	Insufficient dynamic storage.
3	Specified task not installed.
94	Invalid time parameter.
98	Invalid event flag number.
99	Part of DPB is out of task's address space.
100	DIC or DPB size is invalid.

Notes

1. A nonprivileged task cannot specify a UIC that is not equal to its own protection UIC. A privileged task can specify any UIC.

2. The target task must be installed in the system.

3. If there is not enough room in the partition in which a requested task is to run, the Executive places the task in a queue of tasks waiting for that partition. The requested task then runs, depending on priority and resource availability, when the partition is free. Another possibility is that checkpointing can occur. If the current occupant of the partition is checkpointable, has checkpointing enabled, is of lower priority than the requested task, or is stopped for terminal input, it is written to disk when its current outstanding I/O completes. The requested task is then read into the partition.

4. Successful completion means the task has been made active; it does not mean that the task is actually running.

5. The Executive returns the code IE.ITI in the DSW if the directive specifies an invalid time parameter. A time parameter consists of the time interval magnitude, and the time interval unit.

   A legal magnitude value (sm or rm) is related to the value assigned to the time interval unit su or ru.

   The unit values are encoded as follows:

   For an ISA FORTRAN call (CALL START):

0	Ticks - A tick occurs for each clock interrupt tick rate of 64 (decimal) ticks per second.
1	Milliseconds - The subroutine converts the specified magnitude to the equivalent number of system clock ticks.

   For all other high-level language and all macro calls:

1	Ticks - A tick occurs for each clock interrupt tick rate of 64 (decimal) ticks per second.

   For all calls:

2	Seconds

3 Minutes

4 Hours

8.59 SCAA$ - SPECIFY COMMAND ARRIVAL AST

The Specify Command Arrival AST directive instructs the system to enable or disable command arrival ASTs for the issuing CLI task. If command arrival ASTs are enabled, the Executive transfers control to a specified address when commands are queued to the CLI.

Only CLI tasks can use this AST.

When the AST routine is entered, the format of the stack is as follows:

```
SP+10 - Zero since no event flags are involved
SP+06 - PS of task prior to AST
SP+04 - PC of task prior to AST
SP+02 - DSW of task prior to AST
SP+00 - Address of command buffer just queued
```

The AST routine must remove the command buffer address from the stack before issuing an ASTX$ directive.

The command buffer address can be used when issuing a GCCI$ directive.

High-Level Language Call

Not supported

Macro Call

```
SCAA$  [ast]
```

ast — AST service-routine entry point. Omitting this parameter disables command arrival ASTs for the issuing task until the directive is respecified.

Macro Expansion

```
SCAA$   ast
.BYTE   173.,2      ;DIC = 173 (DECIMAL), DPB SIZE = 2 WORDS
.WORD   ast         ;ADDRESS OF AST ROUTINE
```

Local Symbol Definitions

```
S.CAAE      Address of AST routine
```

DSW Return Codes

IE.ITS	ASTs are already not desired.
IE.AST	Directive issued from AST state.
IE.PRV	Issuing task is not a CLI.
IE.UPN	Insufficient dynamic memory.
IE.ADP	Part of the DPB was out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

8.60 SDAT$ - SEND DATA

The Send Data directive instructs the system to declare a significant event and to queue (FIFO) a 13-word block of data for a task to receive.

NOTE

When a local event flag is specified, the indicated event flag is set for the sending task. A significant event is always declared.

High-Level Language Call

```
CALL SEND (tsk,buf,[efn][,ids])
```

tsk	Task name
buf	13-word integer array of data to be sent
efn	Event flag number
ids	Directive status

Macro Call

```
SDAT$   tn,ba[,efn]
```

tn	Task name
ba	Address of 13-word data buffer
efn	Event flag number

Macro Expansion

```
SDAT$     ALPHA,DATBUF,52.
.BYTE     71.,5       ;SDAT$ MACRO DIC, DPB SIZE=5 WORDS
.RAD50    /ALPHA/     ;RECEIVER TASK NAME
.WORD     DATBUF      ;ADDRESS OF 13.-WORD BUFFER
.WORD     52.         ;EVENT FLAG NUMBER 52.
```

Local Symbol Definitions

S.DATN	Task name (4)
S.DABA	Buffer address (2)
S.DAEF	Event flag number (2)

DSW Return Codes

IS.SUC	Successful completion.
IE.INS	Receiver task is not installed.
IE.UPN	Insufficient dynamic memory.
IE.IEF	Invalid event flag number (EFN<0 or EFN>64).
IE.ADP	Part of the DPB or data block is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

Notes

1. Send Data requires dynamic memory.

2. If the directive specifies a local event flag, the flag is local to the sender (issuing) task.

   Normally, the event flag is used to trigger the receiver task into some action. For this purpose, the event flag must be common (33 through 64) rather than local. (Refer to the descriptions of the Receive Data directive and the Exit IF directive.)

3. The Send Data directive is treated as a 13-word Variable Send Data directive.

8.61 SDIR$ - SET-UP DEFAULT DIRECTORY STRING

The Set-Up Default Directory String directive establishes a default directory string for the user whose task calls the directive. (See the Get Default Directory directive.)

High-Level Language Call

```
CALL SETDDS  (mod,iens,ienssz,[idsw])
```

mod	The modifier of the logical name within a table. See Section 2.2 for details.
iens	Character array that contains the equivalence name string
ienssz	Size (in bytes) of the equivalence name string
idsw	Integer to receive the Directive Status Word

Macro Call

```
SDIR$  mod,ens,enssz
```

mod	The modifier of the logical name within a table. See Section 2.2 for details.
ens	Buffer address of the equivalence name string
enssz	Size in bytes of the equivalence name string

Macro Expansion

```
SDIR$   mod,ens,enssz
.BYTE   207.,5          ;SDIR$ MACRO DIC, DPB SIZE = 5 WORDS
.BYTE   3               ;SUBFUNCTION CODE FOR SET DEFAULT
                        ;DIRECTORY
.BYTE   MOD             ;LOGICAL NAME MODIFIER
.WORD   0               ;RESERVED
.WORD   ENS             ;BUFFER ADDRESS OF EQUIVALENCE NAME
.WORD   ENSSZ           ;BYTES COUNT OF EQUIVALENCE STRING
```

Local Symbol Definitions

S.DENS	Address of equivalence name buffer (2)
S.DESZ	Byte count of equivalence name string (2)
S.DFUN	Subfunction code (1)

S.DMOD Logical name modifer (1)

DSW Return Codes

IS.SUC Successful completion of service.

IS.SUP Successful completion of service. A new equivalence name string superseded a previously specified name string.

IE.UPN Insufficient dynamic storage is available to create the logical name.

IE.IBS The length of the logical or equivalence string is invalid. Each string length must be greater than 0 but not greater than 255 (decimal) characters.

IE.ADP Part of the DPB or user buffer is out of the issuing task's address space, or the user does not have proper access to that region.

IE.SDP DIC or DPB size is invalid.

Notes

1. If you specify 0 for argument iens or ens, the system deletes the default directory string for the user who spawned the task that called the SDIR$ directive. The system deletes the equivalence that has the specified mod value.

2. SDIR$ does not check whether or not the directory you specify does in fact exist. No error status code is returned if the directory does not exist.

8.62 SDRC$ - SEND, REQUEST AND CONNECT

The Send, Request And Connect directive performs a Send Data to the specified receiver task. This action causes the Executive to declare a significant event and to queue (FIFO) a 13-word block of data for the receiver task. The SDRC$ directive then requests the receiver task for execution if the task is not already active. It then connects to the receiver task (making it an offspring of the sender) by queuing an Offspring Control Block (OCB) to the receiver task's Task Control Block (TCB) and incrementing the rundown count in the sending (parent) task's TCB.

The rundown count is used to inform the Executive that the parent task has one or more offspring tasks; clean-up is necessary if the parent task exits while offspring tasks are still active. The rundown count is decremented when the offspring task exits. The OCB contains the TCB address as well as sufficient information to effect all of the specified exit events when the offspring task exits.

For high-level languages, call SDRCN instead of SDRC when you do not use ASTs. Using SDRCN stops the system from bringing an additional impure area into your task root, thus saving virtual address space. The interface routines would normally use the additional impure area to save context during an AST.

If an AST address is specified and you do not use SDRCN to invoke the directive, an exit AST routine is effected when the offspring task exits with the address of the task's exit status block on the stack. The AST routine must remove this word from the stack before issuing the AST Service Exit directive.

High-Level Language Call

```
CALL SDRC (rtname, ibuf,[iefn],[iast],[iesb],[iparm][,ids])

CALL SDRCN (rtname, ibuf,[iefn],[iast],[iesb],[iparm][,ids])
```

rtname	Target task name of the offspring task to be connected
ibuf	Name of 13-word send buffer
iefn	Event flag to be set when the offspring task exits or emits status
iast	Name of an AST routine called when the offspring task exits or emits status. This parameter is ignored when you call SDRCN.

NOTE

Refer to Section 7.4.7 for important guidelines on using high-level language AST service routines.

iesb — Name of an eight-word status block to be written when the offspring task exits or emits status

Word	0	Offspring task exit status
Word	1	System abort code
Word	2-7	Reserved

NOTE

The exit status block defaults to one word. To use the eight-word exit status block, you must specify the logical OR of the symbol SP.WXB and the event flag number in the iefn parameter above.

iparm — Name of a word to receive the status block address when an AST occurs

ids — Integer to receive the Directive Status Word

Macro Call

```
SDRC$   tname,buf,[efn],[east],[esb]
```

tname — Target task name of the offspring task to be connected

buf — Address of 13-word send buffer

efn — The event flag to be cleared on issuance and set when the offspring task exits or emits status

east — Address of an AST routine to be called when the offspring task exits or emits status

esb — Address of an eight-word status block to be written when the offspring task exits or emits status

Word	0	Offspring task exit status
Word	1	System abort code
Word	2-7	Reserved

NOTE

The exit status block defaults to one word. To use the eight-word exit status block, you must specify the logical OR of the symbol SP.WXB and the event flag number in the efn parameter above.

Macro Expansion

```
SDRC$     ALPHA,BUFFR,2,SDRCTR,STBLK
.BYTE     141.,7          ;SDRC$ MACRO DIC, DPB SIZE=7 WORDS
.RAD50    ALPHA           ;TARGET TASK NAME
.WORD     BUFFR           ;SEND BUFFER ADDRESS
.BYTE     2               ;EVENT FLAG NUMBER = 2
.BYTE     16.             ;EXIT STATUS BLOCK CONSTANT
.WORD     SDRCTR          ;ADDRESS OF AST ROUTINE
.WORD     STBLK           ;ADDRESS OF STATUS BLOCK
```

Local Symbol Definitions

S.DRTN	Task name (4)
S.DRBF	Buffer address (2)
S.DREF	Event flag (2)
S.DREA	AST routine address (2)
S.DRES	Status block address (2)

DSW Return Codes

IS.SUC	Successful completion.
IE.UPN	Insufficient dynamic memory to allocate a send packet, Offspring Control Block, Task Control

Block, or Partition Control Block.

IE.INS	The specified task is an ACP or has the no-send attribute.
IE.IEF	An invalid event flag number was specified (EFN<0 or EFN>64).
IE.ADP	Part of the DPB or exit status block is not in the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

Notes

1. The virtual mapping of the exit status block should not be changed while the connection is in effect. Doing so can result in errors.

2. If the directive is rejected, the state of the specified event flag is indeterminate.

8.63 SDRP$ - SEND DATA REQUEST AND PASS OCB

This directive instructs the system to send a send data packet for the specified task, chain to the requested task, and request it if it is not already active.

High-Level Language Call

```
CALL SDRP (task,ibuf,[ibfl],[iefn],[iflag],[iparen],[iocbad]
          [,ids])
```

task — Name of an array (REAL,INTEGER,I*4) that contains the Radix-50 name of the target task

ibuf — Name of an integer array that contains the data to be sent

ibfl — Name of an integer that contains the number of words (integers) in the array to be sent. This argument can be in the range of 1 through 256 (decimal). If this argument is not specified, a default value of 13 (decimal) is assumed.

iefn — Name of an integer that contains the number of the event flag that is to be set when this directive is executed successfully.

iflag — Name of an integer that contains the flag bits controlling execution of this directive. They are defined as follows:

- SD.REX = 128. — Force this task to exit upon successful execution of this directive.
- SD.RAL = 1 — Pass all connections to the requested task. (Default is pass none.) If you specify this flag, do not specify the parent task name.
- SD.RNX = 2 — Pass the first connection in the queue, if there is one, to the requested task. If you specify this flag, do not specify the parent task name.

iparen — Name of an array that contains the Radix-50 name of the parent task whose connection should be

passed to the target task. The name of the parent task was returned in the information buffer of the GTCMCI subroutine.

iocbad — Reserved for future use.

ids — Name of an integer to receive the contents of the Directive Status Word.

Macro Call

```
SDRP$ task,bufadr,[buflen],[efn],[flag],[parent],[ocbad]
```

task — Name of task to be chained to

bufadr — Address of buffer to be given to the requested task

buflen — Length of buffer to be given to requested task

efn — Event flag

flag — Flags byte controlling execution of this directive. The flag bits are defined as follows:

- SD.REX = 128. Force this task to exit upon successful completion of this directive.
- SD.RAL = 1 Pass all connections to the requested task. (Default is pass none.) If you specify this flag, do not specify the parent task name.
- SD.RNX = 2 Pass the first connection in the queue, if there is one, to the requested task. If you specify this flag, do not specify the parent task name.

parent — Name of issuing task's parent task whose connection is passed. If not specified, all connections or no connections are passed depending on the flag byte.

ocbad — Reserved for future use.

Macro Expansion

```
SDRP$ task,bufadr,[buflen],[efn],[flag],[parent],[ocbad]

.BYTE   141.,9.          ;DIC = 141, DPB LENGTH =9 WORDS
.RAD50  /task/           ;TASK NAME IN RADIX-50
.WORD   BUFADR           ;BUFFER ADDRESS
.BYTE   EFN,FLAG         ;EVENT FLAG, FLAGS BYTE
.WORD   BUFLEN           ;BUFFER LENGTH
.RAD50  /PARENT/         ;PARENT TASK NAME
.WORD   OCBAD            ;ADDRESS OF OCB
```

Local Symbol Definitions

S.DRTK — Radix-50 name of task to be chained to

S.DRAD — Send data buffer address

S.DREF — Event flag

S.DRFL — Flags byte:

- SD.REX - (200) Force task to exit (task issuing directive).
- SD.RAL - (1) Pass all connections to the requested task (default is pass none). If you specify this flag, do not specify the parent task name.
- SD.RNX - (2) Pass the first connection in the queue, if there is one, to the requested task. If you specify this flag, do not specify the parent task name.

S.DRBL — Length of send data packet (256 decimal words maximum)

S.DRPT — Name of parent whose OCB should be passed

S.DROA — Reserved for future use

DSW Return Codes

IE.NVR — No Offspring Control Block from specified parent.

IE.ALG — Either a parent name or an OCB address was specified, and the pass all connections flag was set.

IE.IBS Length of send packet is illegal. The send packet can be a maximum of 256 (decimal) bytes long.

IE.UPN Insufficient dynamic memory to allocate a send packet, Offspring Control Block, Task Control Block, or partition control block.

IE.INS The specified task is an ACP or has the no-send attribute.

IE.IEF An invalid event flag number was specified (EFN <0 or EFN >64).

IE.ADP Part of the DPB or exit status block is out of the issuing task's address space.

IE.SDP DIC or DPB size is invalid.

Note

If the directive is rejected, the state of the specified event flag is indeterminate.

8.64 SETF$ - SET EVENT FLAG

The Set Event Flag directive instructs the system to set an indicated event flag that reports the flag's polarity before setting.

High-Level Language Call

```
CALL SETEF (efn[,ids])
```

efn	Event flag number
ids	Directive status

Macro Call

```
SETF$   efn
```

efn	Event flag number

Macro Expansion

```
SETF$     52.
.BYTE     33.,2          ;SETF$ MACRO DIC, DPB SIZE=2 WORDS
.WORD     52.            ;EVENT FLAG NUMBER 52.
```

Local Symbol Definitions

S.ETEF	Event flag number (2)

DSW Return Codes

IS.CLR	Flag was clear.
IS.SET	Flag was already set.
IE.IEF	Invalid event flag number (EFN<1 or EFN>64).
IE.ADP	Part of the DPB is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

Note

Set Event Flag does not declare a significant event. It merely sets the specified flag.

8.65 SFPA$ - SPECIFY FLOATING POINT PROCESSOR EXCEPTION AST

The Specify Floating Point Processor Exception AST directive instructs the system to record one of the two following cases:

- Floating Point Processor exception ASTs for the issuing task are desired, and the Executive is to transfer control to a specified address when such an AST occurs for the task.
- Floating Point Processor exception ASTs for the issuing task are no longer desired.

When an AST service routine entry point address is specified, subsequent Floating Point Processor exception ASTs occur for the issuing task, and control will be transferred to the indicated location at the time of the AST's occurrence. When an AST service entry point address is not specified, subsequent Floating Point Processor exception ASTs do not occur until the task issues a directive that specifies an AST entry point. See Notes.

High-Level Language Call

Not supported

Macro Call

```
SFPA$    [ast]
```

ast — AST service routine entry point address

Macro Expansion

```
SFPA$     FLTAST
.BYTE     111.,2          ;SFPA$ MACRO DIC, DPB SIZE=2 WORDS
.WORD     FLTAST          ;ADDRESS OF FLOATING POINT AST
```

Local Symbol Definitions

S.FPAE — AST entry address (2)

DSW Return Codes

IS.SUC — Successful completion.

IE.UPN — Insufficient dynamic memory.

IE.ITS — AST entry point address is already unspecified or task was built without floating-point support (FP switch not specified in Task Builder .TSK file specification).

IE.AST	Directive was issued from an AST service routine, or ASTs are disabled.
IE.ADP	Part of the DPB is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

Notes

1. A Specify Floating Point Processor Exception AST requires dynamic memory.

2. The Executive queues Floating Point Processor exception ASTs when a Floating Point Processor exception trap occurs for the task. No subsequent ASTs of this kind can be queued for the task until the first one queued has actually been effected.

3. The Floating Point Processor exception AST service routine is entered with the task stack in the following state:

```
SP+12 - Event flag mask word
SP+10 - PS of task prior to AST
SP+06 - PC of task prior to AST
SP+04 - DSW of task prior to AST
SP+02 - Floating exception code
SP+00 - Floating exception address
```

   The task must remove the floating exception code and address from the task's stack before an AST Service Exit directive is executed.

4. This directive cannot be issued either from an AST service routine or when ASTs are disabled.

5. This directive applies only to the Floating Point Processor.

8.66 SPND$S - SUSPEND

The Suspend directive instructs the system to suspend execution of the issuing task. A task can suspend only itself, not another task. The task can be restarted by a Resume directive.

High-Level Language Call

```
CALL SUSPND   [(ids)]
```

ids Directive status

Macro Call

```
SPND$S   [err]
```

err Error routine address

Macro Expansion

```
SPND$S  ERR
MOV     (PC)+,-(SP)     ;PUSH DPB ONTO THE STACK
.BYTE   45.,1           ;SPND$S MACRO DIC, DPB SIZE=1 WORD
EMT     377             ;TRAP TO THE EXECUTIVE
BCC     .+6             ;BRANCH IF DIRECTIVE SUCCESSFUL
JSR     PC,ERR          ;OTHERWISE, CALL ROUTINE "ERR"
```

Local Symbol Definitions

None

DSW Return Codes

IS.SPD	Successful completion (task was suspended).
IE.ADP	Part of the DPB is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

Notes

1. A suspended task retains control of the system resources allocated to it. The Executive makes no attempt to free these resources until a task exits.

2. A suspended task is eligible for checkpointing unless it is fixed or declared to be noncheckpointable.

3. Because this directive requires only a one-word DPB, the $S form of the macro is recommended. It requires less space and executes with the same speed as that of the DIR$ macro.

8.67 SPWN$ - SPAWN

The Spawn directive requests a specified task for execution, optionally queuing a command line and establishing the task's TI: as a physical terminal.

When this directive is issued, an Offspring Control Block (OCB) is queued to the offspring TCB, and a rundown count is incremented in the parent task's TCB. The rundown count is used to inform the Executive that the task is a parent task and has one or more offspring tasks; cleanup is necessary if a parent task exits with active offspring tasks. The rundown count is decremented when the spawned task exits. The OCB contains the TCB address as well as sufficient information to effect all of the specified exit events when the offspring task exits.

If a command line is specified, it is buffered in the Executive pool and queued for the offspring task for subsequent retrieval by the offspring task with the Get Command Line directive. The maximum command line length is 255 (decimal) characters.

For high-level languages, call SPAWNN instead of SPAWN when you do not use ASTs. Using SPAWN stops the system from bringing an additional impure area into your task root, thus saving virtual address space. The interface routines would normally use the additional impure area to save context during an AST.

If you specify an AST and do not use SPAWNN to invoke the directive, an exit AST routine is effected when the spawned task exits. The iparm parameter contains the address of the task's exit status block. You can use the address to identify which spawned task has either exited or emitted status.

High-Level Language Call

```
CALL SPAWN (rtname,[iugc],[iumc],[iefn],[iast],[iesb]
       ,[iparm],[icmlin],[icmlen],[iunit],[dnam][,ids])

CALL SPAWNN (rtname,[iugc],[iumc],[iefn],[iast],[iesb]
       ,[iparm],[icmlin],[icmlen],[iunit],[dnam][,ids])
```

rtname	Name (RAD50) of the offspring task to be spawned.
iugc	Group code number for the UIC of the offspring task.
iumc	Member code number for the UIC of the offspring task.

iefn — Event flag to be set when the offspring task exits or emits status.

iast — Name of an AST routine to be called when the offspring task exits or emits status. The system ignores this parameter when you call SPAWNN.

NOTE

Refer to Section 7.4.7 for important guidelines on using high-level language AST service routines.

iesb — Name of an eight-word status block to be written when the offspring task exits or emits status.

Word	0	Offspring task exit status
Word	1	System abort code
Word	2-7	Reserved

NOTE

The exit status block defaults to one word. To use the eight-word exit status block, you must specify the logical OR of the symbol SP.WX8 and the event flag number in the iefn parameter above.

iparm — Name of a word to receive the status block address when the AST occurs.

icmlin — Name of a command line to be queued for the offspring task.

icmlen — Length of the command line (255 characters maximum).

iunit — Unit number of terminal to be used as the TI: for the offspring task. If a value of 0 is specified, the TI: of the issuing task is propagated. A task must be privileged in order to specify a TI: other than the parent task's TI:.

dnam — Device name mnemonic.

ids — Integer to receive the Directive Status Word.

Macro Call

```
SPWN$   tname,pn,pr,[ugc],[umc],[efn],[east],[esb]
             ,[cmdlin],[cmdlen],[unit],[dnam]
```

pn — Partition name--ignored, but must be present

pr — Priority--ignored, but must be present

tname — Name (RAD50) of the offspring task to be spawned.

ugc — Group code number for the UIC of the offspring task.

umc — Member code number for the UIC of the offspring task.

efn — The event flag to be cleared on issuance and set when the offspring task exits or emits status.

east — Address of an AST routine to be called when the offspring task exits or emits status.

esb — Address of an eight-word status block to be written when the offspring task exits or emits status.

Word	0	Offspring task exit status
Word	1	System abort code
Word	2-7	Reserved

NOTE

The exit status block defaults to one word. To use the eight-word exit status block, you must specify the logical OR of the symbol SP.WX8 and the event flag number in the efn parameter above.

cmdlin — Address of a command line to be queued for the offspring task.

cmdlen — Length of the command line (maximum length is 255 decimal).

unit — Unit number of terminal to be used as the TI: for the offspring task. If a value of 0 is specified, the TI: of the issuing task is propagated. A task must be privileged in order to specify a TI: other than the parent task's TI:.

dnam — Device name mnemonic. If not specified, the default is TI:.

Macro Expansion

```
SPWN$   ALPHA,,,3,7,1,ASTRUT,STBLK,CMDLIN,72.,0
.BYTE   11.,13.          ;SPWN$ MACRO DIC, DPB SIZE=13 WORDS
.RAD50  ALPHA            ;NAME OF TASK TO BE SPAWNED
.BLKW   3                ;RESERVED
.BYTE   7,3              ;UMC = 7   UGC = 3
.BYTE   1                ;EVENT FLAG NUMBER = 1
.BYTE   16.              ;EXIT STATUS BLOCK CONSTANT
.WORD   ASTRUT           ;AST ROUTINE ADDRESS
.WORD   STBLK            ;EXIT STATUS BLOCK ADDRESS
.WORD   CMDLIN           ;ADDRESS OF COMMAND LINE
.WORD   72.              ;COMMAND LINE LENGTH (CHARACTERS)
.WORD   0                ;TERMINAL UNIT NUMBER =0
```

NOTE

One additional parameter (device name) can be added for a hardware terminal name. For example, TT0: would have the same macro expansion shown above, plus the following:

```
.ASCII          /TT/ ;ASCII  DEVICE NAME
```

The DPB size will then be 14 words.

Local Symbol Definitions

S.PWTN	Task name (4)
S.PWXX	Reserved (6)
S.PWUM	User member code (1)
S.PWUG	User group code (1)
S.PWEF	Event flag number (2)
S.PWEA	Exit AST routine address (2)
S.PWES	Exit status block address (2)
S.PWCA	Command line address (2)
S.PWCL	Command line length (2)
S.PWVT	Terminal unit number (2)
S.PWDN	Device name (2)

DSW Return Codes

IS.SUC	Successful completion.
IE.UPN	Insufficient dynamic memory to allocate an Offspring Control Block, command line buffer, Task Control Block, or Partition Control Block.
IE.INS	The specified task was not installed.
IE.ACT	The specified task was already active.
IE.PRI	Nonprivileged task attempted to specify an offspring task's TI: to be different from its own.
IE.IDU	The specified terminal unit does not exist or the specified TI: device is not a terminal.
IE.IEF	Invalid event flag number (EFN<0 or EFN>64).
IE.ADP	Part of the DPB, exit status block, or command line is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

Notes

1. If the UIC is defaulted, that task is requested to run under the UIC of the parent task. See the notes for the Request Task (RQST$) directive for more information about task UICs.

2. The virtual mapping of the exit status block should not be changed while the connection is in effect. Doing so can cause obscure errors.

3. The types of operations that a high-level language AST routine can perform are extremely limited. See Chapter 7 for a list of restrictions.

8.68 SRDA$ - SPECIFY RECEIVE DATA AST

The Specify Receive Data AST directive instructs the system to record one of the following two cases:

- Receive data ASTs for the issuing task are desired, and the Executive transfers control to a specified address when data has been placed in the task's receive queue.
- Receive data ASTs for the issuing task are no longer desired.

When the directive specifies an AST service routine entry point, receive data ASTs for the task subsequently occur whenever data has been placed in the task's receive queue; the Executive transfers control to the specified address.

When the directive omits an entry point address, the Executive disables receive data ASTs for the issuing task. Receive data ASTs do not occur until the task issues another Specify Receive Data AST directive that specifies an entry point address. (See Notes.)

High-Level Language Call

Neither the FORTRAN language nor the ISA standard permits direct linking to system trapping mechanisms; therefore, this directive is not available to high-level language tasks.

Macro Call

```
SRDA$   [ast]

  ast     AST service routine entry point address
```

Macro Expansion

```
SRDA$   RECAST
.BYTE   107.,2          ;SRDA$ MACRO DIC, DPB SIZE=2 WORDS
.WORD   RECAST          ;ADDRESS OF RECEIVE AST
```

Local Symbol Definitions

```
S.RDAE      AST entry address (2)
```

DSW Return Codes

IS.SUC	Successful completion.
IE.UPN	Insufficient dynamic memory.
IE.ITS	AST entry point address is already unspecified.
IE.AST	Directive was issued from an AST service routine, or ASTs are disabled.
IE.ADP	Part of the DPB is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

Notes

1. A Specify Receive Data AST requires dynamic memory.

2. The Executive queues receive data ASTs when a message is sent to the task. No future receive data ASTs will be queued for the task until the first one queued has been effected.

3. The task enters the receive data AST service routine with the task stack in the following state:

   SP+06 - Event flag mask word
   SP+04 - PS of task prior to AST
   SP+02 - PC of task prior to AST
   SP+00 - DSW of task prior to AST

   No trap-dependent parameters accompany a receive data AST; therefore, the AST Service Exit directive must be executed with the stack in the same state as when the AST was effected.

4. This directive cannot be issued either from an AST service routine or when ASTs are disabled.

8.69 SREF$ - SEND BY REFERENCE

The Send By Reference directive inserts a packet that contains a reference to a region into the receive-by-reference queue of a specified (receiver) task. The Executive automatically attaches the receiver task for each Send By Reference directive issued by the task to the specified region (the region identified in W.NRID of the Window Definition Block).

The attachment occurs even if the receiver task is already attached to the region, unless bit WS.NAT in W.NSTS of the Window Definition Block is set. The successful execution of this directive causes a significant event to occur.

The send packet contains:

- A pointer to the created attachment descriptor, which becomes the region ID to be used by the receiver task.
- The offset and length words specified in W.NOFF and W.NLEN of the Window Definition Block (which the Executive passes without checking).
- The receiver task's permitted access to the region, contained in the window status word W.NSTS.
- The sender task name.
- Optionally, the address of an eight-word buffer that contains additional information. (If the packet does not include a buffer address, the Executive sends eight words of 0.)

The receiver task automatically has access to the entire region as specified in W.NSTS. The sender task must be attached to the region with at least the same types of access. By setting all the bits in W.NSTS to 0, the receiver task can default the permitted access to that of the sender task.

If the directive specifies an event flag, the Executive sets the flag in the sender task (when the receiver task acknowledges the reference) by issuing the Receive By Reference directive. When the sender task exits, the system searches for unreceived references that specify event flags, and prevents invalid attempts to set the flags. The references themselves remain in the receiver task's receive-by-reference queues.

High-Level Language Call

```
CALL SREF (tsk,[efn],iwdb,[isrb][,ids])
```

tsk	A single-precision, floating-point variable that contains the name of the receiving task in Radix-50 format
efn	Event flag number
iwdb	An eight-word integer array that contains a Window Definition Block (see Section 5.5.2.2)
isrb	An eight-word integer array that contains additional information. (If specified, the address of isrb is placed in iwdb(8). If omitted, the contents of iwdb(8) remain unchanged.)
ids	Directive status

Macro Call

```
SREF$   task,wdb[,efn]
```

task	Name of the receiver task
wdb	Window Definition Block address
efn	Event flag number

Macro Expansion

```
SREF$   ALPHA,WDBADR,48.
.BYTE   69.,5           ;SREF$ MACRO DIC, DPB SIZE=5 WORDS
.RAD50  /ALPHA/         ;RECEIVER TASK NAME
.WORD   48.             ;EVENT FLAG NUMBER
.WORD   WDBADR          ;WDB ADDRESS
```

Definition Block Parameters

Table 8-11 describes the Window Definition Block parameters for this directive.

Table 8-11: Window Definition Block Parameters for SREF$

Array Element	Offset	Description
Input Parameters		
iwdb(4	W.NRID	ID of the region to be sent by reference
iwdb(5)	W.NOFF	Offset word, passed without checking
iwdb(6	W.NLEN	Length word, passed without checking
iwdb(7)	W.NSTS	Bit settings* in window status word (the receiver task's permitted access):
		Bit — **Definition**
		WS.RED — 1 if read access permitted
		WS.WRT — 1 if write access permitted
		WS.EXT — 1 if extend access permitted
		WS.DEL — 1 if delete access permitted
iwdb(8)	W.NSRB	Optional address of an eight-word buffer containing additional information
Output Parameters		
None		

Local Symbol Definitions

S.RETN	Receiver task name (4)
S.REBA	Window Definition Block base address (2)
S.REEF	Event flag number (2)

* If you are a high-level language programmer, refer to Section 5.5.2 to determine the bit values represented by the symbolic names described.

DSW Return Codes

IS.SUC	Successful completion.
IE.UPN	A send packet or an attachment descriptor could not be allocated.
IE.INS	The sender task attempted to send a reference to an Ancillary Control Processor (ACP) task, or task not installed.
IE.PRI	Specified access not allowed to sender task itself.
IE.NVR	Invalid region ID.
IE.IEF	Invalid event flag number (EFN<0 or EFN>64).
IE.HWR	Region had load failure or parity error.
IE.ADP	The address check of the DPB, the WDB, or the send buffer failed.
IE.SDP	DIC or DPB size is invalid.

Notes

1. For the user's convenience, the order of the SREF$ macro arguments does not directly correspond to the format of the DPB. The arguments are arranged so that the optional argument (efn) is at the end of the macro call. This arrangement is also compatible with the SDAT$ macro.

2. Because region attachment requires system dynamic memory, the receiver task should detach from any region to which it was already attached, in order to prevent depletion of the memory pool; that is, the task needs to be attached to a given region only once.

8.70 SREX$ - SPECIFY REQUESTED EXIT AST DIRECTIVE

The Specify Requested Exit AST directive allows the task issuing the directive to specify the AST service routine to be entered if a directive attempts to abort the task. This directive allows a task to enter a routine for clean-up instead of abruptly aborting.

If an AST address is not specified, any previously specified exit AST is canceled.

Privileged tasks enter the specified AST routine each time an abort is issued. Nonprivileged tasks enter the specified AST routine only once. Subsequent attempts to abort the task will actually abort the task.

High-Level Language Call

```
CALL SREX (ast,ipblk,ipblkl,[dummy][,ids])
```

ast — Name of the externally declared AST subroutine

NOTE

Refer to Section 7.4.7 for important guidelines on using high-level language AST service routines.

ipblk — Name of an integer array to receive the trap-dependent parameters

ipblkl — Number of parameters to be returned into the ipblk array.

dummy — Reserved for future use

ids — Name of an optional integer to receive the Directive Status Word

Macro Call

```
SREX$   [ast][,dummy]
```

ast — AST service routine entry point address

dummy — Reserved for future expansion

Macro Expansion

```
SREX$   REQAST
.BYTE   167.,3          ;SREX$ MACRO DIC, DPB SIZE=3 WORDS
.WORD   REQAST          ;EXIT AST ROUTINE ADDRESS
.WORD   0               ;RESERVED FOR FUTURE EXPANSION
```

NOTE

The DPB length for the SREX$ form of the directive is three words.

Local Symbol Definitions

S.REAE	Exit AST routine address (2)

DSW Return Codes

IS.SUC	Successful completion.
IE.UPN	Insufficient dynamic storage.
IE.AST	Directive was issued from an AST service routine, or ASTs are disabled.
IE.ITS	ASTs already not desired, or nonprivileged task attempted to respecify or cancel the AST after one had already occurred.
IE.ADP	Part of the DPB is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

Notes

1. The issuing task can use the information returned on the stack for this directive to decide how to handle the abort attempt.

   After specifying a requested exit AST using the SREX$ form of the directive, the issuing task will enter the AST service routine if any attempt is made to abort the task. Nonprivileged abort attempts must originate from the same TI: as that of the issuing task.

   When the AST service routine is entered and the AST has been specified using the SREX$ directive, the task's stack is in the following state:

SP+12 - Event flag mask word
SP+10 - PS of task prior to AST
SP+06 - PC of task prior to AST
SP+04 - DSW of task prior to AST
SP+02 - Trap-dependent parameter
SP+00 - Number of bytes to add to SP to clean stack (4)

The trap-dependent parameter is formatted as follows:

Bit 0 = 0 if the abort attempt was privileged
= 1 if the abort attempt was nonprivileged

Bit 1 = 0 if the ABRT$ directive was issued

Bits 2 through 15 are reserved for future use

The task must remove the trap-dependent parameters from the stack before an AST Service Exit directive is executed. The recommended method is to add the value stored in SP+00 to SP. This is also the only recommended way to access the non-trap-dependent parameters on the stack.

2. The event flag mask word at the bottom of the stack preserves the Wait For conditions of a task prior to AST entry. After an AST, a task can return to a Wait For state. Because these flags and other stack data are in the user task, they can be modified. However, modifying the stack data can cause unpredictable results. Therefore, such modification is not recommended.

3. Please see Chapter 7 for a list of restrictions on operations that can be performed in a high-level language AST routine.

8.71 SRRA$ - SPECIFY RECEIVE-BY-REFERENCE AST

The Specify Receive-By-Reference AST directive instructs the system to record one of the following two cases:

- Receive-by-reference ASTs for the issuing task are desired, and the Executive transfers control to a specified address when such an AST occurs.
- Receive-by-reference ASTs for the issuing task are no longer desired.

When the directive specifies an AST service routine entry point, receive-by-reference ASTs for the task will occur. The Executive will transfer control to the specified address.

When the directive omits an entry point address, the Executive prevents the occurrence of receive-by-reference ASTs for the issuing task. Receive-by-reference ASTs will not occur until the task issues another Specify Receive-By-Reference AST directive that specifies an entry point address. See Notes.

High-Level Language Call

Neither the FORTRAN language nor the ISA standard permits direct linking to system trapping mechanisms; therefore, this directive is not available to high-level language tasks.

Macro Call

```
SRRA$   [ast]

  ast     AST service routine entry point address (0)
```

Macro Expansion

```
SRRA$   RECAST
.BYTE   21.,2           ;SRRA$ MACRO DIC, DPB SIZE=2 WORDS
.WORD   RECAST          ;ADDRESS OF RECEIVE AST
```

Local Symbol Definitions

```
S.RRAE     AST entry address (2)
```

DSW Return Codes

IS.SUC	Successful completion.
IE.UPN	Insufficient dynamic memory.
IE.ITS	AST entry point address is already unspecified.
IE.AST	Directive was issued from an AST service routine, or ASTs are disabled.
IE.ADP	Part of the DPB is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

Notes

1. Specify Receive-By-Reference AST requires dynamic memory.

2. The Executive queues receive-by-reference ASTs when a message is sent to the task. Future receive-by-reference ASTs will not be queued for the task until the first one queued has been effected.

3. The task enters the receive-by-reference AST service routine with the task stack in the following state:

   ```
   SP+06 - Event flag mask word
   SP+04 - PS of task prior to AST
   SP+02 - PC of task prior to AST
   SP+00 - DSW of task prior to AST
   ```

   No trap-dependent parameters accompany a receive-by-reference AST; therefore, the AST Service Exit directive must be executed with the stack in the same state as when the AST was effected.

4. This directive cannot be issued either from an AST service routine or when ASTs are disabled.

8.72 STIM$ - SET SYSTEM TIME

The Set System Time directive instructs the system to set the system's internal time to the specified time parameters. Optionally, the Set System Time directive returns the system's current internal time to the issuing task before setting the system time to the specified values.

All time parameters must be specified as binary numbers.

A task must be privileged to issue this directive.

When this directive changes the system time by a specified amount, it also effectively changes the time of anything resident on the clock queue by the same amount. Thus, the synchronization of events is maintained.

High-Level Language Call

```
CALL SETTIM (ibufn[,ibufp[,ids]])
```

ibufn	An eight-word integer array (new time specification buffer)
ibufp	An eight-word integer array (previous time buffer)
ids	Directive status

Macro Call

```
STIM$ nbuf,[obuf]
```

nbuf	Address of eight-word new time specification buffer
obuf	Address of eight-word buffer to receive the old (previous) system time parameters

Buffer Format

Word 0	Year (since 1900).
Word 1	Month (1-12).
Word 2	Day (1-n, where n is the highest day possible for the given month and year).
Word 3	Hour (0-23).

Word 4 Minute (0-59).

Word 5 Second (0-59).

Word 6 Tick of second (0-n, where n is the frequency of the system clock minus one). If the next parameter (ticks per second) is defaulted, this parameter is ignored.

Word 7 Ticks per second (must be defaulted or must match the frequency of the system clock at 64 decimal ticks per second). This parameter is used to verify the intended precision of the "tick of second" parameter.

NOTE

If any of the specified new time parameters are defaulted (equal to -1), the corresponding previous system time parameters will remain unchanged and will be substituted for the defaulted parameters during argument validation.

Macro Expansion

```
STIM$   NEWTIM,OLDTIM
.BYTE   61.,3           ;STIM$ DIC, DPB SIZE=3 WORDS
.WORD   NEWTIM          ;ADDRESS OF 8.-WORD INPUT BUFFER
.WORD   OLDTIM          ;ADDRESS OF 8.-WORD OUTPUT BUFFER
```

Local Symbol Definitions

S.TIBA Input buffer address (2)

S.TIBO Output buffer address (2)

The following offsets are assigned relative to the start of each time parameters buffer:

S.TIYR Year (2)

S.TIMO Month (2)

S.TIDA Day (2)

S.TIHR Hour (2)

S.TIMI Minute (2)

S.TICS Second (2)

S.TICT Clock tick of second (2)

S.TICP Clock ticks per second (2)

DSW Return Codes

IS.SUC Successful completion.

IE.PRI The issuing task is not privileged.

IE.ITI One of the specified time parameters is out of range, or both the tick-of-second parameter and the ticks-per-second parameter were specified and the ticks-per-second parameter does not match the system's clock frequency. The system time at the moment the directive is issued (returned in the second buffer) can be useful in determining the cause of the fault if any of the specified time parameters were defaulted.

IE.ADP Part of the DPB or one of the buffers is out of the issuing task's address space.

IE.SDP DIC or DPB size is invalid.

Notes

1. The buffers used in this directive are compatible with those of the Get Time Parameters (GTIM$) directive.

2. The second buffer (previous time) is only filled in if the directive was successfully executed or if it was rejected with an error code of IE.ITI.

8.73 STLO$ - STOP FOR LOGICAL OR OF EVENT FLAGS

The Stop For Logical OR Of Event Flags directive instructs the system to stop the issuing task until the Executive sets one or more of the indicated event flags from one of the following groups:

GR 0	Local flags 1-16
GR 1	Local flags 17-32
GR 2	Common flags 33-48
GR 3	Common flags 49-64

The task does not stop itself if any of the indicated flags are already set when the task issues the directive. This directive cannot be issued at AST state.

A task that is stopped for one or more event flags can only be unstopped by setting the specified event flag; it cannot be unstopped with the Unstop directive.

For high-level language calls, you can use the STLORS routine to receive the Directive Status Word.

High-Level Language Call

```
CALL STLOR (ief1,ief2,ief3, ... ief(n))

CALL STLORS (ids,ief1,ief2,ief3, ... ief(n))
```

ids	Integer to receive the Directive Status Word (STLORS only)
ief1 ... ief(n)	List of event flag numbers

Macro Call

```
STLO$   grp, msk
```

grp	Desired group of event flags
msk	A 16-bit mask word

Macro Expansion

```
STLO$   1,47
.BYTE   137.,3          ;STLO$ MACRO DIC, DPB SIZE=3 WORDS
.WORD   1               ;GROUP 1 FLAGS (FLAGS 17-32)
.WORD   47              ;MASK WORD= 47 (FLAGS 17,18,19,22)
```

Local Symbol Definitions

S.TLGR	Group flags (2)
S.TLMS	Mask word (2)

DSW Return Codes

IS.SUC	Successful completion.
IE.AST	The issuing task is at AST state.
IE.IEF	An event flag group other than 0 through 3 was specified, or the event flag mask word is zero.
IE.ADP	Part of the DPB is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

Notes

1. There is a one-to-one correspondence between bits in the mask word and the event flags in the specified group; that is, if group 1 were specified (as in the above macro expansion example), bit 0 in the mask word would correspond to event flag 17, bit 1 to event flag 18, and so forth.

2. The Executive does not arbitrarily clear event flags when Stop For Logical OR Of Event Flags conditions are met. Some directives (Queue I/O Request, for example) implicitly clear a flag; otherwise, they must be explicitly cleared by a Clear Event Flag directive.

3. The argument list specified in the high-level language call must contain only event flag numbers that lie within one event flag group. If event flag numbers are specified that lie in more than one group, or if an invalid event flag number is specified, a fatal high-level language error is generated.

4. Tasks stopped for event flag conditions cannot be unstopped by issuing the Unstop directive; tasks stopped in this manner can only be unstopped by meeting event flag conditions.

5. The grp operand must always be of the form n regardless of the macro form used. In all other macro calls, numeric or address values for $S form macros have the form:

```
#n
```

For STLO$S this form of the grp argument would be:

n

8.74 STOP$S - STOP

The Stop directive stops the issuing task. This directive cannot be issued at AST state. A task stopped in this manner can only be unstopped by another task that issues an Unstop directive directed to the task or the task issuing an Unstop directive at AST state.

High-Level Language Call

```
CALL STOP ([ids])
```

ids	Integer to receive the Directive Status Word

Macro Call

```
STOP$S  [err]
```

err	Error routine address

Macro Expansion

```
STOP$S
MOV     (PC)+,-(SP)     ;PUSH DPB ONTO THE STACK
.BYTE   131.,1          ;STOP$ MACRO DIC, DPB SIZE=1 WORD
EMT     377             ;TRAP TO THE EXECUTIVE
```

Local Symbol Definitions

None

DSW Return Codes

IS.SET	Successful completion.
IE.AST	The issuing task is at AST state.
IE.ADP	Part of the DPB is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

8.75 STSE$ - STOP FOR SINGLE EVENT FLAG

The Stop For Single Event Flag directive instructs the system to stop the issuing task until the specified event flag is set. If the flag is set at issuance, the task is not stopped. This directive cannot be issued at the AST state.

A task that is stopped for one or more event flags can only become unstopped by setting the specified event flag. The Unstop directive cannot be used to unstop the task.

High-Level Language Call

```
CALL STOPFR (iefn[,ids])
```

iefn	Event flag number
ids	Integer to receive Directive Status Word

Macro Call

```
STSE$   efn
```

efn	Event flag number

Macro Expansion

```
STSE$   7
.BYTE   135.,2          ;STSE$ MACRO DIC, DPB SIZE=2 WORDS
.WORD   7               ;LOCAL EVENT FLAG NUMBER = 7
```

Local Symbol Definitions

S.TSEF	Event flag number (2)

DSW Return Codes

IS.SUC	Successful completion.
IE.AST	The issuing task is at AST state.
IE.IEF	Invalid event flag number (EFN<1 or EFN>64).
IE.ADP	Part of the DPB is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

8.76 SVDB$ - SPECIFY SST VECTOR TABLE FOR DEBUGGING AID

The Specify SST Vector Table For Debugging Aid directive instructs the system to record the address of a table of SST service routine entry points for use by an intratask debugging aid (ODT, for example).

To deassign the vector table, omit the parameters **a** and **l** from the macro call.

Whenever an SST service routine entry is specified in both the table used by the task and the table used by a debugging aid, the trap occurs for the debugging aid, not for the task.

High-Level Language Call

Neither the FORTRAN language nor the ISA standard permits direct linking to system trapping mechanisms; therefore, this directive is not available to high-level language tasks.

Macro Call

```
SVDB$   [a][,l]
```

a	Address of SST vector table
l	Length of (number of entries in) the table in words

The vector table has the following format:

Word 0	Odd address of nonexistent memory error
Word 1	Memory protect violation
Word 2	T-bit trap or execution of a BPT instruction
Word 3	Execution of an IOT instruction
Word 4	Execution of a reserved instruction
Word 5	Execution of a non-RSX EMT instruction (see Note)
Word 6	Execution of a TRAP instruction
Word 7	Reserved for future use

A 0 entry in the table indicates that the task will not process the corresponding SST.

SVDB$ - SPECIFY SST VECTOR TABLE FOR DEBUGGING AID

Macro Expansion

```
SVDB$   SSTTBL,4
.BYTE   103.,3          ;SVDB$ MACRO DIC, DPB SIZE=3 WORDS
.WORD   SSTTBL          ;ADDRESS OF SST TABLE
.WORD   4               ;SST TABLE LENGTH=4 WORDS
```

Local Symbol Definitions

S.VDTA	Table address (2)
S.VDTL	Table length (2)

DSW Return Codes

IS.SUC	Successful completion.
IE.ADP	Part of the DPB or table is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

Note

A non-RSX EMT instruction is any EMT instruction not normally used by the system (EMT 1 through 375).

8.77 SVTK$ - SPECIFY SST VECTOR TABLE FOR TASK

The Specify SST Vector Table For Task directive instructs the system to record the address of a table of SST service routine entry points for use by the issuing task.

To deassign the vector table, omit the parameters **ta** and **tl** from the macro call.

Whenever an SST service routine entry is specified in both the table used by the task and the table used by a debugging aid, the trap occurs for the debugging aid, not for the task.

High-Level Language Call

Neither the FORTRAN language nor the ISA standard permits direct linking to system trapping mechanism; therefore, this directive is not available to high-level language tasks.

Macro Call

```
SVTK$   [ta][,tl]
```

ta — Address of SST vector table

tl — Length of (that is, the number of entries in) the table, in words

The vector table has the following format:

Word 0	Odd address of nonexistent memory error
Word 1	Memory protect violation
Word 2	T-bit trap or execution of a BPT instruction
Word 3	Execution of an IOT instruction
Word 4	Execution of a reserved instruction
Word 5	Execution of a non-RSX EMT instruction (See Note)
Word 6	Execution of a TRAP instruction
Word 7	Reserved for future use

A 0 entry in the table indicates that the task does not want to process the corresponding SST.

Macro Expansion

```
SVTK$   SSTTBL,4
.BYTE   105.,3          ;SVTK$ MACRO DIC, DPB SIZE=3 WORDS
.WORD   SSTTBL          ;ADDRESS OF SST TABLE
.WORD   4               ;SET TABLE LENGTH=4 WORDS
```

Local Symbol Definitions

S.VTTA	Table address (2)
S.VTTL	Table length (2)

DSW Return Codes

IS.SUC	Successful completion.
IE.ADP	Part of the DPB or table is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

Note

A non-RSX EMT instruction is any EMT instruction not normally used by the system (EMT 1 through 375).

8.78 SWST$ - SWITCH STATE

The SWST$ directive makes it possible for a privileged task that is not itself mapped to the Executive to map subroutines that require access to the Executive. The subroutines must be written in position-independent code (PIC). Address references must use absolute mode or PC-relative mode. (See the *PDP-11 Macro-11 Reference Manual*.)

The SWST$ directive maps the subroutine through APR5 (that is, it uses virtual addresses 120000 through 137777 octal). Therefore, the subroutine must fall within the limits of 4K words of the base virtual address specified in the directive. The subroutine itself is executed as part of the SWST$ directive and is, therefore, in system state during its execution. Local data references must also be within the 4K word limit.

High-Level Language Call

Not supported

Macro Call

```
SWST$   base,addr
```

base — The base virtual address within the task for mapping the subroutine through APR5.

addr — Virtual address of the subroutine to be executed in system state by the directive.

Macro Expansion

```
SWST$   BASE,ADDR
.BYTE   175.,3          ;SWST$ MACRO DIC, DPB SIZE = 3 WORDS
.WORD   BASE            ;BASE VIRTUAL ADDRESS FOR MAPPING
                        ;THE SUBROUTINE THROUGH APR5
.WORD   ADDR            ;VIRTUAL ADDRESS OF THE SUBROUTINE
                        ;EXECUTED AT SYSTEM STATE
```

Local Symbol Definitions

S.WBAS — Base virtual address for mapping the subroutine through APR5

S.WADD — Virtual address of the subroutine executed at system state

DSW Return Codes

IS.SUC	Successful completion.
IE.PRI	The issuing task is not privileged.
IE.MAP	The specified system state routine is greater than 4K words from the specified base.
IE.ADP	Part of the DPB is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

Notes

1. User mode register contents are preserved across execution of the kernel mode subroutine. Contents of the user mode registers are passed into the kernel mode registers. Contents of the kernel mode registers are discarded when the subroutine has completed execution.

2. User mode registers appear at the following octal stack offsets when executing the specified subroutine in kernel mode:

```
User mode R0 at S.WSR0 Offset on kernel stack
User mode R1 at S.WSR1 Offset on kernel stack
User mode R2 at S.WSR2 Offset on kernel stack
User mode R3 at S.WSR3 Offset on kernel stack
User mode R4 at S.WSR4 Offset on kernel stack
User mode R5 at S.WSR5 Offset on kernel stack
```

   If you want to return any register values to the user mode registers, you must store the desired values on the stack using the above offsets.

3. Virtual address values passed to system state in a register must be realigned through kernel APR5. For example, if R5 contains address n, and the base virtual address in the DPB is 1000 (octal), the value in R5 must be aligned using the formula:

```
n+120000+base virtual address
```

   Therefore, the resultant value is n+121000.

4. The system state subroutine should exit by issuing an RTS PC instruction. This causes a successful directive status to be returned when the directive is terminated.

NOTE

Keep in mind that the memory management unit rounds the base address to the nearest 32-word boundary.

8.79 TFEA$ - TEST TASK FEATURE

The Test Task Feature directive tests for a specified task characteristic, such as use of fast remapping feature,

High-Level Language Call

```
CALL TFEA (isym,idsw)
```

isym	Symbol for the specified task feature. See Table 8-12 for a list of task features.
idsw	Integer to receive the Directive Status Word.

Macro Call

```
TFEA$  feat
```

feat	Symbol for the specified task feature. See Table 8-12 for a list of task features.

Macro Expansion

```
TFEA$   T4$FMP
.BYTE   209.,2      ;TFEA$ MACRO DIC, DPB SIZE=2 WORDS
.WORD   T4$FMP      ;FEATURE BEING TESTED (FAST REMAP HERE)
```

Local Symbol Definition

F.TEAF	Feature word (2)

DSW Return Codes

IS.CLR	Successful completion; feature not present
IS.SET	Successful completion; feature present
IE.ADP	Part of the DPB is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

Table 8-12: Task Feature Symbols

Symbol	Meaning
T2$WFR	Task in Wait For state (1=yes)
T2$WFA	Saved T2$WFR on AST in progress
T2$SPN	Task suspended (1=yes)
T2$SPA	Saved T2$SPN on AST in progress
T2$STP	Task stopped (1=yes)
T2$STA	Saved T2$SPN on AST in progress
T2$ABO	Task marked for abort (1=yes)
T2$AFF	Task is installed with affinity
T2$SIO	Task stopped for buffered i/o
T2$SEF	Task stopped for event flag(s) (1=yes)
T2$REX	Requested exit AST specified
T2$CHK	Task not checkpointable (1=yes)
T2$DST	AST recognition disabled (1=yes)
T2$AST	AST in progress (1=yes)
T3$GFL	Reserved
T3$SWS	Reserved
T3$CMD	Task is executing a CLI command
T3$MPC	Mapping change with outstanding I/O
T3$NET	Network protocol level
T3$ROV	Task has resident overlays
T3$CAL	Task has checkpoint space in image
T3$NSD	Task does not allow send data

Symbol	Meaning
T3$RST	Task is restricted (1=yes)
T3$CLI	Task is a command line interpreter
T3$SLV	Task is a slave task (1=yes)
T3$MCR	Task requested as external MCR function
T3$PRV.	Task is privileged (1=yes)
T3$REM	Remove task on exit (1=yes)
T3$PMD	Dump task on synchronous abort (0=yes)
T3$ACP	Task is Ancillary Control Processor (1=yes)
T4$SNC	Task uses commons for synchronization
T4$DSP	Task was built for user I/D space
T4$PRV	Task was privileged, but has cleared T3.PRV with WIMP$S (may reset with WIMP$S if T4$PRV set)
T4$PRO	TCB is (or should be) a prototype
T4$LDD	Task's load device has been dismounted
T4$MUT	Task is a multiuser task
T4$CTC	Task has been processed by CTRL/C abort
T4$FMP	Task has fast remap header extension

8.80 TLOG$ - TRANSLATE LOGICAL NAME

The Translate Logical Name directive translates a logical name to its equivalence value, returning the equivalence value string to a specified buffer.

High-Level Language Call

```
CALL TRALOG (mod,[itbmsk],[dummy],lns,lnssz,iens,ienssz,
             [irsize],[irtbmo],[idsw])
```

mod	The modifier of the logical name within a table. See Section 2.2 for details.
itbmsk	Table mask indicating which tables should not be searched during the translation. See Notes.
dummy	Reserved.
lns	Character array that contains the logical name string.
lnssz	Size (in bytes) of the logical name string.
iens	Character array that contains the equivalence name string.
ienssz	Size (in bytes) of the equivalence name string.
irsize	Address of the word to which the size of the resulting equivalence name string is returned .
irtbmo	Address of the word to which the table number (low byte) and mode (high byte) of the resulting equivalence string is returned.
idsw	Integer to receive the Directive Status Word.

Macro Call

```
TLOG$  mod,tbmsk,0,lns,lnssz,ens,enssz,rsize,rtbmod
```

mod	The modifier of the logical name within a table. See Section 2.2 for details.
tbmsk	Table mask indicating which tables should not be searched during the translation. See Notes.
lns	Character array containing the logical name string.

lnssz	Size (in bytes) of the logical name string.
iens	Character array containing the equivalence name string.
ienssz	Size (in bytes) of the equivalence name string.
rsize	Address of the word to which the size of the resulting equivalence name string is returned.
rtbmod	Address of the word to which the table number (low byte) and mode (high byte) of the resulting equivalence string is returned.

Macro Expansion

```
TLOG$   MOD,TBMASK,0,LNS,LNSSZ,ENS,ENSSZ,RSIZE,RTBMOD
.BYTE   207.,9          ;TLOG$ MACRO DIC, DPB SIZE = 9 WORDS
.BYTE   1               ;SUBFUNCTION CODE FOR TRANSLATION
.BYTE   MOD             ;LOGICAL NAME MODIFIER
.BYTE   TBMASK          ;TABLE MASK
.BYTE   0               ;RESERVED FOR FUTURE USE
.WORD   LNS             ;ADDRESS OF LOGICAL NAME BUFFER
.WORD   LNSSZ           ;BYTE COUNT OF LOGICAL NAME STRING
.WORD   ENS             ;ADDRESS OF EQUIVALENCE NAME BUFFER
.WORD   ENSSZ           ;BYTE COUNT OF EQUIVALENCE NAME
                        ;STRING
.WORD   RSIZE           ;ADDRESS OF BUFFER INTO WHICH
                        ;EQUIVALENCE NAME STRING IS TO BE
                        ;RETURNED
.WORD   RTBMOD          ;ADDRESS OF BUFFER INTO WHICH TABLE
                        ;NUMBER AND MODIFIER ARE TO BE
                        ;RETURNED
```

Local Symbol Definitions

T.LFUN	Subfunction (1)
T.LMOD	Logical name modifier (1)
T.LTBL	Logical table number (1)
T.LLNS	Address of logical name string (2)
T.LLSZ	Byte count of logical name string (2)
T.LENS	Address of equivalence name string (2)
T.LESZ	Byte count of equivalence name string (2)

T.LRSZ Buffer address for returned equivalence string (2)

T.LRTM Buffer address for returned table number and modifier (2)

DSW Return Codes

IS.SUC Successful completion.

IE.RBS The resulting equivalence name string is too large for the buffer to receive it.

IE.LNF The specified logical name string was not found.

IE.IBS The length of the logical or equivalence string is invalid. Each string length must be greater than 0 but not greater than 255 (decimal) characters.

IE.ADP Part of the DPB or user buffer is out of the issuing task's address space, or the user does not have proper access to that region.

IE.SDP DIC or DPB size is invalid.

Note

The table mask is a bit field that indicates which tables the system will *not* search when translating a logical. To search all tables, do not set any of the bits. Set the bits by specifying the following values to inhibit particular tables:

Table	Mask Value (Decimal)
LT.SYS	1
LT.SES	16
LT.USR	4

8.81 UMAP$ - UNMAP ADDRESS WINDOW

The Unmap Address Window directive unmaps a specified window. After the window has been unmapped, references to the corresponding virtual addresses are invalid and cause a processor trap to occur.

High-Level Language Call

```
CALL UNMAP (iwdb[,ids])
```

iwdb	An eight-word integer array containing a Window Definition Block (see Section 5.5.2.2)
ids	Directive status

Macro Call

```
UMAP$   wdb
```

wdb	Window Definition Block address

Macro Expansion

```
UMAP$   WDBADR
.BYTE   123.,2          ;UMAP$ MACRO DIC, DPB SIZE=2 WORDS
.WORD   WDBADR          ;WDB ADDRESS
```

Definition Block Parameters

Table 8-13 describes the Window Definition Block parameters for this directive.

Local Symbol Definitions

U.MABA	Window Definition Block address (2)

Table 8-13: Window Definition Block Parameters for UMAP$

Array Element	Offset	Description
Input parameters		
iwdb(1) bits 0-7	W.NID	ID of the window to be unmapped
Output parameters		
iwdb(7)	W.NSTS	Bit settings* in the window status word:

Bit	Definition
WS.UNM	1 if the window was successfully unmapped

DSW Return Codes

IS.SUC	Successful completion.
IE.ITS	The specified address window is not mapped
IE.NVW	Invalid address window ID.
IE.ADP	DPB or WDB out of range.
IE.SDP	DIC or DPB size is invalid.

Notes

1. See Chapter 5 for complete information on using the memory management features of the Professional.

2. A fast remapping feature is available for frequently mapped regions. See Section 5.7 for details.

---

* If you are a high-level language programmer, see Section 5.5.2 to determine the bit values represented by the symbolic names described.

8.82 USTP$ - UNSTOP TASK

The Unstop Task directive unstops the specified task that has stopped itself by either the Stop, or the Receive Data Or Stop directive. It does not unstop tasks stopped for an event flag or for tasks stopped for buffered I/O. If the Unstop directive is issued to a task previously stopped by means of the Stop, or Receive Or Stop directive, while at task state, and the task is presently at AST state, the task only becomes unstopped when it returns to task state.

It is the responsibility of the unstopped task to determine whether or not it has been validly unstopped.

The Unstop directive does not cause a significant event.

High-Level Language Call

```
CALL USTP (rtname[,ids])
```

rtname	Name of task to be unstopped
ids	Integer to receive directive status information

Macro Call

```
USTP$   tname
```

tname	Name of task to be unstopped

Macro Expansion

```
USTP$   ALPHA
.BYTE   133.,3          ;USTP$ MACRO DIC, DPB SIZE=3 WORDS
.RAD50  /ALPHA/         ;NAME OF TASK TO BE UNSTOPPED
```

Local Symbol Definitions

U.STTN	Task name (4)

DSW Return Codes

IS.SUC	Successful completion.
IE.INS	The specified task is not installed in the system
IE.ACT	The specified task is not active.
IE.ITS	The specified task is not stopped, or it is stopped for event flag or buffered I/O.

IE.ADP	Part of the DPB is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

8.83 VRCD$ - VARIABLE RECEIVE DATA

The Variable Receive Data directive instructs the system to dequeue a variable-length data block for the issuing task; the data block has been queued (FIFO) for the task by a Variable Send Data directive. When a sender task is specified, only data sent by the specified task is received.

The buffer size can be 256 (decimal) words maximum. If no buffer size is specified, the buffer size is 13 (decimal) words. If a buffer size greater than 256 (decimal) is specified, an IE.IBS error is returned.

A two-word sender task name (in Radix-50 form) and the data block are returned in the specified buffer, with the task name in the first two words. For this reason, the storage you allocate within the buffer should be two words greater than the size of the data portion of the message specified in the directive.

Variable-length data blocks are transferred from the sending task to the receiving task by means of buffers in the secondary pool.

If the directive was successful, it returns the number of words transferred into the user buffer. If the directive encounters an error during execution, it returns the error code in the ids parameter.

Any error return of the form IE.XXX is a negative word value. If the status is positive, the value of the status word is the number of words transferred including the task name. For example, if you specify a buffer size of 13 in the VRCD$ call, the value returned in the Directive Status Word is 15 (13 words of data plus the two words needed to return the task name).

High-Level Language Call

```
CALL VRCD ([task],bufadr,[buflen][,ids])
```

task	Sender task name
buf	Address of buffer to receive the sender task name and data
buflen	Length of buffer
ids	Integer to receive the Directive Status Word.

Macro Call

```
VRCD$     [tn],ba[,bl],[ti]
```

tn	Sender task name
ba	Buffer address
bl	Buffer size in words
ti	Reserved for future use. When using the $S or $C forms, specify a null argument for ti.

Macro Expansion

```
VRCD$     TN,BA,,TI
.BYTE     75.,6.          ;VRCD$ MACRO DIC, DPB SIZE=6 WORDS
.RAD50    /TN/            ;SENDER TASK NAME
.WORD     BA              ;ADDRESS OF DATA BUFFER
.WORD     13.             ;LENGTH OF DATA BUFFER (DEFAULT)
.WORD     TI              ;RESERVED
```

Local Symbol Definitions

R.VDTN	Sender task name (4)
R.VDBA	Buffer address (2)
R.VDBL	Buffer length (2)

DSW Return Codes

IS.SUC	Successful completion.
IE.INS	Specified task not installed.
IE.ITS	No data in task's receive queue.
IE.RBS	Receive buffer is too small.
IE.IBS	Invalid buffer size specified (greater than 256 decimal)
IE.ADP	Part of the DPB or buffer is out of the issuing task's address space
IE.SDP	DIC or DPB size is invalid.

8.84 VRCS$ - VARIABLE RECEIVE DATA OR STOP

The Variable Receive Data Or Stop directive instructs the system to dequeue a variable-length data block for the issuing task; the data block has been queued (FIFO) for the task by a Variable Send Data directive.

If there is no such packet to be dequeued, the issuing task is stopped. In this case, another task (the sender task) is expected to issue an Unstop directive after sending the data. When stopped in this manner, the directive status returned is IS.SET, indicating that the task was stopped and that no data has been received; however, since the task must be unstopped in order to see this status, the task can now reissue the Variable Receive Data Or Stop directive to actually receive the data packet.

When a sender task is specified, only data sent by the specified task is received.

Buffer size can be 256 (decimal) words maximum. If no buffer size is specified, the default is 13 (decimal) words. If a buffer size greater than 256 (decimal) is specified, an IE.IBS error is returned.

A two-word sender task name (in Radix-50 form) and the data block are returned in the specified buffer, with the task name in the first two words. For this reason, the storage you allocate within the buffer should be two words greater than the size of the data portion of the message specified in the directive.

Variable-length data blocks are transferred from the sending task to the receiving task by means of buffers in the secondary pool.

High-Level Language Call

```
CALL VRCS ([task],bufadr,[buflen][,ids])
```

task	Sender task name
buf	Address of buffer to receive the sender task name and data
buflen	Length of buffer
ids	Integer to receive the directive status word

If the directive was successful, it returns the number of words transferred into the user buffer. If the directive execution encountered an error, it returns the error code in the ids parameter.

Any error return of the form IE.XXX is a negative word value. If the status is positive, the value of the status word is the number of words transferred including the taskname. For example, if you specify a buffer size of 13 in the VRCS$ call, the value returned in the directive status word is 15 (13 words of data plus the two words needed to return the taskname).

Macro Call

```
VRCS$     [tn],ba[,bl][,ti]
```

tn	Sender task name
ba	Buffer address
bl	Buffer size in words
ti	Reserved for future use. When using the $S or $C forms, you must specify a null argument for ti.

Macro Expansion

```
VRCS$     TN,BA,,TI
.BYTE     139.,6.          ;VRCS$ MACRO DIC, DPB SIZE=6 WORDS
.RAD50    /TN/             ;SENDER TASK NAME
.WORD     BA               ;ADDRESS OF DATA BUFFER
.WORD     13.              ;LENGTH OF DATA BUFFER (DEFAULT)
.WORD     TI               ;RESERVED
```

Local Symbol Definitions

R.VSTN	Sender task name (4)
R.VSBA	Buffer address (2)
R.VSBL	Buffer length (2)
R.VSTI	Reserved (2)

DSW Return Codes

IS.SUC	Successful completion.
IS.SET	Task was stopped and no data was received.
IE.INS	Specified task not installed.
IE.RBS	Receive buffer is too small.

IE.IBS	Invalid buffer size specified (greater than 256 decimal).
IE.ADP	Part of the DPB or buffer is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

8.85 VRCX$ - VARIABLE RECEIVE DATA OR EXIT

The Variable Receive Data Or Exit directive instructs the system to dequeue a variable-length data block for the issuing task; the data block has been queued (FIFO) for the task by a Variable Send Data directive. When a sender task is specified, only data sent by the specified task is received.

A two-word sender task name (in Radix-50 form) and the data block are returned in the specified buffer, with the task name in the first two words. For this reason, the storage you allocate within the buffer should be two words greater than the size of the data portion of the message specified in the directive.

If no data has been sent, a task exit occurs. To prevent the possible loss of send data packets, the user should not rely on I/O rundown to take care of any outstanding I/O or open files; the task should assume this responsibility.

Buffer size can be 256 (decimal) words maximum. If no buffer size is specified, the buffer size is 13 (decimal) words. If you specify a buffer size greater than 256, error IE.IBS is returned.

Variable-length data blocks are transferred from the sending task to the receiving task by means of buffers in the secondary pool.

High-Level Language Call

```
CALL VRCX ([task],bufadr,[buflen][,ids])
```

task	Sender task name
buf	Address of buffer to receive the sender task name and data
buflen	Length of buffer
ids	Integer to receive the directive status word

If the directive is successful, it returns the number of words transferred into the user buffer. If the directive execution encounters an error, it returns the error code in the ids parameter.

Any error return of the form IE.XXX is a negative word value. If the status is positive, the value of the status word is the number of words transferred, including the taskname. For example, if you specify a buffer size of 13 in the VRCX$ call, the value returned in the directive status word is 15 (13 words of data plus the two words needed to return the taskname).

VRCX$ - VARIABLE RECEIVE DATA OR EXIT

Macro Call

```
VRCX$     [tn],ba[,bl][,ti]
```

tn	Sender task name
ba	Buffer address
bl	Buffer size in words
ti	Reserved for future use. When using the $S or $C forms, you must specify a null argument for ti.

Macro Expansion

```
VRCX$     TN,BA,,TI
.BYTE     77.,6.          ;VRCX$ MACRO DIC, DPB SIZE=6 WORDS
.RAD50    /TN/            ;SENDER TASK NAME
.WORD     BA              ;ADDRESS OF DATA BUFFER
.WORD     13.             ;LENGTH OF DATA BUFFER (DEFAULT)
.WORD     TI              ;RESERVED
```

Local Symbol Definitions

R.VXTN	Sender task name (4)
R.VXBA	Buffer address (2)
R.VXBL	Buffer length (2)
R.VXTI	Reserved (2)

DSW Return Codes

IS.SUC	Successful completion.
IE.INS	Specified task not installed.
IE.RBS	Receive buffer is too small.
IE.IBS	Invalid buffer size specified (greater than 256 decimal).
IE.ADP	Part of the DPB or buffer is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

8.86 VSDA$ - VARIABLE SEND DATA

The Variable Send Data directive instructs the system to queue a variable-length data block for the specified task to receive.

Buffer size can be 256 (decimal) words maximum. If no buffer size is specified, the buffer size is 13 (decimal) words. If a buffer size greater than 256 (decimal) is specified, an IE.IBS error is returned.

When an event flag is specified, a significant event is declared if the directive is successfully executed, and the indicated event flag is set for the sending task.

Variable-length data blocks are transferred from the sending task to the receiving task by buffers in the secondary pool.

High-Level Language Call

```
CALL VSDA (task,bufadr,[buflen],[efn][,ids])
```

task	Receiver task name
buf	Address of buffer to receive the receiver task name and data
buflen	Length of buffer
efn	Event flag number
ids	Integer to receive the directive status word

Macro Call

```
VSDA$     tn,ba[,bl][,efn][,spri][,ti]
```

tn	Receiver task name
ba	Buffer address
bl	Buffer size in words
spri	Reserved for future use. When using the $S or $C forms, you must specify a null argument for spri.
ti	Reserved for future use. When using the $S or $C forms, you must specify a null argument for ti.

Macro Expansion

```
VSDA$     TN,BA,,4,SPRI,TI
.BYTE     71.,8.          ;VSDA$ MACRO DIC, DPB SIZE=8 WORDS
.RAD50    /TN/            ;RECEIVER TASK NAME
.WORD     BA              ;ADDRESS OF DATA BUFFER
.WORD     4               ;EVENT FLAG 4
.WORD     13.             ;LENGTH OF DATA BUFFER (DEFAULT)
.WORD     SPRI            ;RESERVED
.WORD     TI              ;RESERVED
```

Local Symbol Definitions

S.DATN	Sender task name (4)
S.DABA	Buffer address (2)
S.DAEF	Event flag number (2)
S.DABL	Buffer length (2)
S.DATI	Reserved (2)

DSW Return Codes

IS.SUC	Successful completion.
IE.UPN	Insufficient dynamic storage.
IE.INS	Specified task not installed.
IE.IBS	Invalid buffer size specified (greater than 256 decimal).
IE.IEF	Invalid event flag number (EFN<0 or EFN>64).
IE.ADP	Part of the DPB or buffer is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

8.87 VSRC$ - VARIABLE SEND, REQUEST AND CONNECT

The Variable Send, Request and Connect directive performs a Variable Send Data to the specified task, requests the task if it is not already active, and then connects to the task. The receiver task normally returns status by the Emit Status or the Exit With Status directive.

Buffer size can be 256 (decimal) words maximum. If no buffer size is specified, the default is 13 (decimal) words. If a buffer size greater than 256 (decimal) is specified, an IE.IBS error is returned.

For high-level languages, call VSRCN instead of VSRC when you do not use ASTs. Using VSRCN stops the system from bringing an additional impure area into your task root, thus saving virtual address space. The interface routines would normally use the additional impure area to save context during an AST.

High-Level Language Call

```
CALL VSRC (rtname,ibuf,[ibuflen],[iefn],[iast],[iesb],
     [iparm][,ids])

CALL VSRCN (rtname,ibuf,[ibuflen],[iefn],[iast],[iesb],
     [iparm][,ids])
```

rtname	Target task name of the offspring task to be connected
ibuf	Name of send buffer
ibuflen	Length of the buffer
iefn	Event flag to be set when the offspring task exits or emits status
iast	Name of an AST routine to be called when the offspring task exits or emits status. The system ignores this parameter when you call VSRCN.
iesb	Name of an eight-word status block to be written when the offspring task exits or emits status

Word	0	Offspring task exit status
Word	1	System abort code
Words	2-7	Reserved

NOTE

The exit status block defaults to one word. To use the eight-word exit status block, you must specify the logical OR of the symbol SP.WX8 and the event flag number in the iefn parameter above.

iparm — Name of a word to receive the status block address when an AST occurs

ids — Integer to receive the Directive Status Word

Macro Call

```
VSRC$   tname,buf[,buflen],[efn],[east],esb]
```

tname — Target task name of the offspring task to be connected

buf — Address of send buffer

buflen — Length of buffer

efn — The event flag to be cleared on issuance and set when the offspring task exits or emits status

east — Address of an AST routine to be called when the offspring task exits or emits status

esb — Address of an eight-word status block to be written when the offspring task exits or emits status

Word	0	Offspring task exit status

Word 1 System abort code

Word 2-7 Reserved

NOTE

The exit status block defaults to one word. To use the eight-word exit status block, you must specify the logical OR of the symbol SP.WX8 and the event flag number in the efn parameter above.

Macro Expansion

```
VSRC$     ALPHA,BUFFR,BUFSIZE,2,SDRCTR,STBLK
.BYTE     141.,8          ;VSRC$ MACRO DIC, DPB SIZE=8 WORDS
.RAD50    /ALPHA/         ;TARGET TASK NAME
.WORD     BUFFR           ;SEND BUFFER ADDRESS
.BYTE     2               ;EVENT FLAG NUMBER = 2
.BYTE     16.             ;EXIT STATUS BLOCK CONSTANT
.WORD     BUFSIZE         ;LENGTH OF BUFFER IN WORDS
.WORD     SDRCTR          ;ADDRESS OF AST ROUTINE
.WORD     STBLK           ;ADDRESS OF STATUS BLOCK
```

Local Symbol Definitions

V.SRTN Task name (4)

V.SRBF Buffer address (2)

V.SREF Event flag (2)

V.SRBL Length of buffer (2)

V.SREA AST routine address (2)

V.SRES Status block address (2)

DSW Return Codes

IS.SUC Successful completion.

IE.UPN Insufficient dynamic memory to allocate a send packet, Offspring Control Block, Task Control Block, or Partition Control Block.

IE.INS The specified task is an ACP or has the no-send attribute.

IE.IEF An invalid event flag number was specified (EFN<0 or EFN>64).

IE.ADP Part of the DPB or exit status block is not in the issuing task's address space.

IE.SDP DIC or DPB size is invalid.

Note

Changing the virtual mapping of the exit status block while the connection is in effect can result in errors that are difficult to diagnose.

8.88 WIMP$ - WHAT'S IN MY PROFESSIONAL

The What's In My Professional directive is a general purpose system information retrieval mechanism. The directive allows a nonprivileged task to retrieve specific information stored by the system without requiring the task to be mapped to the Executive. In all forms, the WIMP$ directive requires three parameters: a subfunction code, a return buffer, and the return buffer size.

The subfunction code specifies the type of information to be returned. The return buffer is space allocated within your task and must be large enough to contain the information returned. Refer to the descriptions of the implemented subfunctions for the specific size of the return buffer, as well as any additional parameters required.

High-Level Language Call

```
CALL    WIMP (sfcn,p1,p2,p3,p4,p5,p6,ids)
```

Macro Call

```
WIMP$   sfcn,p1,p2,p3,p4,p5,p6
```

sfcn	subfunction code
p1	parameter 1
p2	parameter 2
p3	parameter 3
p4	parameter 4
p5	parameter 5
p6	parameter 6
ids	Directive status

NOTE

All parameters are required in high-level language calls. You can use null parameters as placeholders if necessary.

Macro Expansion

```
WIMP$     SFCN,P1,P2,P3,P4,P5,P6

.BYTE 169.,variable
.WORD SFCN          ;SUBFUNCTION CODE
.WORD P1            ;PARAMETER 1
.WORD P2            ;PARAMETER 2
.WORD Pn            ;PARAMETER n
```

Local Symbol Definitions

G.INSF	Subfunction code (2)
G.IP01	Parameter 1 (2)
G.IP02	Parameter 2 (2)
G.IP03	Parameter 3 (2)
G.IP04	Parameter 4 (2)
G.IP05	Parameter 5 (2)
G.IP06	Parameter 6 (2)

DSW Return Codes

IS.SUC	Successful completion.
IE.IDU	Invalid hardware for requested operation.
IE.SDP	DIC, DPB size, or subfunction is invalid .
IE.ADP	Return buffer fails address checks.
IE.RBS	Return buffer too small to receive full information (GI.MSD only).

Implemented Subfunctions

GI.CFG -- Get Configuration Table

The value of the GI.CFG symbol is 12 (decimal).

This subfunction allows you to obtain information about the current hardware configuration of a Professional. This information is stored in a configuration table that is present in the Professional's base system ROM.

The format is:

WIMP$ GI.CFG,buf,siz

buf Return buffer address.

siz Size in words of return buffer. (The size is variable. See Table 8-15 at the end of this section.)

The configuration table consists of several sections: header, device section, boot section, and additional information section. (Note that the additional information section applies only to the PC380.) Table 8-15 at the end of the WIMP$ description shows each section and lists the byte offsets for each item in the return buffer.

For a description of each item in the configuration table return buffer, see Appendix C.

Changes you make to the information in the return buffer are not reflected in the system configuration table. You cannot change any values in the configuration table.

NOTE

After a successful call to Get Configuration Table, the Directive Status Word contains the size in words of the returned configuration table, rather than a DSW success code.

GI.FMK -- Get System Feature Masks and Version Numbers

The value of the GI.FMK symbol is 3 (decimal).

This subfunction allows you to obtain the P/OS base level number and system version number stored by the Executive.

NOTE

Use the FEAT$ directive to obtain feature masks. Never obtain the system feature masks using this WIMP$ subfunction.

The format is:

WIMP$ GI.FMK,buf,siz

buf Return buffer address.

siz Size in words of return buffer. (The size is variable. See Table 8-14)

Table 8-14 shows the return buffer for the GI.FMK subfunction.

Table 8-14: Return Buffer for Get System Version Numbers

Buffer Word	Contents
0	Reserved
1	Reserved
2	Reserved
3	Reserved
4	Reserved
5	P/OS current baselevel number in ASCII, first word
6	P/OS current baselevel number in ASCII, second word
7	P/OS current version number in ASCII, first word
8	P/OS current version number in ASCII, second word

GI.MSD -- Get Mass Storage Device Information

The value of the GI.MSD symbol is 16 (decimal).

This subfunction returns either only those devices that are accessible to the issuing task, or it returns all mass storage devices, whether or not they are accessible to the issuer.

The format is:

WIMP$ GI.MSD,buf,siz,flag

buf Return buffer address

siz Size in words of return buffer (greater than 1)

flag Indicates which set of devices to report:
0 accessible mass storage devices only
1 all mass storage devices

The return buffer must be at least 1 word in length in order to receive a count of the devices found, or else the system returns a DSW code of IE.SDP. The buffer must begin and end on a word boundary.

The first word of the return buffer contains the number of devices (low byte) and the size in words of the device entries (high byte). Following the first word are the device entries for the devices found.

Each returned device entry consists of 4 words in the return buffer. Figure 8-1 shows the format of a device entry. Note that the word values shown in the figure are offsets from the beginning of the device entry.

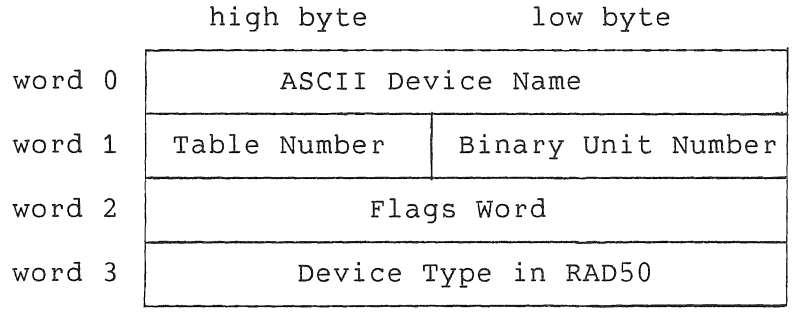


Figure 8-1: Device Entry in GI.MSD Return Buffer

Word 2 of a device entry, the flags word, currently defines four flags. Figure 8-2 shows the format of the flags word.

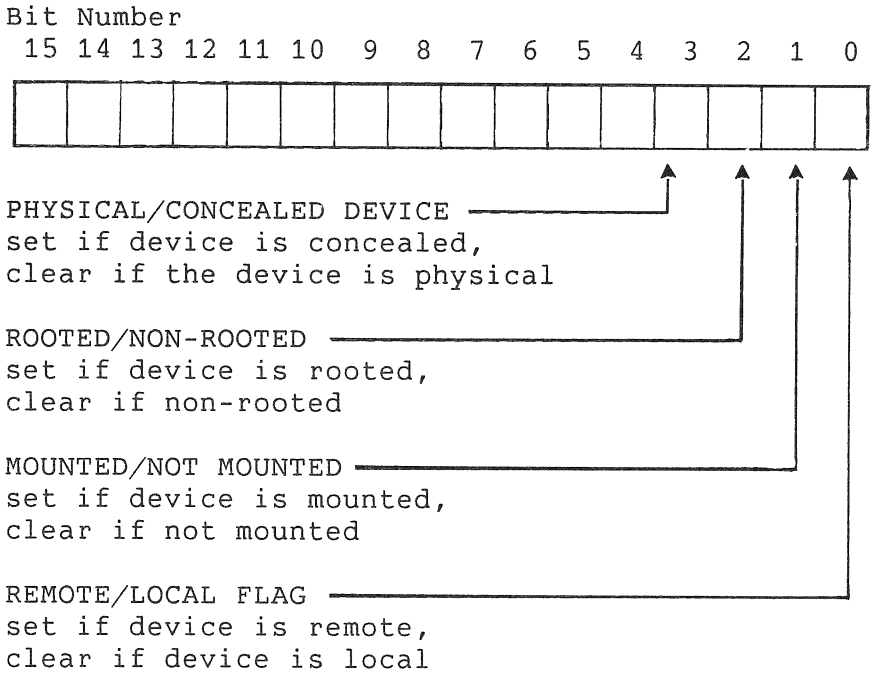


Figure 8-2: Flags Word in GI.MSD Return Buffer

Word 3 of a device entry, the device type, is the generic device code (RX or RD, for example). The system returns the device type only if the device is on-line.

The system returns as much device information as possible in the return buffer. If the system cannot return all the device information requested, it returns a DSW error of IE.RBS. In this case, word 0 of the return buffer contains the total count of devices. Note that the system returns no partial entries.

GI.PRO -- Get or Set Default File Protection

The value of GI.PRO is 18 (decimal).

This subfunction allows you to either get or set the current default file protection.

The format is:

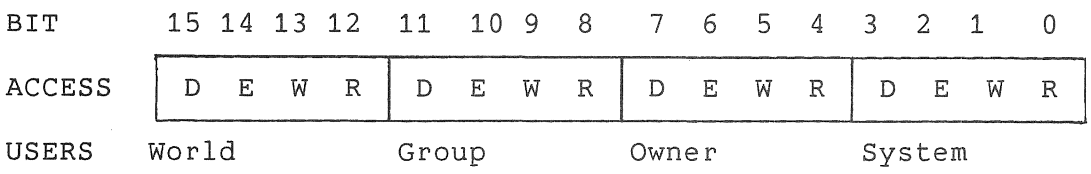
```
WIMP$     GI.PRO,prot,flag
```

prot For Get Protection, address of a word to receive the protection mask For Set Protection, address of a word containing the protection mask

flag Flag that indicates set or get the protection:

0 = Get
1 = Set

The protection mask format for the prot parameter is:



R = READ
W = WRITE
E = EXTEND
D = DELETE

For each type of user, set bits (1) indicate the type of access allowed; clear bits (0) indicate the type of access denied. If all bits are clear for all types of users, there is no user default file protection.

When a file is created, the user can specify an explicit file protection. If the user does not specify this value, the FILES-11 ACP applies the user default file protection, which you can set using the GI.PRO subfunction. If the user default file protection has a value of 0, the ACP uses the default file protection specified when the volume was initialized.

GI.SSN -- Get System Serial Number

The value of the GI.SSN symbol is 13 (decimal).

This subfunction obtains the system serial number. The format is:

```
WIMP$     GI.SSN,buf,siz
```

buf Return buffer address.

siz Size in words of return buffer. (The size must be 3.)

The format of the return buffer is:

word 0 low word of system serial number
word 1 middle word
word 2 high word

Table 8-15: Configuration Table Output Buffer Format

Contents	Byte Offset (decimal)
Header Section	
Table length in bytes	0
Serial number ROM ID	2
High word of serial number	4
Middle word of serial number	6
Low word of serial number	8
Number of option slots	10
Device Section	
Device section length in bytes	12
Slot 0 ID	14
Error/status of slot 0	16
Slot 1 ID	18
Error/status of slot 1	20
Slot 2 ID	22
Error/status of slot 2	24
Slot 3 ID	26
Error/status of slot 3	28
Slot 4 ID	30
Error/status of slot 4	32
Slot 5 ID	34
Error/status of slot 5	36

Contents	Byte Offset (decimal)
Slot 6 ID	38
Error/status of slot 6	40
Slot 7 ID (not used)	42
Error/status of slot 7 (not used)	44
Keyboard ID (supplied by the keyboard, this could be some other input device)	46
Keyboard error/status	48
Base processor ID	50
Base processor error	52
Primary memory ID	54
Primary memory error (low byte) and size (high byte) in 32-kilobyte units	56
Base system ROM ID	58
Base system ROM error	60
Video monitor ID	62
Video monitor error/status	64
Audio device ID (unused)	66
Audio device error/status (unused)	68
Keyboard interface ID (2661)	70
Keyboard interface error/status	72
Printer port interface ID (2661)	74
Printer port interface error/status	76
Console maintenance port ID	78
Console maintenance port status	80
Communication port interface ID	82

Contents	Byte Offset (decimal)
Communication port interface error/status	84
Time of day clock ID	86
Time of day clock error/status	88
Nonvolatile RAM ID	90
Nonvolatile RAM error/status	92
Floating point unit ID	94
Floating point unit error/status	96
Interrupt controller ID	98
Interrupt controller error/status	100
Reserved locations	102
Reserved locations	104
Reserved locations	106
Reserved locations	108
Reserved locations	110
Reserved locations	112
Reserved locations	114
Reserved locations	116
Reserved locations	118
Reserved locations	120
Reserved locations	122
Reserved locations	124
Reserved locations	126
Reserved locations	128
Reserved locations	130

Contents	Byte Offset (decimal)
Reserved locations	132
Boot Section	
Soft restart address	134
Offset value into boot code	136
Booted device ID number	138
Booted device unit number, slot, and type	140
Current boot search return address	142
Error flag for ROM diagnostics	144
Additional Information Section--PC380 Only	
Additional information section length in bytes	146
Screen display information for boot failure, word 3	148
Screen display information for boot failure, word 4	150
Screen display information for boot failure, word 5	152
Screen display information for boot failure, word 6	154
Screen display information for boot failure, word 7	156
Screen display information for boot failure, word 8	158
Scratch memory starting address	160
Return PC for regaining control after software crash	162

Sample High-Level Language Call

The following Basic-Plus-2 program fragment calls the Get Configuration Table subfunction.

```
10      !------------------ Variables  --------------------------
        declare word configtbl(81)      !configuration table buffer
        declare word dsw                !directive status word

        !---------------- Named Constants ----------------------
        declare word constant tblsize = 81 !configtbl
        declare word constant gi.cfg = 12  !wimp subfunction

        !---------------- External Entry -----------------------
        external sub wimp by ref(word,word dim(),word,,,,,word)

        CALL WIMP(gi.cfg,configtbl(),tblsize,,,,,dsw)
```

8.89 WSIG$S - WAIT FOR SIGNIFICANT EVENT

The Wait For Significant Event directive is used to suspend the execution of the issuing task until the next significant event occurs. It is an especially effective way to block a task that cannot continue because of insufficient dynamic memory, since significant events occurring throughout the system often result in the release of dynamic memory. Execution of a Wait For Significant Event directive does not itself constitute a significant event.

High-Level Language Call

```
CALL WFSNE
```

Macro Call

```
WSIG$S  [err]

  err     Error routine address
```

Macro Expansion

```
WSIG$S  ERR
MOV     (PC)+,-(SP)     ;PUSH DPB ONTO THE STACK
.BYTE   49.,1           ;WSIG$S MACRO DIC, DPB SIZE=1 WORD
EMT     377             ;TRAP TO THE EXECUTIVE
BCC     .+6             ;BRANCH IF DIRECTIVE SUCCESSFUL
JSR     PC,ERR          ;OTHERWISE, CALL ROUTINE "ERR"
```

Local Symbol Definitions

None

DSW Return Codes

IS.SUC	Successful completion.
IE.ADP	Part of the DPB is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

Notes

1. If a directive is rejected for lack of dynamic memory, this directive is the only technique available for blocking task execution until dynamic memory can again be available.

2. The wait state induced by this directive is satisfied by the first significant event to occur after the directive has been issued. The significant event that occurs might or might not be related to the issuing task.

3. Because this directive requires only a one-word DPB, the $S form of the macro is recommended. It requires less space and executes with the same speed as that of the DIR$ macro.

4. Significant events include the following:

   - I/O completion.
   - Task exit.
   - Execution of a Send Data directive.
   - Execution of a Send Data, Request and Pass OCB directive.
   - Execution of a Send, Request and Connect directive.
   - Execution of a Send By Reference directive or a Receive by Reference directive.
   - Execution of an Alter Priority directive.
   - Removal of an entry from the clock queue (for instance, resulting from the execution of a Mark Time directive or the issuance of a rescheduling request).
   - Execution of a Declare Significant Event directive.
   - Execution of the round-robin scheduling algorithm at the end of a round-robin scheduling interval.
   - Execution of an Exit, an Exit with Status, or Emit Status directive.

8.90 WTLO$ - WAIT FOR LOGICAL OR OF EVENT FLAGS

The Wait For Logical OR Of Event Flags directive instructs the system to block the execution of the issuing task until the Executive sets the indicated event flags from one of the following groups:

GR 0	Flags 1-16
GR 1	Flags 17-32
GR 2	Flags 33-48
GR 3	Flags 49-64

The task does not block itself if any of the indicated flags are already set when the task issues the directive. See Notes below.

For high-level language calls, you can use the WFLORS routine to receive the Directive Status Word.

High-Level Language Call

```
CALL WFLOR (efn1,efn2,...efnn)

CALL WFLORS (ids,efn1,efn2,...efnn)
```

ids	Integer to receive the Directive Status Word (WFLORS only)
efn	List of event flag numbers taken as the set of flags to be specified in the directive

Macro Call

```
WTLO$   set,mask
```

set	Desired group of event flags
mask	A 16-bit flag mask word

Macro Expansion

```
WTLO$   2,160003
.BYTE   43.,3           ;WTLO$ MACRO DIC, DPB SIZE=3 WORDS
.WORD   2               ;FLAGS GROUP NUMBER 2 (FLAGS 33:48.)
.WORD   160003          ;EVENT FLAGS 33,34,46,47 AND 48.
```

Local Symbol Definitions

None

DSW Return Codes

IS.SUC	Successful completion.
IE.IEF	No event flag specified in the mask word or flag group indicator other than 0, 1, 2, 3, 4, or 5.
IE.ADP	Part of the DPB is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

Notes

1. There is a one-to-one correspondence between bits in the mask word and the event flags in the specified group. That is, if group 1 were specified, then bit 0 in the mask word would correspond to event flag 17, bit 1 to event flag 18, and so forth.

2. The Executive does not arbitrarily clear event flags when Wait For conditions are met. Some directives (Queue I/O Request, for example) implicitly clear a flag; otherwise, they must be explicitly cleared by a Clear Event Flag directive.

3. The set operand must always be of the form n regardless of the macro form used. In all other macro calls, numeric or address values for $S form macros have the form:

   ```
   #n
   ```

   For WTLO$S this form of the set argument would be:

   ```
   n
   ```

4. The argument list specified in the high-level language call must contain only event flag numbers that lie within one event flag group. If event flag numbers are specified that are in more than one group, or if an invalid event flag number is specified, a fatal high-level language error is generated.

5. If the issuing task has outstanding buffered I/O when it enters the Wait For state, it will be stopped. When the task is in a stopped state, it can be checkpointed by any other task regardless of priority. The task is unstopped when:

   - The outstanding buffered I/O completes.
   - The Wait For condition is satisfied.
   - The issuing task exits before the Wait For condition is satisfied.

8.91 WTSE$ - WAIT FOR SINGLE EVENT FLAG

The Wait For Single Event Flag directive instructs the system to block the execution of the issuing task until the indicated event flag is set. If the flag is set at issuance, task execution is not blocked.

High-Level Language Call

```
CALL WAITFR (efn[,ids])
```

efn	Event flag number
ids	Directive status

Macro Call

```
WTSE$   efn
```

efn	Event flag number

Macro Expansion

```
WTSE$   52.
.BYTE   41.,2           ;WTSE$ MACRO DIC, DPB SIZE=2 WORDS
.WORD   52.             ;EVENT FLAG NUMBER 52.
```

Local Symbol Definitions

W.TSEF	Event flag number (2)

DSW Return Codes

IS.SUC	Successful completion.
IE.IEF	Invalid event flag number (EFN<1, or EFN>64).
IE.ADP	Part of the DPB is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

Note

If the issuing task has outstanding buffered I/O when it enters the Wait For state, it will be stopped. When the task is in a stopped state, it can be checkpointed by any other task regardless of priority. The task is unstopped when:

- The outstanding buffered I/O completes.
- The Wait For condition is satisfied.
- The issuing task exits before the Wait For condition is satisfied.

PART III

THE I/O DRIVERS

CHAPTER 9

SYSTEM INPUT/OUTPUT CONVENTIONS

This chapter describes the characteristics, functions, error conditions, and programming suggestions associated with the device drivers supported by the system. If P/OS does not support a particular device, you can develop and maintain your own device driver. (See the *Guide to Writing a P/OS I/O Driver and Advanced Programmer's Notes*.)

Input/output (I/O) operations provide a degree of device independence, while at the same time allowing device-specific operations when required. Programs issue I/O requests to logical units that have been previously associated with particular physical device units.

Each program or task is able to establish its own correspondence between physical device units and logical unit numbers (LUNs). I/O requests are queued as issued; they are subsequently processed according to the issuing task's relative priority.

Tasks can issue I/O requests for appropriate devices by means of the Record Management Services (RMS), or they can interface directly to an I/O driver by means of the Queue I/O (QIO) system directive.

A function code included in the QIO directive indicates the particular input or output operation to be performed. You can use these I/O functions to request such operations as:

- Attaching or detaching a physical device unit for a task's exclusive use
- Reading or writing a logical or virtual block of data
- Cancelling a task's I/O requests

You can specify a wide variety of device-specific I/O operations with QIO directives (for example, reading from a terminal without echoing characters).

9.1 PHYSICAL, LOGICAL, AND VIRTUAL I/O

There are three possible modes in which an I/O transfer can take place: physical, logical, and virtual.

Physical I/O involves reading and writing data in the actual physical units used by the hardware (for example, sectors or tracks).

Logical I/O involves reading and writing data in units of data that are device-independent, such as blocks. That is, your task need not have knowledge of the physical device geometry in order to perform I/O.

When you issue a QIO to a device driver, the driver translates the logical block numbers to physical block numbers. Logical blocks are numbered beginning at 0, and are always 512 (decimal) bytes in length.

Virtual I/O pertains to reading and writing data in open files. When reading and writing data in file-structured devices such as disks, virtual blocks are the same size as logical blocks, and are numbered starting at 1 instead of 0. Virtual blocks provide independence from needing to know where the logical blocks are allocated on the disk for that file.

When you issue a QIO to read or write a virtual block in an open file, the system translates virtual blocks into logical blocks. When you issue a QIO to read or write a virtual block to a non-file-structured device such as a terminal, the Executive changes the QIO from a read-write virtual block to a read-write logical block.

9.2 LOGICAL UNITS

This section describes the construction of the logical unit table and the use of logical unit numbers.

9.2.1 Logical Unit Number

A logical unit number, or LUN, is a number associated with a physical device unit during system I/O operations. A LUN represents an association between a logical unit and a physical device unit.

For example, LUN 1 might be associated with the terminal, LUN 2 with the printer port, LUNs 3 and 4 with the RX50. Once the association has been made, the LUN provides a direct, efficient mapping to the physical device unit, and eliminates the need to search the device tables whenever the system encounters a reference to a physical device unit.

The association is dynamic. Each task running in the system can establish its own correspondence between LUNs and physical device units, and can change any LUN/physical-device-unit association. The flexibility of this association contributes heavily to system device independence.

Keep in mind that reassignment of a LUN at run time causes pending I/O requests for the previous LUN assignment to be cancelled. It is your responsibility to verify that all outstanding I/O requests for a LUN have been serviced before that LUN is associated with another physical device unit. You cannot reassign a LUN if there is an open file associated that LUN.

9.2.2 Logical Unit Table

There is one Logical Unit Table (LUT) for each task running in a system. This table is a variable-length block contained in the task header. Each LUT contains sufficient two-word entries for the number of logical units specified at task-build time by the "UNITS=" option.

Each entry or slot contains a pointer to the physical device unit currently associated with that LUN. Whenever you issue an I/O request, the system matches the appropriate physical device unit to the LUN specified in the call by indexing into the LUT by the number supplied as the LUN.

For example, if the call specifies 6 as the LUN, the system accesses the sixth two-word entry in the LUT and associates the I/O request with the physical device unit to which the entry points. The number of LUN assignments valid for a task ranges from 0 to 255 (decimal), but cannot be greater than the number of LUNs specified at task-build time.

9.2.3 Changing LUN Assignments

Logical unit numbers have no significance until they are associated with a physical device unit by means of one of the methods described below:

1. At task-build time, you can specify an ASG= keyword option, which associates a physical device unit with a LUN referenced in the task being built.

2. At run time, a task can dynamically change a LUN assignment by issuing the Assign LUN system directive, which changes the LUN association LUN with a physical device unit during task execution.

NOTE

If you do not assign any LUNs, the Task Builder by default assigns LUNs 1 through 6. See the *RSX-11M/M-PLUS and Micro/RSX Task Builder Manual* for details.

9.3 ISSUING AN I/O REQUEST

User tasks perform I/O in the system by submitting requests for I/O service in the form of QIO or QIO And Wait system directives.

In this system, and in most multiprogramming systems, tasks normally do not directly access physical device units. Instead, they use input/output services provided by the Executive, since the Executive can effectively share the use of physical device units among many users.

The Executive routes I/O requests to the appropriate device driver and queues them according to the priority of the requesting task. I/O operations proceed concurrently with other activities in the system.

Before a request is queued, it must pass acceptance tests administered by the Executive. If the request fails, the Executive sets the C-bit. You should check for directive rejection by following the QIO directive with a BCS instruction. An I/O operation can also fail even though the directive request succeeded. You should also check the status block associated with the QIO request.

After an I/O request has been queued, the system does not wait for the operation to complete. If at any time the user task that issued the QIO request cannot proceed until the I/O operation has completed, it should specify an event flag (see Chapter 3) in the QIOW$ request. The task then waits for completion of I/O, which is signaled when the event flag is set and the Wait For condition is satisfied.

A QIO directive must supply sufficient information to identify and queue the I/O request. You should also include locations to receive error/status codes. You can optionally specify the address of an asynchronous system trap service routine. Certain types of I/O operations require the specification of device-dependent information as well.

Typical QIO parameters are the following:

- I/O function to be performed.
- Logical unit number associated with the physical device unit to be accessed.
- Optional event flag number for synchronizing I/O completion processing (required for QIOW).
- Optional address of the I/O status block to which information indicating successful or unsuccessful completion is returned.
- Optional address of an asynchronous system trap service routine to be entered on completion of the I/O request.
- Optional device- and function-dependent parameters specifying such items as the starting address of a data buffer, the size of the buffer, and a block number.

A set of system macros that facilitates issuing QIO directives is supplied with the system. These macros, which reside in the System Library Account in (RSXMAC.SML), must be made available to the source program by means of the MACRO-11 Assembler directive .MCALL.

During expansion of a QIO macro, a value of 0 is defaulted for all null (omitted) parameters. Inclusion of the device- and function-dependent parameters depends on the physical device unit and function specified. If you want to specify only an I/O function code, a LUN, and an address for an asynchronous system trap service routine, the following might be issued:

```
QIO$C   IO.ATT,6,,,,ASTOX
```

The I/O function code for attach: IO.ATT

The LUN: 6

The AST address: ASTOX

Null arguments for the event flag number, the request priority, and the address of the I/O status block are indicated by consecutive commas.

No additional device- or function-dependent parameters are required for an attach function. The C form of the QIO$ macro is used here.

For convenience, any comma can be omitted if no parameters appear to the right of it. Therefore, the command above could be issued as follows, if the asynchronous system trap is not desired:

```
QIO$C  IO.ATT,6
```

All extra commas have been dropped. However, if a parameter appears to the right of any place-holding comma, that comma must be retained.

9.3.1 QIO Macro Format

The arguments for a specific QIO macro call can be different for each I/O device accessed and for each I/O function requested. The general format of the call is common to all devices and is as follows:

```
QIO$C  fnc,lun,[efn],[pri],[isb],[ast][,<p1,p2,...,p6>]
```

If function-dependent parameters <p1,...,p6> are required, these parameters must be enclosed within angle brackets (<>). The following paragraphs summarize the use of each QIO parameter.

The fnc parameter is a symbolic name representing the I/O function to be performed. This name is of the form

```
IO.xxx
```

xxx

Identifies the particular I/O operation

For example, a QIO request to attach the physical device unit associated with a LUN specifies the function code IO.ATT.

A QIO request to cancel (or kill) all I/O requests for a specified LUN begins in the following way:

```
QIO$C  IO.KIL,...
```

The fnc parameter specified in the QIO request is stored internally as a function code in the high-order byte and modifier bits in the low-order byte of a single word. The function code is in the range 0 through 31 and is a binary value supplied by the system to match the symbolic name specified in the QIO request.

The correspondence between global symbolic names and function codes is defined in the system object module library, which is automatically searched by the Task Builder. Local symbolic definitions can also be obtained by the FILIO$ and SPCIO$ macros, which reside in the System Macro Library and are summarized in Appendix A.

Several similar functions can have identical function codes, and can be distinguished only by their modifier bits. Only the modifier bits for these two operations are stored differently.

The lun parameter represents the logical unit number (LUN) of the associated physical device unit to be accessed by the I/O request. The association between the physical device unit and the LUN is specific to the task that issues the I/O request, and the LUN reference is usually device-independent. An attach request to the physical device unit associated with LUN 14 (decimal) begins in the following way:

```
QIO$C  IO.ATT,14.,...
```

Because each task has its own LUT in which the physical device unit-LUN correspondences are established, the validity of a LUN parameter is specific to the task that includes this parameter in a QIO request. In general, the LUN must be in the following range:

```
0 < LUN < length of task's LUT (if nonzero)
```

The number of LUNs specified in the LUT of a particular task cannot exceed 255.

The efn parameter is a number representing the event flag to be associated with the I/O operation. It can optionally be included in a QIO, and is required for a QIO And Wait request.

The specified event flag is cleared when the I/O request is queued; it is set when the I/O operation is completed. If the task has issued the QIO And Wait directive, execution is automatically suspended until the I/O completes. If a QIO directive has been issued (with no Wait For directive), then task execution proceeds in parallel with the I/O.

When the task continues to execute, it can test the event flag whenever it chooses by using the Read All Event Flags system directive or the Read Extended Flags system directive (for all event flags). The optional event flag number must be in the range 1 through 64 (decimal). If an event flag specification is not desired, efn can be omitted or can be supplied with a value of 0.

See Chapter 3 for details on event flags and significant events.

NOTE

If an event flag is not specified in a QIOW request, the Executive treats the directive as if it were a simple QIO request.

An I/O request automatically assumes the priority of the requesting task.

The optional isb parameter identifies the address of the I/O status block (I/O status double-word) associated with the I/O request. Figure 9-1 shows the format of an I/O status block (IOSB).

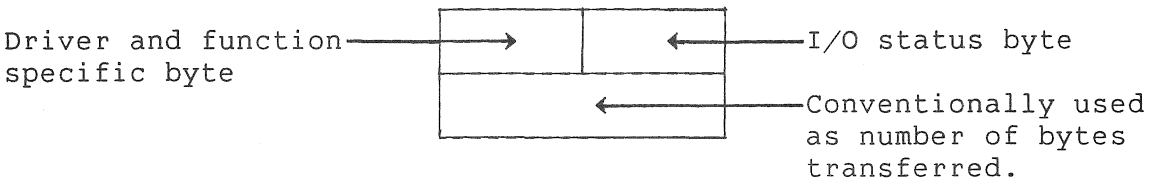


Figure 9-1: Format of I/O Status Block

The IOSB is a two-word array in which an I/O status is returned on completion of the operation.

The low byte of the first IOSB word, is a status code corresponds to a symbolic name of the form IS.xxx (for successful returns) or IE.xxx (for error returns). You can use one of the status symbols to test for a particular error. For example, the symbolic status IE.BAD is returned if a bad parameter is encountered.

Appendix B lists the status codes.

The following illustrates the examination of the I/O status block, IOSB, to determine if a bad parameter has been detected:

```
QIO$C   IO.ATT,14.,2,,IOSB
BCS     DIRERR
WTSE$C  2
        .
        .
        .
CMPB    #IS.SUC,IOSB
BNE     ERROR
```

The correspondence between global symbolic names and I/O completion codes is defined in the system object module library, which is automatically searched by TKB.

Certain device-dependent information is returned to the high-order byte of the first word of isb on completion of the I/O operation. If a read or write operation is successful, the second word is also significant. For example, in the case of a read function on a terminal, the number of bytes transferred are contained in the second word of the IOSB.

The status block can be omitted from a QIO request if you do not intend to test for successful completion of the request.

The optional ast parameter specifies the address of an asynchronous system trap service routine. This routine is entered when when the the I/O request completes.

Chapter 3 describes traps in detail.

The additional QIO parameters, <p1,p2,...,p6>, depend on the particular function and device specified in the I/O request. Typical parameters can include I/O buffer address, I/O buffer length, and so forth. Between zero and six parameters can be included, depending on the particular I/O function.

9.3.2 I/O-RELATED ASTs

Using the AST routine to service I/O-related events provides a response time that is much better than a polling mechanism, and provides for better overlap processing than the simple QIO and Wait for sequence. Asynchronous system traps also provide an ideal mechanism for use in multiple buffering of I/O operations.

All ASTs are inserted in a FIFO queue on a per-task basis as they occur; that is, the event that they are to signal has expired. They are effected one at a time whenever the task does not have

ASTs disabled and is not already in the process of executing an AST service routine.

The process of effecting an AST involves storing certain information on the task's stack, including the task's Wait For mask word and address, the Directive Status Word (DSW), the PS, the PC and any trap-dependent parameters. The task's general-purpose registers R0-R5 are not saved, and thus it is the responsibility of the AST service routine to save and restore the registers it uses.

After an AST is processed, the trap-dependent parameters (if any) must be removed from the task's stack and an AST Service Exit directive executed. The ASTX$S directive macro is used to issue the AST Service Exit directive.

9.4 DIRECTIVE PARAMETER BLOCKS

The DPB for a QIO directive has a length of 12 words. It is generated as the result of expanding a QIO macro call. The first byte of the DPB contains the directive identification code (DIC)--always 1 for QIO. The second byte contains the size of the DPB in words--always 12.

At run time, the Executive uses the arguments stored in each DPB to create, for each request, an I/O packet in system dynamic storage. The packet is entered by priority into a queue of I/O requests for the specified physical device unit. This queue is created and maintained by the Executive and is ordered by the priority of the tasks that issued the requests. The I/O drivers examine their respective queues for the I/O request with the highest priority capable of being executed. This request is dequeued (removed from the queue) and the I/O operation is performed. The process is then repeated until the queue is emptied of all requests.

After the I/O request has been completed, the Executive declares a significant event and may set an event flag, transfer execution to an AST routine, return the I/O status, depending on the arguments specified in the original QIO macro call. Figure 9-2 illustrates the layout of a sample DPB.

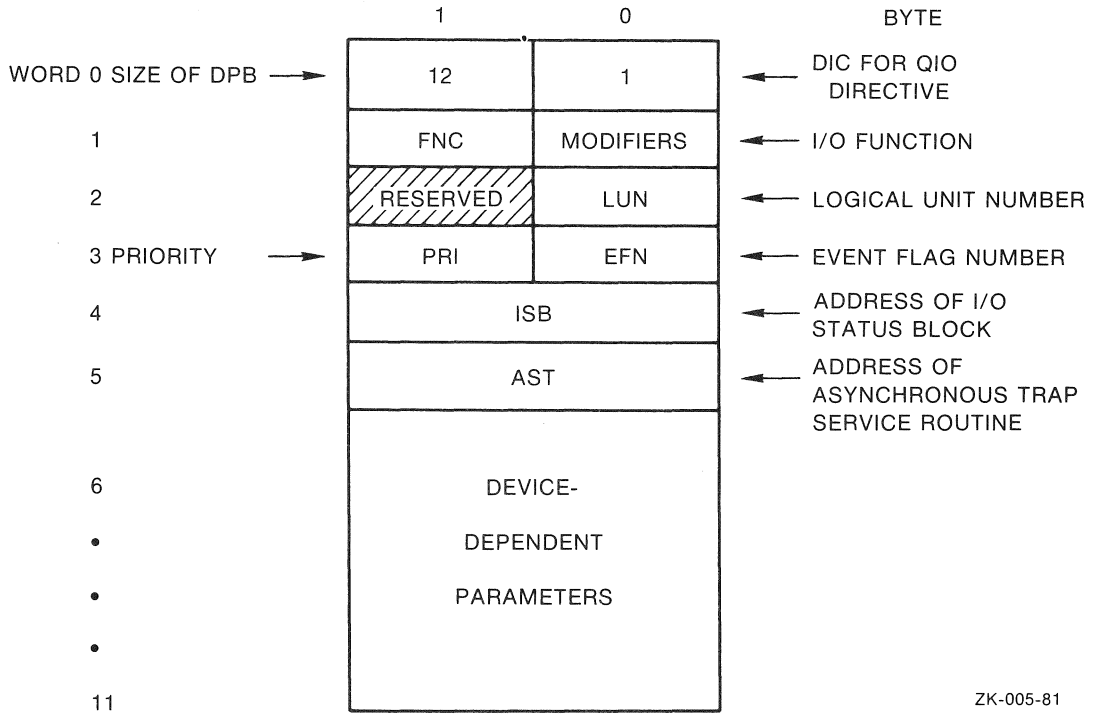


Figure 9-2: QIO Directive Parameter Block

9.5 I/O-RELATED MACROS

Several system macros issue I/O requests and return information about I/O requests. These macros reside in the System Macro Library, and are made available to your program during assembly by the MACRO-11 assembler directive .MCALL.

9.5.1 The QIO$ Macro

You use QIO$ to request an I/O operation and supply parameters for that request Section 8.45 describes the QIO$ directive macro. Also, Section 7.3.1 describes the three forms of this directive macro.

The following example uses the DIR$ macro to actually generate the code to execute the QIO$ directive. DIR$ provides no QIO parameters of its own, but references the QIO directive parameter block at address QIOREF by supplying this label as an argument.

```
QIOREF:  QIO$    IO.RLB,6,2,,,AST01,<BUFFER,80.>

.                ;CREATE QIO DPB

.

.
READ1:   DIR$    #QIOREF                    ;ISSUE I/O REQUEST

.

.

.
READ2:   DIR$    #QIOREF                    ;ISSUE I/O REQUEST
```

9.5.2 The QIOW$ Macro

The QIOW$ macro is equivalent to a QIO$ followed by a WTSE$. It is more economical to issue a QIO And Wait request than to use the two separate macros. An event flag (efn parameter) must be specified with QIOW$ if you actually want to wait.

9.5.3 The ALUN$ Macro

The Assign LUN macro associates a logical unit number with a physical device unit at run time.

Assign LUN does not request I/O for the physical device unit, nor does it attach the unit for exclusive use by the issuing task. It simply establishes a LUN-physical device unit relationship, so that when the task requests I/O for a particular LUN, the associated physical device unit is referenced.

The *Tool Kit User's Guide* describes the device names that you can specify in a call to ALUN$.

The following example associates LUN 10 with terminal unit 1:

```
ALUN$C  10.,TT,1
```

The next example illustrates the use of the three forms of the ALUN$ macro.

```
;
; DATA DEFINITIONS
;

ASSIGN: ALUN$   10.,TT,2        ; GENERATE DPB
        .
        .
        .

;
; EXECUTABLE SECTION
;

        DIR$    #ASSIGN         ; EXECUTE DIRECTIVE
        .
        .
        .
        ALUN$C  10.,TT,2        ; GENERATE DPB IN SEPARATE
        .                       ; P-SECTION, THEN GENERATE
        .                       ; CODE TO EXECUTE
        .                       ; THE DIRECTIVE
        ALUN$S  #10.,#"TI,#0    ; GENERATE DPB ON STACK, THEN
                                ; EXECUTE DIRECTIVE
```

9.5.4 The GLUN$ Macro

The Get LUN Information macro requests that information about a LUN-physical device unit association be returned in a six-word buffer specified by the issuing task.

The following example to requests information about the disk unit associated with LUN 8,:

```
GLUN$C   8.,IOBUF
```

The next example illustrates use of the three forms of the GLUN$ macro.

```
;
; DATA DEFINITIONS
;

GETLUN: GLUN$   6,DSKBUF        ; GENERATE DPB
        .
        .
        .

;
; EXECUTABLE SECTION
;

        DIR$    #GETLUN         ; EXECUTE DIRECTIVE
        .
        .
        .
        GLUN$C  6,DSKBUF        ; GENERATE DPB IN SEPARATE
        .                       ; P-SECTION, THEN GENERATE
        .                       ; CODE TO EXECUTE
        .                       ; THE DIRECTIVE
        GLUN$S  #6,#DSKBUF      ; GENERATE DPB ON STACK, THEN
                                ; EXECUTE DIRECTIVE
```

9.5.5 The ASTX$S Macro

The AST Service Exit macro terminates execution of an AST service routine. Use the S-form.

9.5.6 The WTSE$ Macro

The Wait For Single Event Flag macro instructs the system to suspend execution of the issuing task until the event flag specified in the macro call is set. This macro is useful in synchronizing activity on completion of an I/O operation.

WTSE$ causes the task to be blocked from execution until the specified event flag is set. Frequently, an efn parameter is also included in a QIO$ macro call, and the event flag is set on completion of the I/O operation specified in that call.

The following example illustrates task blocking until the setting of the specified event flag occurs. This example also illustrates the use of the three forms of the macro call.

```
;
; DATA DEFINITIONS
;

WAIT:   WTSE$   5               ; GENERATE DPB
IOSB:   .BLKW   2               ; I/O STATUS BLOCK
        .
        .
        .
;
; EXECUTABLE SECTION
;
        ALUN$S  #14.,#"DW,#1    ; ASSIGN LUN 14 TO DW1:
        QIO$C   IO.ATT,14.,5    ; ATTACH DEVICE
        DIR$    #WAIT           ; EXECUTE WAIT FOR DIRECTIVE
        .
        .
        .
        QIO$S   #IO.RLB,#14.,#2,,#IOSB,,<#BUF,#80.>
        .                       ; READ RECORD, USE EFN2
        .
        .
        WTSE$S  #2              ; WAIT FOR READ TO COMPLETE
        .
        .
        .
        QIO$C   IO.WLB,14.,3,,IOSB,,<BUF,80.>
        .                       ; WRITE RECORD, USE EFN3
        .
        .
        WTSE$C  3               ; WAIT FOR WRITE TO COMPLETE
        .
        .
        .
        QIO$C   IO.DET,14.      ; DETACH DEVICE
        .
        .
        .
```

9.6 STANDARD I/O FUNCTIONS

The number of I/O operations that can be specified by means of the QIO directive is large. A particular operation can be requested by including the appropriate function code as the first parameter of a QIO macro call.

Certain functions are standard. These functions are almost totally device independent and can thus be requested for nearly every device described in this manual. Others are device dependent and are specific to the operation of only one or two I/O devices. This section summarizes the function codes and characteristics of the following device-independent I/O operations:

- Attaching to an I/O device
- Detaching from an I/O device
- Cancelling I/O requests
- Reading a logical block
- Reading a virtual block
- Writing a logical block
- Writing a virtual block

NOTE

In the following descriptions, the five QIO directive parameters lun, efn, pri, isb, and ast are represented by an ellipsis (...).

9.6.1 IO.ATT: Attaching to an I/O Device

The function code IO.ATT is specified by a user task when that task requires exclusive use of an I/O device. This function code is included as the first parameter of a QIO macro call in the following way:

```
QIO$C   IO.ATT,...
```

Successful completion of an IO.ATT request causes the specified physical device unit to be dedicated for exclusive use by the issuing task. This enables the task to process input or output in an unbroken stream and is especially useful on sequential, non-file-oriented devices such as terminals.

An attached physical device unit remains under control of the issuing task until it is explicitly detached by that task, or when the task exits.

size

The data buffer size in bytes.

pn

One to four optional parameters, used to specify such additional information as block numbers for certain devices.

9.6.5 IO.RVB: Reading a Virtual Block

The function code IO.RVB is used to read a virtual block of data from a file on a file-structured device. For a file on a on sequential, record-oriented, non-file-structured device, IO.RVB is converted to IO.RLB before being issued.

NOTE

Any subfunction bits specified in the IO.RVB request are stripped off in this conversion.

We recommend that all tasks use virtual, rather than logical reads when a subfunction is unnecessary. However, if a virtual read is issued for a file-structured device (disk), your task must ensure that a file is open on the specified physical device unit.

This function code is included as the first parameter of a QIO macro call in the following way:

```
QIO$C   IO.RVB,...,<stadd,size,pn>
```

stadd

The starting address of the data buffer.

size

The data buffer size in bytes.

pn

One to four optional parameters, used to specify such additional information as block numbers for certain devices.

9.6.6 IO.WLB: Writing a Logical Block

The function code IO.WLB is specified by a task to write a block of data to the physical device unit specified in the macro call.

NOTE

On P/OS Server systems, a privileged task running on a workstation cannot perform Write Logical Block operations to remote server disks.

Include IO.WLB as the first parameter of a QIO macro call as follows:

```
QIO$C   IO.WLB,...,<stadd,size,pn>
```

stadd

The starting address of the data buffer.

size

The data buffer in bytes.

pn

One to four optional parameters, used to specify such additional information as block numbers or format control characters for certain devices.

9.6.7 IO.WVB: Writing a Virtual Block

The function code IO.WVB is used to write a virtual block of data to a file on a file-structured device.

For sequential, record-oriented, non-file-structured devices such as terminals and line printers, the function IO.WVB is converted to IO.WLB.

NOTE

Any subfunction bits specified in the IO.WVB request are stripped off in this conversion.

We recommend that all tasks use virtual rather than logical writes. However, if a virtual write is issued for a file-structured device (disk), you must ensure that a file is open on the specified physical device unit.

This function code is included as the first parameter of a QIO macro call in the following way:

```
QIO$C   IO.WVB,...,<stadd,size,pn>
```

stadd

The starting address of the data buffer.

size

The data buffer size in bytes.

pn

One to four optional parameters, used to specify such additional information as block numbers or format control characters for certain devices.

9.7 I/O COMPLETION

When an I/O request has been completed, either successfully or unsuccessfully, one or more actions can be taken by the Executive. Selection of return conditions depends on the parameters included in the QIO macro call. There are three major returns:

1. A significant event is declared on completion of an I/O operation. If an efn parameter was included in the I/O request, the corresponding event flag is set.

2. If an isb parameter was specified in the QIO macro call, the I/O status is returned. A carry clear return from the directive itself simply means that the directive was accepted and the I/O request was queued, not that the actual input/output operation was successfully performed.

3. If an ast parameter was specified in the QIO macro call, execution transfers to the AST service routine that begins at the location identified by ast occurs on completion of the I/O operation. See Chapter 3 for a detailed description of AST service routines.

9.8 RETURN CODES

There are two kinds of status conditions recognized and handled by the system when they occur in I/O requests:

- Directive conditions, which indicate the acceptance or rejection of the QIO directive itself
- I/O status conditions, which indicate the success or failure of the I/O operation

Directive conditions relevant to I/O operations can indicate any of the following:

- Directive acceptance
- Invalid buffer specification
- Invalid efn parameter
- Invalid lun parameter
- Invalid DIC number or DPB size
- Unassigned LUN
- Insufficient memory

A code indicating the acceptance or rejection of a directive is returned to the Directive Status Word at symbolic location $DSW. This location can be tested to determine the type of directive condition.

I/O conditions indicate the success or failure of the I/O operation specified in the QIO directive. I/O driver errors include such conditions as device not ready, privilege violation, file already open, or write-locked device.

If an isb parameter is included in the QIO directive, identifying the address of a two-word I/O status block, an I/O status code is returned in the low-order byte of the first word of this block on completion of the I/O operation. This code is a binary value corresponding to a symbolic name of the form IS.xxx or IE.xxx.

The low-order byte of the word can be tested symbolically, by name, to determine the type of status return. The correspondence between global symbolic names and directive and I/O completion status codes is defined in the system object module library. Local symbolic definitions can also be obtained by the DRERR$ and IOERR$ macros.

Binary values of status codes always have the meanings shown in Table 9-1.

Table 9-1: Meaning of Status Code Binary Values

Code	Meaning
Positive (greater than 0)	Successful completion
0	Operation still pending
Negative	Unsuccessful completion

A pending operation means that the I/O request is still in the queue of requests for the respective driver, or the driver has not yet completely serviced the request.

9.8.1 Directive Conditions

Table 9-2 summarizes the directive conditions that can be encountered in QIO directives. The acceptance condition is first, followed by error codes indicating various reasons for rejection, in alphabetical order. (See Appendix B for a summary of error codes.)

Table 9-2: Directive Conditions

Symbolic	Description
IS.SUC	**Directive accepted.** The first six parameters of the QIO directive were valid, and sufficient dynamic memory was available to allocate an I/O packet. The directive is accepted.
IE.ADP	**Invalid address.** The I/O status block or the QIO DPB was outside of the issuing task's address space or was not aligned on a word boundary.

Symbolic	Description
IE.HWR	**Device handler not resident.** The driver for the requested device was not loaded in memory.
IE.IEF	**Invalid event flag number.** The efn specification in a QIO directive was less than 0 or greater than 96.
IE.ILU	**Invalid logical unit number.** The lun specification in a QIO directive was invalid for the issuing task. For example, there were only five logical unit numbers associated with the task, and the value specified for lun was greater than 5.
IE.PRI	**Privilege violation.** The user does not have the required privilege for the requested operation.
IE.SDP	**Invalid DIC number or DPB size.** The directive identification code (DIC) or the size of the Directive Parameter Block (DPB) was incorrect; the legal range for a DIC is from 1 through 127, and all DIC values must be odd. Each individual directive requires a DPB of a certain size. If the size is not correct for the particular directive, this code is returned. The size of the QIO DPB is always 12 words.
IE.ULN	**Unassigned LUN.** The logical unit number in the QIO directive was not associated with a physical device unit. The user may recover from this error by issuing a valid Assign LUN directive and then reissuing the rejected directive.
IE.UPN	**Insufficient dynamic memory.** There was not enough dynamic memory to allocate an I/O packet for the I/O request. The user can try again later by blocking the task with a Wait for Significant Event directive. Note that Wait For Significant Event is the only effective way for the issuing task to block its execution, since other directives that could be used for this purpose themselves require dynamic memory for their execution (for example, Mark Time).

9.8.2 I/O Status Conditions

The following list summarizes status codes that can be returned in the I/O status block specified in the QIO directive on

completion of the I/O request. The I/O status block is a two-word block with the following format:

- The low-order byte of the first word receives a status code of the form IS.xxx or IE.xxx on completion of the I/O operation.
- The high-order byte of the first word is usually device dependent. In cases where you might find information in this byte helpful, this manual identifies that information.
- The second word contains the number of bytes transferred or processed if the operation is successful and involves reading or writing.

If the isb parameter of the QIO directive is omitted, this information is not returned.

The following illustrates a sample two-word I/O status block on completion of a terminal read operation:

	Byte 1	Byte 0
Word 0	0	-10
Word 1	Number of bytes read	

where -10 is the status code for IE.EOF (end of file). If this code is returned, it indicates that input was terminated by typing CTRL/Z, which is the end-of-file termination sequence on a terminal.

To test for a particular error condition, you generally compare the low-order byte of the first word of the I/O status block with a symbolic value, as in the following:

```
CMPB    #IE.DNR,IOSB
```

However, to test for certain types of successful completion of the I/O operation, the entire word value must be compared. For example, if a carriage return terminates a line of input from the terminal, a successful completion code of IS.CR is returned in the I/O status block. If an Escape character is the terminator, a code of IS.ESC is returned. To check for these codes, you should first test the low-order byte of the first word of the block for IS.SUC and then test the full word for IS.CC, IS.CR, IS.ESC, or IS.ESQ.

Note that both of the following comparisons will test as equal since the low-order byte in both cases is +1.

```
CMP     #IS.CR,IOSB

CMPB    #IS.SUC,IOSB
```

In the case of a successful completion where the carriage return is the terminal indicator (IS.CR), the following illustrates the status block:

	Byte 1	Byte 0
Word 0	15	+1
Word 1	Number of bytes read (excluding the CR)	

where 15 is the octal code for carriage return and +1 is the status code for successful completion.

The codes described in Table 9-3 are general status codes that apply to the majority of devices described in subsequent chapters. Error codes specific to only one or two drivers are described only in relation to the devices for which they are returned. The list below describes successful and pending codes first, then error codes in alphabetical order.

Table 9-3: I/O Status Conditions

Symbol	Description
IS.SUC	**Successful completion.** The I/O operation specified in the QIO directive was completed successfully. The second word of the I/O status block can be examined to determine the number of bytes processed, if the operation involved reading or writing.
IS.PND	**I/O request pending.** The I/O operation specified in the QIO directive has not yet been executed. The I/O status block is filled with zeros.
IE.ABO	**Operation aborted.** The specified I/O operation was cancelled with IO.KIL while in progress or while still in the I/O queue.

Symbol	Description
IE.ALN	**File already open.** The task attempted to open a file on the physical device unit associated with the specified LUN, but a file has already been opened by the issuing task on that LUN.
IE.BAD	**Bad parameter.** An illegal specification was supplied for one or more of the device-dependent QIO parameters (words 6-11). For example, a bad channel number or gain code was specified in an analog-to-digital converter I/O operation.
IE.BBE	**Bad block on device.** One or more bad blocks were found by executing the BAD utility. Data cannot be written on bad blocks.
IE.BLK	**Illegal block number.** An illegal block number was specified for a file-structured physical device unit.
IE.BYT	**Byte-aligned buffer specified.** Byte alignment was specified for a buffer, but only word (or double-word) alignment is legal for the physical device unit. For example, a disk function requiring word alignment was requested, but the buffer was aligned on a byte boundary.
IE.DAA	**Device already attached.** The physical device unit specified in an IO.ATT function was already attached to the issuing task. This code indicates that the issuing task has already attached the desired physical device unit, not that the unit was attached by another task.
IE.DNA	**Device not attached.** The physical device unit specified in an IO.DET function was not attached to the issuing task. This code has no bearing on the attachment status with respect to other tasks.
IE.DNR	**Device not ready.** The physical device unit specified in the QIO directive was not ready to perform the desired I/O operation. This code is often returned as the result of an interrupt time-out; that is, a "reasonable" amount of time has passed, and the physical device unit has not responded.
IE.EOF	**End-of-file encountered.** An end-of-file mark, record, or control character was recognized on the input device.

Symbol	Description
IE.FHE	**Fatal hardware error.** Controller is physically unable to reach the location where input/output is to be performed on the device. The operation cannot be completed.
IE.IFC	**Illegal function.** A function code was specified in an I/O request that was illegal for the specified physical device unit. This code is returned if the task attempts to execute an illegal function or if, for example, a read function is requested on an output-only device, such as the line printer.
IE.NLN	**File not open.** The task attempted to close a file on the physical device unit associated with the specified LUN, but no file was currently open on that LUN.
IE.NOD	**Insufficient buffer space.** Dynamic storage space has been depleted, and there was insufficient buffer space available to allocate a secondary control block. For example, if a task attempts to open a file, buffer space for the window and file control block must be supplied by the Executive. This code is returned when there is not enough space for such an operation.
IE.OFL	**Device off line.** The physical device unit associated with the LUN specified in the QIO directive was not on line. When the system was booted, a device check indicated that this physical device unit was not in the configuration.
IE.OVR	**Illegal read overlay request.** A read overlay was requested and the physical device unit specified in the QIO directive was not the physical device unit from which the task was installed. The read overlay function can only be executed on the physical device unit from which the task image containing the overlays was installed.
IE.PRI	**Privilege violation.** The task that issued a request was not privileged to execute that request.
IE.SPC	**Illegal address space.** The buffer requested for a read or write request was partially or totally outside the address space of the issuing task. Alternately, a byte count of 0 was specified.

Symbol	Description
IE.VER	**Unrecoverable error.** After the system's standard number of retries have been attempted upon encountering an error, the operation still could not be completed.
IE.WCK	**Write check error.** An error was detected during the check (read) following a write operation.
IE.WLK	**Write-locked device.** The task attempted to write on a write-locked physical device unit.

CHAPTER 10

DISK DRIVERS

The system's disk drivers support the disks summarized in Table 10-1. Subsequent sections describe the devices in detail.

Table 10-1: Standard Disk Devices

Drive	RPM	Sectors	Heads	Cylinders	Bytes per Drive	Blocks per Drive
RX50	300	10	2	80 per diskette	819,200	801
RD50	3600	16	4	153 per surface	5 Mb	9728
RD51	3600	16	4	306 per surface	10 Mb	19520
RD52	3600	16	8	511 per surface	33 Mb	65408
RD31	3600	16	4	614 per surface	20 Mb	39295

10.1 RX50 DESCRIPTION

The RX50 subsystem consists of a 5.25-inch dual floppy diskette drive and a separate single-board controller module. The module enables a data processing system to store or retrieve information from any location on one side of each front-loadable diskette.

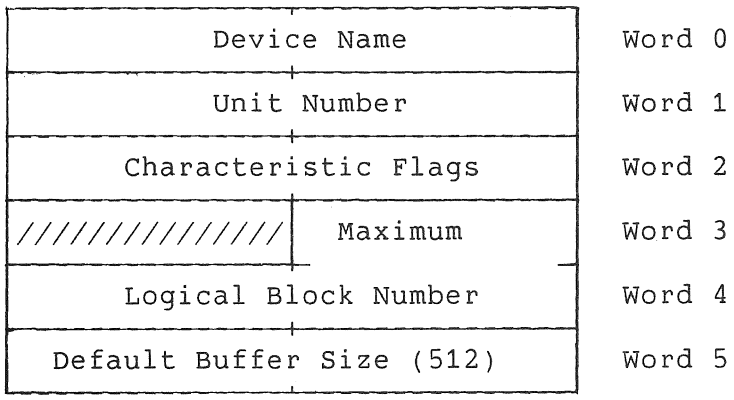
10.2 RD-SERIES DESCRIPTION

The RD31, RD50, RD51, and RD52 ("RD-Series") are Winchester hard disk, multiplatter, random-access devices. They store data in fixed-length blocks on 130mm rigid disk media. Winchester technology uses moving head, noncontact recording. Unlike the RX50, RD-Series storage media cannot be removed from the drive.

10.3 GET LUN INFORMATION FOR DISK DRIVERS

You can invoke the Get LUN Information system directive to obtain device characteristics of a disk device. The directive returns the device characterstics in the last four words of a six-word buffer you supply. Figure 10-1 shows the contents of the buffer after a call to the directive.

Figure 10-1: Get LUN Information Return Buffer



Word 2 of the buffer (the first device characteristic word) contains 16 flags indicating the state of various characteristics. Table 10-2 describes each of these characteristics. (A bit setting of 1 indicates that the described characteristic is true for disks.)

Words 3 and 4 contain the maximum logical block number (LBN) for the disk.

NOTE

As the figure shows, the system uses only the low byte of word 3 and all of word 4, thus leaving the high byte of word 3 undefined. Consequently, you must always clear the high byte of word 3 before using the returned maximum LBN.

Word 5 contains the default buffer size, which is 512 bytes for all disks.

Table 10-2: Get LUN Characteristic Flags for Disks

Bit	Setting	Meaning
0	0	Record-oriented device
1	0	Carriage-control device
2	0	Terminal device
3	1	File-structured device
4	0	Single-directory device
5	0	Sequential device
6	1	Mass storage device
7	1 or 0	User-mode diagnostics supported (device dependent)
8	1	Device supports 22-bit direct addressing
9	0	Unit software write-locked
10	0	Input spooled device
11	0	Output spooled device
12	0	Pseudo-device
13	0	Device mountable as a communications channel
14	1	Device mountable as a Files-11 volume
15	1	Device mountable

10.4 OVERVIEW OF I/O OPERATIONS

The RX50 and RD-series disks used on the Professional are FILES-11 structured and, therefore, compatible with RMS-11. RMS-11 lets you transfer data between your task and FILES-11 structured volumes without having to be concerned with the physical organization of the data on the volume. When transferring data to and from FILES-11 volumes, always use the RMS-11 MACRO or high-level language facilities.

The information in this chapter is intended primarily for those cases where you may not want to be restricted to a FILES-11 data format and thus cannot use RMS-11. Examples of such cases are:

- Performing I/O with CP/M-structured RX50 diskettes
- Creating a special utility, such as for data backup

The QIOs documented in this chapter let you bypass RMS-11 to handle such cases. Be aware that using QIOs instead of RMS-11 makes program development more complex and increases the chances of error.

With some exceptions, all the disks supported on the Professional can be accessed in the same manner. Differences are pointed out in the discussion.

Disks are divided into a series of 256-word blocks, and data is transferred in blocks to and from disks. These transfers can be performed in three possible modes: physical, logical, or virtual. The difference between theses modes is the manner in which the disk is addressed.

(See *PRO/RMS-11: An Introduction* for definitions of physical, logical, and virtual blocks, as well as a generic description of disk geometry.)

10.4.1 Physical I/O Operations

Physical I/O operations are allowed with all the disk devices.

In physical I/O operations, data is read or written to the actual sectors (physical blocks) of the disk. No consideration is given to factors such as the interleave of blocks or the track skew, which can be introduced to compensate for any lag of the read-write head. Consequently, physical blocks are numbered consecutively on any one track. Physical blocks are numbered starting with zero.

The RX50 is a single head device. On the RX50, the address of a physical block is expressed as a track and sector address. The first physical block is on track 0, sector 1.

The RD-Series disks are multiplatter devices. On these, the address of a physical block is derived by first obtaining the sector number within a track, then the track number within a cylinder, and then the cylinder number. The first physical block is on track 0, sector 0.

10.4.2 Logical I/O Operations

Logical I/O operations transfer data to or from the logically addressable blocks of the disk. Unlike physical blocks, logical blocks are not necessarily contiguous.

For the RX50 disk, the sector interleave and track skew are automatically taken into account when a logical READ or WRITE is performed.

Logical blocks are numbered starting with 0. Note the following:

- On the RX50, the first logical block is at track 1, sector 1. The highest-numbered logical block is at track 0, at the highest sector for that track.

- On the RD-Series disks, the first logical block is at track 0, sector 1.

10.4.3 Virtual I/O Operations

Virtual I/O operations have meaning only within the context of a file. The virtual blocks of a file are numbered consecutively, starting with virtual block 1, which is the start of the file. The consecutively numbered virtual blocks of a file map to logical blocks, which might or might not be contiguous. Virtual I/O operations are converted by the file processor into logical READS and WRITES.

10.5 QIO MACRO FUNCTIONS FOR DISK DRIVERS

Table 10-3 lists the functions of the QIO macro that are valid for disks. These are all standard QIO functions.

NOTE

If your task is transferring data to a FILES-11 structured volume, use RMS-11 to perform I/O operations.

Table 10-3: QIO Functions for Disks

Format	Function
QIO$C IO.ATT,...	Attach device [1]
QIO$C IO.DET,...	Detach device
QIO$C IO.KIL,...	Kill I/O [2]
QIO$C IO.RLB,...,<stadd,size,,blkh,blkl>	READ logical block
QIO$C IO.RPB,...,<stadd,size,,blkh,blkl>	READ physical block
QIO$C IO.RVB,...,<stadd,size,,blkh,blkl>	READ virtual block
QIO$C IO.WLB,...,<stadd,size,,blkh,blkl>	WRITE logical block
QIO$C IO.WPB,...,<stadd,size,,blkh,blkl>	WRITE physical block
QIO$C IO.WVB,...,<stadd,size,,blkh,blkl>	WRITE virtual block

Parameters Shown in Table 10-3

stadd — The starting address of the data buffer (must be on a word boundary).

size — The data buffer size in bytes (must be even and greater than 0).

blkh/blkl — Block high and block low, combining to form a double-precision number that indicates the actual logical/virtual block address on the disk where the transfer starts; blkh represents the high 8 bits of the address, and blkl the low 16 bits.

Notes to Table 10-3

1. Only volumes mounted foreign may be attached. Any other attempt to attach a mounted volume will result in an IE.PRI status being returned in the I/O status doubleword. You should not attach any volumes, because this prevents the mount/dismount mechanisms from working and can possibly lead to a program hang or a system hang.)

2. In-progress disk operations are allowed to complete when IO.KIL is received, because they take such a short time. I/O requests that are queued when IO.KIL is received are killed immediately. An IE.ABO status is returned in the I/O status doubleword.)

To perform physical or logical I/O operations with QIOs, your task must be privileged.

IO.RVB and IO.WVB are associated with file operations. For these functions to be executed, a file must be open on the specified LUN if the volume associated with the LUN is mounted. Otherwise, the virtual I/O request is converted to a logical I/O request using the specified block numbers.

Use of the QIO virtual I/O operations requires caution as it is possible to quickly exhaust system resources. Simply writing and reading files with QIOs is not sufficient. The file must be extended with $EXTEND calls to RMS-11, or the end-of-file pointer will not be moved. Reading and writing a file via QIOs without properly setting the FB$SHR field in the RMS-11 FAB block can cause system hangs when system free space is exhausted.

Observe the following if you are using FCS instead of RMS-11:

- When writing a new file using QIOs, the task must explicitly issue .EXTND File Control System library routine calls as necessary to reserve enough blocks for the file, or the file must be initially created with enough blocks allocated for the file.

- The task must put an appropriate value in the FDB for the end-of-file block number (F.EFBK) before closing the file.

Each disk driver supports the subfunction bit IQ.X: inhibit retry attempts for error recovery. The subfunction bit is used by ORing it into the desired QIO; for example:

```
QIO$C IO.WLB!IQ.X,...,<stadd,size,,blkh,blkl>
```

The IQ.X subfunction permits user-specified retry algorithms for applications in which data reliability must be high.

10.6 STATUS RETURNS FOR DISK DRIVERS

The error and status conditions listed in Table 10-4 are returned by the disk drivers described in this chapter.

When a disk I/O error condition is detected, an error is usually not returned immediately. Instead, the system attempts to recover from most errors by retrying the function as many as eight times. Unrecoverable errors are generally parity, timing, or other errors caused by a hardware malfunction.

Table 10-4: Disk Status Returns

Symbol	Description
IS.SUC	**Successful completion.** The operation specified in the QIO directive was completed successfully. The second word of the I/O status block can be examined to determine the number of bytes processed, if the operation involved reading or writing.
IS.PND	**I/O request pending.** The operation specified in the QIO directive has not yet been executed. The I/O status block is filled with zeros.
IE.ABO	**Request aborted.** An I/O request was queued (not yet acted upon by the driver) when an IO.KIL was issued.
IE.ALN	**File already open.** The task attempted to open a file on the physical device unit associated with the specified LUN, but a file has already been opened by the issuing task on that LUN.
IE.BLK	**Illegal block number.** An illegal logical block number was specified.
IE.BBE	**Bad block error.** The disk sector (block) being read was marked as a bad block in the header word.

Symbol	Description
IE.BYT	**Byte-aligned buffer specified.** Byte alignment was specified for a buffer, but only word alignment is legal for disk. Alternatively, the length of a buffer is not an appropriate number of bytes.
IE.DNR	**Device not ready.** The physical device unit specified in the QIO directive was not ready to perform the desired I/O operation.
IE.FHE	**Fatal hardware error.** The controller is physically unable to reach the location where input/output operation is to be performed. The operation cannot be completed.
IE.IFC	**Illegal function.** A function code was specified in an I/O request that is illegal for disks.
IE.MII	**Media inserted incorrectly.** The controller has detected that the media (such as a floppy diskette) were not inserted correctly. To correct the problem, reinsert the media properly.
IE.NLN	**File not open.** The task attempted to close a file on the physical device unit associated with the specified LUN, but no file is currently open on that LUN.
IE.NOD	**Insufficient buffer space.** Dynamic storage space was depleted, and there is insufficient buffer space available to allocate a secondary control block. For example, if a task attempts to open a file, buffer space for the window and file control block must be supplied by the Executive. This code is returned when there is not enough space for this operation.
IE.OFL	**Device off line.** The physical device unit associated with the LUN specified in the QIO directive was not on-line. When the system was booted, a device check indicated that this physical device unit was not in the configuration.
IE.OVR	**Illegal read overlay request.** A read overlay was requested, but the physical device unit specified in the QIO directive was not the physical device unit from which the task was installed. The read overlay function can only be executed on the physical device unit from which the task image containing the overlays was installed.

Symbol	Description
IE.PRI	**Privilege violation.** The task that issued the request was not privileged to execute that request. For disk, this code is returned if a nonprivileged task attempts to read or write a mounted volume directly (that is, using IO.RLB or IO.WLB). Also, this code is returned if any task attempts to attach a mounted volume.
IE.SPC	**Illegal address space.** The buffer specified for a read or write request was partially or totally outside the address space of the issuing task. Alternately, a byte count of zero was specified.
IE.VER	**Unrecoverable error.** After the system's standard number of retries has been attempted upon encountering an error, the operation still could not be completed. For disk, unrecoverable errors are usually parity errors.
IE.WCK	**Write check error.** An error was detected during the write check portion of an operation.
IE.WLK	**Write-locked device.** The task attempted to write on a disk that was write-locked.

CHAPTER 11

THE TERMINAL DRIVER

The Professional terminal driver, called TTDRV, includes the following features:

- Full-duplex operation
- Type-ahead buffering
- Eight-bit characters
- Transparent read and write
- Formatted read and write
- Read after prompt
- Read with no echo
- Read with special terminator
- Optional timeout on solicited input
- Device-independent cursor control

In addition to these features, the terminal driver supports the following hardware components:

- **Bitmap Display**

  The bitmap display is the primary terminal device for the Professional.

- **Printer Port**

  The printer port is an asynchronous port to which a printer or terminal can be connected.

- **Quad Serial Line Unit**

  The Quad Serial Line Unit (SLU) is a hardware option that provides four full-duplex, asynchronous ports. You can use the ports to connect additional serial devices to a Professional.

11.1 GET LUN INFORMATION MACRO FOR TERMINAL DRIVER

Word 2 (the first characteristic word) of the buffer filled by the Get LUN Information system directive contains the information shown in Table 11-1. A setting of 1 indicates that the described characteristic is true for terminals.

Table 11-1: Get LUN Information for Terminal Driver

Bit	Setting	Meaning
0	1	Record-oriented device
1	1	Carriage-control device
2	1	Terminal device
3	0	File-structured device
4	0	Single-directory device
5	0	Sequential device
6	0	Mass storage device
7	0	User-mode diagnostics supported

Bit	Setting	Meaning
8	N/A	Reserved
9	0	Unit software write-locked
10	0	Input spooled device
11	0	Output spooled device
12	0	Pseudo device
13	0	Device mountable as a communication channel
14	0	Device mountable as a FILES-11 volume
15	0	Device mountable

Words 3 and 4 of the buffer are undefined. Word 5 indicates the default buffer size (the width of the terminal carriage or display screen).

11.2 QIO MACRO FOR TERMINAL DRIVER

Table 11-2 lists the standard and driver-specific functions of the QIO macro that are valid for terminals.

Table 11-2: QIO Functions for Terminals

Format	Function
Standard Functions	
QIO$C IO.ATT,...	ATTACH device
QIO$C IO.DET,...	DETACH device
QIO$C IO.KIL,...	CANCEL I/O requests
QIO$C IO.RLB,...,<stadd,size[,tmo]>	READ logical block (read typed input into buffer)

Format	Function
QIO$C IO.RVB,...,<stadd,size[,tmo]>	READ virtual block (read typed input into buffer)
QIO$C IO.WLB,...,<stadd,size,vfc>	WRITE logical block (print buffer contents)
QIO$C IO.WVB,...,<stadd,size,vfc>	WRITE virtual block (print buffer contents)
Device-Specific Functions	
QIO$C IO.ATA,...,<ast,[param2][,ast2]>	ATTACH device, specify unsolicited-input-character AST
QIO$C IO.CCO,...,<stadd,size,vfc>	CANCEL CTRL/O (if in effect), then write logical block
QIO$C IO.GTS,...,<stadd,size>	GET terminal support
QIO$C IO.RAL,...,<stadd,size[,tmo]>	READ logical block, pass all bits
QIO$C IO.RNE,...,<stadd,size[,tmo]>	READ logical block, do not echo
QIO$C IO.RPR,..., <stadd,size,[tmo],pradd,prsize,vfc>	READ logical block after prompt
QIO$C IO.RSD,...,<stadd,size,tmo,type>	READ special data
QIO$C IO.RST,...,<stadd,size[,tmo]>	READ logical block ended by special terminators
QIO$C IO.RTT,...,<stadd,size,[tmo],table>	READ logical block ended by specified special terminator
QIO$C IO.WAL,...,<stadd,size,vfc>	WRITE logical block, pass all bits
QI0$C IO.WBT,...,<stadd,size,vfc>	WRITE logical block, break through most I/O conditions at terminal

Format	Function
QIO$C IO.WSD,...,<stadd,size,,type>	WRITE special data
QIO$C SF.GMC,...,<stadd,size>	GET multiple characteristics
QIO$C SF.SMC,...,<stadd,size>	SET multiple characteristics

Parameters Shown in Table 11-2

ast — The entry point for an unsolicited input character AST.

param2 — A number you can use to identify the terminal that is the input source upon entry to an unsolicited character AST routine.

ast2 — The entry point for an INTERRUPT/DO sequence AST. (See Section 11.4.2.)

pradd — The starting address of the byte buffer where the prompt is stored.

prsize — The size of the pradd prompt buffer in bytes. The specified size must be greater than 0 and less than or equal to 8128. The buffer must be within your task's address space.

size — The size of the stadd data buffer in bytes. The specified size must be greater than 0 and less than or equal to 8128. The buffer must be within your task's address space. For SF.GMC, IO.GTS, and SF.SMC functions, size must be an even value.

stadd — The starting address of the data buffer. The address must be word aligned for SF.GMC, IO.GTS, and SF.SMC, IO.RSD, IO.WSD; otherwise, stadd can be on a byte boundary.

table — The address of the 16-word special terminator table.

tmo — An optional timeout count in 10-second intervals. timeout is the maximum time allowed between two input characters before the driver terminates the read operation. A timeout occurs when the type-ahead buffer is empty and the driver receives no input termination after the specified length of time.

If you specify 0 for the tmo parameter, the read times out immediately after reading any data that can be in the type-ahead buffer.

type The data type of the buffer contents.

vfc A character for vertical format control from Table 11-16.

11.2.1 Using Subfunction Bits

You can use *subfunction bits* to qualify QIO functions. To select one or more subfunctions, you perform a logical OR operation with one or more subfunctions and a QIO function.

For example, the following QIO request IO.RPR uses two subfunction bits to perform a read after prompt that is not echoed (TF.RNE) and terminated by a special terminator character (TF.RST).

```
QIO$C IO.RPR!TF.RNE!TF.RST,...,<stadd,size,,pradd,prsize,vfc>
```

Table 11-3 lists the subfunction bit symbolic names and their corresponding subfunctions.

Table 11-3: Subfunction Bit Symbolic Names and Description

Symbol	Subfunction
TF.AST	Unsolicited-input-character AST.
TF.BIN	Binary prompt.
TF.CCO	Cancel CTRL/O.
TF.ESQ	Recognize escape sequences.
TF.NOT	Unsolicited input AST notification; unsolicited characters are stored in the type-ahead buffer until they are read by the task.
TF.RAL	Read all bits.
TF.RCU	Restore cursor position.
TF.RNE	Read with no echo.

Symbol	Subfunction
TF.RST	Read with special terminators.
TF.TMO	Read with time-out.
TF.WBT	Break through write.
TF.WAL	Write all bits.
TF.XCC	Send an INTERRUPT/DO sequence to the P/OS Dispatcher when the terminal is attached with AST notification.

Table 11-4 shows which subfunction bits are available for each QIO function.

Table 11-4: Subfunction Bits Available for Driver Requests

Function	Equivalent Subfunction	Allowed Subfunction Bits
Standard Functions		
IO.ATT	None	TF.AST, TF.ESQ, TF.NOT, TF.XCC
IO.DET	None	None
IO.KIL	None	None
IO.RLB	None	TF.RAL[1], TF.RNE[1], TF.RST, TF.TMO
IO.RVB	None	None
IO.WLB	None	TF.CCO, TF.RCU, TF.WBT, TF.WAL
IO.WVB	None	None
Driver-Specific Functions		
IO.ATA	IO.ATT!TF.AST	TF.ESQ, TF.NOT, TF.XCC
IO.CCO	IO.WLB!TF.CCO	TF.WAL[2], TF.WBT

Function	Equivalent Subfunction	Allowed Subfunction Bits
IO.GTS	None	None
IO.RAL	IO.RLB!TF.RAL	TF.RNE, TF.RST[1], TF.TMO
IO.RNE	IO.RLB!TF.RNE	TF.RAL[1], TF.RST[1], TF.TMO
IO.RPR	None	TF.BIN, TF.RAL[1], TF.RNE, TF.RST[1], TF.TMO
IO.RSD	None	None
IO.RST	IO.RLB!TF.RST	TF.RAL[1], TF.RNE, TF.TMO
IO.RTT	None	TF.RAL, TF.RNE, TF.RST, TF.TMO
IO.WAL	IO.WLB!TF.WAL	TF.CCO[2], TF.RCU[2], TF.WBT
IO.WBT	IO.WLB!TF.WBT	TF.CCO, TF.RCU, TF.WAL[2]
IO.WSD	None	None
SF.GMC	None	None
SF.SMC	None	None

Notes to Table 11-4

1. Avoid using Read All (TF.RAL) and Read with Special Terminators (TF.RST) in the same request.

2. During a write-pass-all operation (IO.WAL or IO.WLB!TF.WAL) the driver writes characters without interpretation; it does not keep track of cursor position.

If a task invokes a subfunction bit that the system does not support, the driver accepts the QIO request but ignores the subfunction bit.

11.2.2 Driver-Specific QIO Functions

This section describes each of the driver-specific QIO functions.

11.2.2.1 IO.ATA and IO.ATT!TF.AST - IO.ATA is a variation of the Attach function, and is equivalent to IO.ATT logically ORed with the subfunction bit TF.AST.

You use IO.ATA to specify asynchronous system trap (AST) routines to process unsolicited input characters. Use the following request format to attach the terminal and identify entry points (ast and ast2) for an unsolicited-input-character AST:

```
QIO$C   IO.ATA,...,<[ast],[param2][,ast2]>
```

Control passes to ast whenever the driver receives an unsolicited character other than CTRL/Q, CTRL/S, CTRL/X, or CTRL/O.

You must supply a minimum of one AST parameter (ast or ast2) in the QIO request. If you specify the ast2 parameter, an INTERRUPT/DO sequence or CTRL/C character results in the ast2 routine being entered. If you do not specify ast2, an INTERRUPT/DO sequence results in the specified AST being entered in the ast parameter.

Unless you specify the TF.XCC subfunction, your task traps the INTERRUPT/DO or CTRL/C sequence; the sequence does not reach the P/OS Dispatcher. Thus, if your task uses IO.ATA without the TF.XCC subfunction, it should recognize some input sequence as a request to terminate since the P/OS Dispatcher cannot abort your task in case of difficulty.

Note that you can use either ast2 or TF.XCC, but not both in the same QIO request. If you specify both in a QIO request, the system returns an IE.SPC error.

Upon entry to the AST routines, the unsolicited character and param2 are in the top word on the stack, as shown in Table 11-5. Your task must remove the top word from the stack before exiting the AST.

Table 11-5: Task Stack Format

Current Stack Pointer	Contents
SP+10	Event flag mask word
SP+06	PS of task prior to AST
SP+04	PC of task prior to AST
SP+02	Task's directive status word
SP+00	The low byte contains the unsolicited character. The high byte contains the param2 argument that you optionally specified in the QIO request. This argument is useful if you want to identify an individual terminal in a common AST routine.

When you include TF.NOT in the IO.ATA function and the terminal driver receives unsolicited terminal input (except INTERRUPT/DO or CTRL/C), the resulting AST serves only as notification of unsolicited terminal input. Note the following in this case:

- The terminal driver does not pass the character to your task. Upon entry to the AST service routine, the high byte of the first word on the stack identifies the terminal causing the AST (param2). See Table 11-5.

- After the AST has occurred, the AST becomes disabled until your task issues a Read request. If the driver receives multiple characters before your task issues the Read request, then the driver stores those characters in the type-ahead buffer.

- Upon receiving the Read request, the driver returns the contents of the type-ahead buffer, including the character causing the AST, to your task. The driver then enables the AST for new unsolicited input characters. Thus, using the TF.NOT subfunction allows a task to read multiple characters per AST, thereby resulting in a significant system performance increase.

See Chapter 3 for further details on ASTs.

11.2.2.2 IO.ATT!TF.ESQ - The task issuing this function attaches a terminal and notifies the driver that it recognizes escape sequences received from that terminal. The driver recognizes escape sequences only for solicited input. (See Section 11.5 for a description of escape sequences.)

If your task has not declared the terminal capable of generating escape sequences, IO.ATT!TF.ESQ has no effect other than attaching the terminal. The driver does not return a full escape sequence to the task as a terminator.

You must also use the SF.SMC function to declare the terminal capable of generating escape sequences. (See Table 11-7 for a description of the driver-terminal characteristics for the SF.GMC and SF.SMC functions.)

11.2.2.3 IO.CCO and IO.WLB!TF.CCO - IO.CCO is equivalent to IO.WLB logically ORed with the subfunction bit TF.CCO.

This Write function directs the driver to write to the terminal regardless of any CTRL/O condition. If CTRL/O is in effect, the driver cancels it before performing the Write operation.

11.2.2.4 IO.DET - This function detaches the terminal and, in doing so, cancels any CTRL/O condition in effect. It cancels any AST specified when the terminal was attached.

11.2.2.5 IO.GTS - This function is a Get Terminal Support request that returns information to a four-word buffer specifying which features are part of the terminal driver. Only two of these words (words 0 and 1) are currently defined. A system module, TTSYM, defines the various symbols that the IO.GTS, SF.GMC, and SF.SMC functions use. These symbols include:

- F1.xxx and F2.xxx (Table 11-6)
- TC.xxx (Table 11-7)
- T.xxxx (Table 11-9)
- The SE.xxx error returns described in Table 11-11.

The Task Builder automatically defines these symbols.

Table 11-6: Information Returned by IO.GTS

Symbol	Meaning When Set to 1
Word 0 of Buffer	
F1.ACR	Automatic CR/LF on long lines
F1.BUF	Checkpointing during terminal input
F1.UIA	Unsolicited-input-character AST
F1.CCO	Cancel CTRL/O before writing
F1.ESQ	Recognize escape sequences in solicited input
F1.LWC	Lower- to uppercase conversion
F1.RNE	Read with no echo
F1.RPR	Read after prompting
F1.RST	Read with special terminators
F1.RUB	CRT rubout
F1.TRW	Read all and write all
F1.UTB	Input characters buffered in task's address space
F1.VBF	Variable-length terminal buffers
Word 1 of Buffer	
F2.SCH	Set characteristics QIO (SF.SMC)
F2.GCH	Get characteristics QIO (SF.GMC)
F2.SFF	Formfeed can be simulated
F2.CUP	Cursor positioning
F2.FDX	Full Duplex Terminal Driver

11.2.2.6 IO.RAL and IO.RLB!TF.RAL - IO.RAL is equivalent to IO.RLB logically ORed with the subfunction bit TF.RAL.

The Read All function causes the driver to pass all bits to the requesting task. The driver does not intercept control characters or mask out the high-order bit. For example, the driver passes the sequences CTRL/Q, CTRL/S, CTRL/O, CTRL/Z and CTRL/C (which is the same as INTERRUPT/DO) to your task without interpretation.

Note that IO.RAL echoes the characters that are read. To read all bits without echoing, use IO.RAL!TF.RNE.

The IO.RAL function terminates either when a full character count (input buffer full) or a timeout occurs.

11.2.2.7 IO.RNE and IO.RLB!TF.RNE - The IO.RNE function is equivalent to IO.RLB logically ORed with the subfunction bit TF.RNE.

This function reads terminal input characters without echoing the characters back to the terminal for immediate display. Use this feature to read sensitive information, such as a password or combination.

(Note that you can select the no-echo mode with the SF.SMC function. See Table 11-7, bit TC.NEC.)

11.2.2.8 IO.RPR - The IO.RPR Read After Prompt functions as an IO.WLB (to write a prompt to the terminal) followed by IO.RLB. However, IO.RPR differs from this combination of functions as follows:

- Performance is better with the IO.RPR because the system processes only one QIO.
- The terminal driver cancels any CTRL/O that is in effect prior to issuing the IO.RPR.

You can logically OR subfunction bits with IO.RPR to write the prompt as a Write All (TF.BIN). In addition, you can use the Read subfunction bits TF.RAL, TF.RNE, and TF.RST with IO.RPR.

11.2.2.9 IO.RPR!TF.BIN - This function results in a Read after a binary prompt; that is, the driver writes a prompt with no character interpretation (as if it were issued as an IO.WAL). TF.BIN exists because TF.WAL and TF.RAL are the same bit.

11.2.2.10 IO.RST and IO.RLB!TF.RST - IO.RST is equivalent to IO.RLB logically ORed with the TF.RST subfunction bit.

This function is similar to an IO.RLB, except that certain special characters terminate the Read. These characters are in the ranges 0-037 and 175-177. The driver does not interpret the terminating character, with certain exceptions. For example, the driver does not expand a horizontal TAB (011), and a RUBOUT (or DEL, 177) does not erase.

NOTE

If upper- to lowercase conversion is disabled, characters 175 and 176 do not act as terminators, and CTRL/O, CTRL/Q, and CTRL/S (017, 021, and 023, respectively) are not special terminators. The driver interprets them as output control characters in a normal manner.

Upon successful completion of an IO.RST request that was not terminated by filling the input buffer, the first word of the I/O status block contains the terminating character in the high byte and the IS.SUC status code in the low byte. The second word contains the number of bytes contained in a buffer. The driver does not put the terminating character in the buffer.

11.2.2.11 IO.RTT - This QIO function reads characters in a manner like the IO.RLB function, except a user-specified character terminates the read operation. The specified character's code can range from 0 through 377 octal. To specify the terminator, set the appropriate bit in a 16-word table that corresponds to the desired character. To specify multiple characters, set their corresponding bits.

The 16-word table starts at the address specified by the table parameter. The first word contains bits that represent the first 16 ASCII character codes (0 through 17 octal); similarly, the second word contains bits that represent the next 16 character codes (20-37 octal), and so forth, through the sixteenth word, bit 15, which represents character code 377 octal. For example, to specify the % symbol (code 045 octal) as a read terminator

character, set bit 05 in the third word, since the third word of the table contains bits representing character codes 40 through 57 octal.

If you desire CTRL/S (023), CTRL/Q (021), or any characters whose codes are greater than 177 octal as the terminator character(s), the terminal must be set to Read-Pass-All operation (TC.BIN=1), or to read-pass 8-bits (TC.8BC), as listed in Table 11-7.

You can include the optional timeout count parameter.

11.2.2.12 IO.WAL and IO.WLB!TF.WAL - IO.WAL is equivalent to IO.WLB logically ORed with the subfunction bit TF.WAL.

The Write All function causes the driver to pass all output from the buffer without interpretation. The driver does not intercept control characters, nor does it wrap long lines if you have selected input/output wrap-around.

11.2.2.13 IO.WBT - The IO.WBT function instructs the driver to write the buffer regardless of the I/O status of the receiving terminal. If you issue an IO.WBT function on a system that does not support IO.WBT, the driver treats it as an IO.WLB function.

Note the following:

- If another Write operation is currently in progress, the driver finishes the current request and issues IO.WBT as the next Write request. The effect of this is that a CTRL/S can stop IO.WBT functions. Therefore, it can be desirable for tasks to time out on IO.WBT operations.
- If a read is currently posted, the IO.WBT proceeds and the driver automatically performs a CTRL/R to redisplay any input that it received before the breakthrough Write occurred.
- The driver cancels CTRL/O if it is in effect.
- The driver rubs out any escape sequence that was interrupted.

An IO.WBT function cannot break through another IO.WBT that is in progress. Only a privileged task can issue a breakthrough Write.

11.2.2.14 IO.WSD - Use the Write Special Data function to communicate nontext information to the terminal task. The buffer address and length are the same as for IO.WLB. The data type parameter indicates to the terminal task the type of data that the buffer contains. The data type is:

SD.GDS PRO/GIDIS output

Note that this QIO implements a data path to the terminal subsystem that is also used by the CORE Graphics Library and by the VT125 (ReGIS) terminal emulator within the PRO/Communications application.

11.2.2.15 IO.RSD - The Read Special Data function is also used in communicating nontext information to the terminal subsystem. The buffer address, length, and timeout are the same as for IO.RLB. The data type parameter indicates to the terminal what type of data is to be read.

The following restrictions apply to the use of IO.RSD:

- In some ways, IO.RSD is the same as a normal Read operation. One result of this is that if there is a Read request currently outstanding to the keyboard, the IO.RSD does not take effect until the Read to the keyboard is complete.

- While an IO.RSD is pending, no input processing occurs until the IO.RSD request completes. Consequently, any characters that originate from the keyboard go directly to the type-ahead buffer. Also, no ASTs occur and no characters are echoed.

- When the driver receives special data from the terminal task (for example, a PRO/GIDIS report) and no IO.RSD request is outstanding, the special data enters a type-ahead buffer. The type-ahead buffer is capable of holding a maximum of 36 bytes. If more characters are input than the buffer can hold, the driver discards those characters, but does not return an error message.

  If there is an IO.RSD pending when the driver receives special data from the terminal task, the data directly enters a read buffer. However, the length of one report cannot exceed 36 bytes.

As a result of these restrictions, the recommended means of obtaining a special data report is to first issue the IO.WSD to cause the report to occur, and then to issue an IO.RSD to receive the exact length of the request. This causes the IO.RSD to

complete immediately, preventing it from blocking the keyboard input.

Note that this QIO implements a data path to the terminal subsystem that is also used by the CORE Graphics Library and by the VT125 (ReGIS) terminal emulator within the PRO/Communications application.

11.2.2.16 SF.GMC - The Get Multiple Characteristics function returns terminal characteristic information. This function is the complementary function for SF.SMC.

The format is:

```
QIO$C SF.GMC,...,<stadd,size>
```

stadd — The starting address of a buffer whose length in bytes is specified in the size parameter. Each buffer word has the form:

```
.BYTE  charname
.BYTE  0
```

charname — One of the characteristic bit names listed in Table 11-7 (except TC.CLC, which is valid only with the SF.SMC function). In the high byte of each byte pair, the driver returns 1 if the characteristic is true for the terminal, and 0 if it is not true.

11.2.2.17 SF.SMC - The Set Multiple Characteristics function enables a task to set and reset the characteristics of a terminal. This function is the complementary function for SF.GMC.

The format is:

```
QIO$C SF.SMC,...,<stadd,size>
```

stadd — The starting address of a buffer whose length in bytes is specified in the size parameter. Each word in the buffer has the form

```
.BYTE charname
.BYTE value
```

charname — One of the symbolic bit names listed in Table 11-7.

value — Either 0 (to clear a given characteristic) or 1 (to set a characteristic).

If charname is TC.TTP (terminal type), the parameter value can have any of the values listed in Table 11-9.

Specifying any value for TC.TBF flushes (clears) the type-ahead buffer; that is, it forces the type-ahead buffer count to 0.

Table 11-7: Driver-Terminal Characteristics, SF.GMC and SF.SMC

Symbol	Note	Octal Value	Meaning, If Asserted
TC.ACR	-	24	Wrap-around mode.
TC.ANI	-	122	ANSI CRT terminal.
TC.AVO	-	123	VT100-family terminal display with advanced video.
TC.BIN	-	65	Binary input mode (read-pass-all). No characters are interpreted as control characters.
TC.CLC	[2]	151	CTRL/C notification without terminal attached (SF.SMC only).
TC.CTS	[3]	72	Suspend output to terminal: 0 = resume 1 = suspend
TC.EDT	-	125	Terminal performs editing functions.
TC.EPA	[1]	40	Parity sense, valid only when TC.PAR is enabled: 0 = odd parity 1 = even parity
TC.ESQ	-	35	Input escape sequence recognition.
TC.FDX	-	64	Full-duplex mode.

Symbol	Note	Octal Value	Meaning, If Asserted
TC.HFF	-	17	Hardware form-feed capability (if 0, form-feeds are simulated using TC.LPP).
TC.HFL	-	13	Number of fill characters to insert after a RETURN (0 through 7 equals x).
TC.HHT	-	21	Horizontal tab capability (if 0, horizontal tabs are simulated using spaces).
TC.LPP	-	2	Page length (1 through 255 decimal equals x).
TC.NEC	-	47	Echo suppressed.
TC.PAR	[1]	37	Enable and disable parity: 1 = enabled 0 = disabled
TC.RGS	-	126	Terminal supports ReGIS instructions.
TC.RSP	[1]	3	Receiver speed, bits per second. See Table 11-10.
TC.SCP	-	12	Terminal is a scope (CRT).
TC.SXL	-	150	Printer supports sixel graphics.
TC.SFC	-	131	Terminal supports soft character set.
TC.SMR	-	25	Uppercase conversion disabled.
TC.TBF	[6]	71	Type-ahead buffer count (read), or flush (write).
TC.TTP	[4]	10	Terminal type (0 through 255 decimal) See Table 11-9.
TC.VFL	-	14	Send four fill characters after line feed.
TC.WID	[5]	1	Page width (1 through 255 decimal).

Symbol	Note	Octal Value	Meaning, If Asserted
TC.XSP	[1]	4	Transmitter speed, bits per second. See Table 11-10.
TC.8BC	-	67	Pass eight bits on input, even if not binary input mode (TC.BIN).

Notes to Table 11-7

1. The characteristic is device dependent. Both the printer port (TT2:) and quad SLU terminals have the characteristic.

2. **TC.CLC**

   The TC.CLC characteristic consists of two words instead of one word. The format is:

```
.BYTE  TC.CLC
.BYTE  3
.WORD  ast_add
```

   where

   ast_add is the address of a CTRL/C AST routine. If this value is zero, then the terminal driver simply disables the last CTRL/C AST request that was previously set by the task.

   You need not re-enable CTRL/C trapping after setting the TC.CLC characteristic; the system will recognize CTRL/C ASTs until you explicitly disable them, or when the task exits.

   Multiple tasks can handle CTRL/C ASTs. The terminal driver services these ASTs on a last in-first-out (LIFO) basis.

   Upon entry to the AST routine, the top word on the stack contains the value 3. Your task must remove the top word from the stack before exiting the AST. Table 11-8 illustrates the stack upon entry to the AST routine.

Table 11-8: Stack Upon Entry to AST Routine

Current Stack Pointer	Contents
SP+10	Event flag mask word
SP+06	PS of task prior to AST
SP+04	PC of task prior to AST
SP+02	Task's directive status word
SP+00	3

3. **TC.CTS**

The TC.CTS characteristic returns the present suspend (CTRL/S), resume (CTRL/Q), or suppress (CTRL/O) state set via the SF.SMC function. The driver returns the following values:

Value Returned	State
0	Resume (CTRL/Q)
1	Suspend (CTRL/S)
2	Suppress (CTRL/O)
3	Both suppress and suspend

Using a value of 0 with the SF.SMC function clears the suspend state; a value of 1 selects the suspend state.

NOTE

If you stop output to the terminal screen by pressing the HOLD SCREEN key on a Professional, TC.CTS does not indicate that output has stopped. Also, if you stop output to the terminal screen by pressing NO SCROLL on a VT100-series or HOLD SCREEN on a Professional, you cannot resume output with TC.CTS.

4. **TC.TTP**

For the TC.TTP characteristic (terminal type), the driver returns a value listed in Table 11-9 in the high byte.

The TC.TTP characteristic, when read by the terminal driver, sets implicit values for several of the other terminal characteristics. Table 11-9 shows the characteristics that the driver sets. You can change implicit values by issuing subsequent Set Multiple Characteristics requests. (Characteristics not listed in the table are not implicitly set by the driver.)

The terminal driver also uses TC.TTP to determine cursor positioning commands, as appropriate.

5. **TC.WID**

Unsolicited input that fills the buffer before the driver receives a terminator is probably invalid. When this happens, the driver discards the input by simulating a CTRL/U and echoing ^U.

6. **TC.TBF**

The TC.TBF characteristic returns the number of unprocessed characters in the type-ahead buffer for the specified terminal. This allows tasks to determine if any characters were typed that did not require AST processing. In addition, the value returned can be used to read the exact number of characters typed, rather than a typical value of 80 (decimal) or 132 (decimal) characters for the terminal.

Note the following:

- The capacity of the type-ahead buffer is 36 (decimal) characters.
- Using TC.TBF in an SF.SMC function flushes the type-ahead buffer.

Table 11-9: Terminal Type Values (TC.TTP) for SF.SMC and SF.GMC

Symbol/Terminal Type	Implicit Characteristics*														
	TC. ANI	TC. AVO	TC. CUP	TC. DEC	TC. EDT	TC. HFF	TC. HFL	TC. HHT	TC. LPP	TC. RGS	TC. SCP	TC. SFC	TC. SXL	TC. VFL	TC. WID
T.BMP1/PC3xx**	1	1	3	1	1	1	-	1	24	-	1	1	-	-	80
T.LA12/LA12	-	-	-	-	-	1	-	1	66	-	-	-	-	-	132
T.LA50/LA50	-	-	-	-	-	1	-	1	66	-	-	-	1	-	80
T.L100/LA100	-	-	-	-	-	1	-	1	66	-	-	-	1	-	132
T.L120/LA120	-	-	-	-	-	1	-	-	66	-	-	-	-	-	132
T.L210/LA210	-	-	-	-	-	1	-	1	66	-	-	-	1	-	132
T.LN03/LN03	-	-	-	-	-	1	-	1	66	-	-	-	1	-	132
T.LQP2/LPQ02	-	-	-	-	-	1	-	1	66	-	-	-	-	-	132
T.LQP3/LQP03	-	-	-	-	-	1	-	1	66	-	-	-	-	-	132
T.UNK0/Unknown	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
T.V100/VT100	1	1	3	1	-	-	-	1	24	-	1	-	-	-	80
T.V101/VT101	1	-	3	1	-	-	-	1	24	-	1	-	-	-	80
T.V102/VT102	1	1	3	1	1	-	-	1	24	-	1	-	-	-	80
T.V105/VT105	1	-	3	1	-	-	-	1	24	-	1	-	-	-	80
T.V125/VT125	1	1	3	1	-	-	-	1	24	1	1	-	-	-	80
T.V2XX/VT2xx	1	1	3	1	1	1	-	1	24	-	1	1	-	-	80

NOTES

* Implicit characteristics are shown as supported by the driver. Values not shown are not automatically set by the driver. An unknown terminal type has no implicit characteristics.

** The PC3xx Series Bitmap Display is the default terminal for the Professional. Note that the only difference between T.BMP1 and T.V2XX is the Terminal Subsystem's response to a device attributes request. See the Terminal Subsystem Manual for details.

Table 11-10 lists the values for terminal receiver and transmitter speeds for TC.ASP, TC.RSP, and TC.XSP.

Table 11-10: Receiver/Transmitter Speed Values

TC.ASP, TC.RSP, TC.XSP Symbol	Decimal Value	Actual Baud Rate (Bits per Second)	Available for Printer Port?	Available for Quad SLU?
S.50	2	50	yes	no
S.75	3	75	yes	yes
S.110	5	110	yes	yes
S.134	6	134	yes	yes
S.150	7	150	yes	yes
S.300	9	300	yes	yes
S.600	10	600	yes	yes
S.1200	11	1200	yes	yes
S.1800	12	1800	yes	no
S.2000	13	2000	yes	yes
S.2400	14	2400	yes	yes
S.3600	15	3600	yes	no
S.4800	16	4800	yes	yes
S.7200	17	7200	yes	yes
S.9600	18	9600	yes	yes
S.19.2	21	19200	yes	yes

11.3 STATUS RETURNS FOR TERMINAL DRIVER

Table 11-11 describes error and status conditions that the terminal driver returns in the I/O status block.

Most error and status codes returned are byte values. For example, the value for IS.SUC is 1. However, IS.CC, IS.CR, IS.ESC, and IS.ESQ are word values. When the driver returns any of these codes, the low byte indicates successful completion, and the high byte shows what type of completion occurred.

To test for one of these word-value return codes, first test the low byte of the first word of the I/O status block for the value IS.SUC. Then, test the full word for IS.CC, IS.CR, IS.ESC, or IS.ESQ. (If the full word tests equal to IS.SUC, then its high byte is 0, indicating byte-count termination of the read.)

You can consider the return IE.EOF to be a successful read, since characters returned to the task's buffer can be terminated by a CTRL/Z character.

The driver returns the SE.xxx codes after a request to either the SF.GMC or SF.SMC functions, described in Sections 11.2.2.16 and 11.2.2.17. When the driver returns any of these codes, the low byte in the first word of the I/O status block contains IE.ABO. The second word of the IOSB contains an offset (starting from 0) to the byte in error in the QIO's stadd buffer.

Table 11-11: Terminal Status Returns

Symbol	Description
IE.EOF	**Successful completion on a read with end-of-file.** The line of input read from the terminal terminated with the end-of-file character CTRL/Z. The second word of the I/O status block contains the number of bytes read before the driver received the CTRL/Z. The input buffer contains those bytes.
IS.SUC	**Successful completion.** The operation specified in the QIO directive completed successfully. If the operation involved reading or writing, you can examine the second word of the I/O status block to determine the number of bytes processed. The input buffer contains those bytes.

STATUS RETURNS FOR TERMINAL DRIVER

Symbol	Description
IS.CC	**Successful completion on a read.** The line of input read from the terminal terminated with an INTERRUPT/DO or CTRL/C sequence. The input buffer contains the bytes read.
IS.CR	**Successful completion on a read.** The line of input read from the terminal terminated with a RETURN. The input buffer contains the bytes read.
IS.ESC	**Successful completion on a read.** The line of input read from the terminal terminated with an escape character (CSI or SS3). The input buffer contains the bytes read.
IS.ESQ	**Successful completion on a read.** The line of input read from the terminal terminated with an escape sequence. The input buffer contains the bytes read and the escape sequence.
IS.PND	**I/O request pending.** The operation specified in the QIO directive has not yet been executed. The I/O status block is filled with zeroes.
IS.TMO	**Successful read with time-out.** The line of input read from the terminal was terminated by a time-out. (TF.TMO was set and the specified time interval was exceeded.) The input buffer contains the bytes read.
IE.ABO	**Operation aborted.** IO.KIL canceled the specified I/O operation while in progress or while in the I/O queue. The second word of the I/O status block indicates the number of bytes the driver read or wrote before the kill occurred.
IE.BAD	**Bad parameter.** The size of the data buffer in the QIO request exceeds 8128 bytes.
IE.DAA	**Device already attached.** The issuing task already attached the physical device unit specified in an IO.ATT request. This code does not indicate that the unit was attached by another task. The subfunction bits TF.AST or TF.ESQ specified in the attach have no effect.
IE.DNA	**Device not attached.** The issuing task did not attach the physical device unit specified in an IO.DET function. This code has no bearing on the attachment status of other tasks.

STATUS RETURNS FOR TERMINAL DRIVER

Symbol	Description
IE.DNR	**Device not ready.** The physical device unit specified in the QIO directive was not ready to perform the desired I/O operation. The driver returns this code to indicate that a time-out occurred on the physical device unit (that is, an interrupt was lost).
IE.IES	**Invalid escape sequence.** An escape sequence was started, but escape-sequence syntax was violated before the sequence was completed. The character causing the violation is the last character in the buffer. For details, see Section 11.5.4.
IE.IFC	**Invalid function.** A function code specified in an I/O request is invalid for terminals.
IE.NOD	**Buffer allocation failure.** System dynamic storage has been depleted, resulting in insufficient space available to allocate an intermediate buffer for an input request or an AST block for an attach request.
IE.PES	**Partial escape sequence.** An escape sequence was started, but read-buffer space was exhausted before the sequence was completed. The buffer specified for a Read or Write request was partially or totally outside the address space of the issuing task, a byte count of 0 was specified, or an odd or 0 AST address was specified. For details, see Section 11.5.4.3.
IE.PRI	**Privilege violation.** A nonprivileged task issued an IO.WBT, directed an SF.SMC to a terminal other than TI:, or attempted to set its privilege bit.
SE.NIH	**Not in handler.** An SF.GMC or SF.SMC request named an invalid terminal characteristic, or a task attempted to assert TC.PRI. For a list of valid terminal characteristics, see Table 11-7.
SE.FIX	**Fixed Characteristic.** An attempt was made to change a fixed characteristic in a SF.SMC subfunction request (for example, an attempt was made to change the unit number).

Symbol	Description
SE.VAL	**Invalid value.** The new value specified in an SF.SMC request for the TC.TTP terminal characteristic is invalid. For a list of the valid values, see Table 11-9.

11.4 CONTROL CHARACTERS AND SPECIAL KEYS

This section describes the system's special terminal control characters and keys. Note that the driver does not recognize control characters and special keys during a Read All request (IO.RAL), and recognizes only some of them during a Read with Special Terminators (IO.RST).

11.4.1 Control Characters

A control character is input from a terminal by holding the control key (CTRL) down while typing one other key. The driver echoes three of the control characters described in Table 11-12, CTRL/R, CTRL/U, and CTRL/Z, as ^R, ^U, and ^Z, respectively.

Table 11-12: Terminal Control Characters

Character	Meaning
CTRL/O	CTRL/O suppresses terminal output. For attached terminals, CTRL/O remains in effect (output is suppressed) until one of the following occurs: • The terminal is detached. • Another CTRL/O character is typed. • An IO.CCO or IO.WBT function is issued. • Input is entered. For unattached terminals, CTRL/O suppresses output for only the current output buffer (typically one line).

Character	Meaning
CTRL/Q	CTRL/Q resumes terminal output previously suspended by means of either the HOLD SCREEN key or CTRL/S.
CTRL/R	Typing CTRL/R results in a RETURN and line feed being echoed, followed by the incomplete (unprocessed) input line. Any tabs that were input are expanded and the effect of any rubouts is shown. On hardcopy terminals, CTRL/R verifies the effect of tabs or rubouts in an input line. CTRL/R is also useful for CRT terminals for the CRT rubout. For example, after rubbing out the leftmost character on the second displayed line of a wrapped input line, the cursor does not move to the right of the first displayed line. In this case, CTRL/R brings the input line and the cursor together again.
CTRL/S	CTRL/S suspends terminal output. (Output can be resumed by pressing the HOLD SCREEN key or typing CTRL/Q. See the description of CTRL/Q.)
CTRL/U	Typing CTRL/U before typing a line terminator deletes previously-typed characters back to the beginning of the line. The system echoes this character as ^U followed by a RETURN and a line feed.
CTRL/X	CTRL/X clears the type-ahead buffer.
CTRL/Z	CTRL/Z indicates an end-of-file for the current terminal input.

11.4.2 INTERRUPT/DO AST Information

Table 11-13 describes the driver's response when your application requests the IO.ATA QIO without specifying the TF.XCC subfunction.

NOTE

The driver does not recognize INTERRUPT/DO from terminals attached through the Quad SLU or printer port. It does recognize CTRL/C, however.

Table 11-13: Response with IO.ATA (Omitting TF.XCC)

Key One	Key Two	Driver Response
Other than INTERRUPT	DO	The driver handles the first key as usual and sends your task the escape sequence for the DO key.
INTERRUPT	Other than DO	The driver discards the INTERRUPT key and handles the second key as if INTERRUPT had not been pressed.
INTERRUPT	DO	The driver activates your task's AST routine just as if a CTRL/C had been typed.
CTRL/C	-	The driver activates your task's AST routine as for RSX-11M-PLUS.

Table 11-14 describes the driver's response when your application does not request IO.ATA, or if it has requested IO.ATA!TF.XCC.

Table 11-14: Response Without IO.ATA or with IO.ATA!TF.XCC

Key One	Key Two	Driver Response
Other than INTERRUPT	DO	The driver handles the first key as usual and sends your task the escape sequence for the DO key.
INTERRUPT	Other than DO	The drver discards the INTERRUPT key and handles the second key as if INTERRUPT had not been pressed.

Key One	Key Two	Driver Response
INTERRUPT	DO	The driver notifies the PRO/Dispatcher, which aborts all application tasks. Your application receives no indication that anything happened.
CTRL/C	-	The driver notifies the PRO/Dispatcher, which aborts aborts all application tasks. Your application receives no indication that anything happened.

If your application puts the terminal in Read Pass All mode or if it specifies TF.RAL on a read, all keys, except for HOLD SCREEN and PRINT SCREEN, enter into the type-ahead buffer unprocessed. The INTERRUPT and DO keys enter as the escape sequences that they represent.

The driver places any characters for which there is no read or AST request outstanding into the 36-byte type-ahead buffer. If the buffer is full, and if a character is typed that would go into the type-ahead buffer, the driver echoes a bell and discards the additional character. When either CTRL/C or the INTERRUPT/DO sequence is entered, the driver flushes the type-ahead buffer.

11.4.3 Special Keys

The RETURN, and DELETE keys have special significance for terminal input, as described in Table 11-15.

A line can be terminated by a RETURN or CTRL/Z characters, or by completely filling the input buffer. You can determine the standard buffer size for a terminal by issuing the Get LUN Information system directive and examining word 5 of the buffer.

Table 11-15: Special Terminal Keys

Key	Meaning
RETURN	Pressing RETURN terminates the current line and causes the carriage or cursor to return to the first column on the line.
DELETE	Pressing DELETE deletes the last character typed on an input line. The operator can delete only characters typed since the last line terminator and can delete several characters in sequence by pressing successive DELETEs. DELETE causes the driver to remove the last typed character (if any) from the incomplete input line and to echo the following character sequence for that terminal: backspace-space-backspace If the last typed character was a tab, the driver issues enough backspaces to move the cursor to the character position before the tab. If a long input line was split or wrapped by the automatic-return option, and a DELETE erases the last character of a previous line, the driver does not move the cursor to the previous line.

11.5 ESCAPE SEQUENCES

Escape sequences are strings of two or more characters beginning with an ESC (033 octal) character, or--in 8-bit environments only--with either the CSI (233 octal) or SS3 (217 octal) characters.

Escape sequences provide a way to pass input to a task without interpretation by the operating system. You can do this with a number of one-character Read All function requests, but escape sequences allow you to use IO.RLB requests.

11.5.1 Format of Escape Sequences

The format of an escape sequence that begins with the ESC character is:

ESC [i]... t...

ESC	The escape character, 33 octal.
[i]...	Optional intermediate character combinations, which can consist only of the characters 40 (octal) to 57 (octal), inclusive.
t...	Terminating character combinations, which can consist only of the characters 60 (octal) to 176 (octal) inclusive.

The format of an escape sequence that begins with the CSI character is:

CSI [i]... t...

CSI	The 8-bit control sequence introducer character, 233 octal, or the seven-bit combination ESC [(33 + 133 octal).
[i]...	Optional intermediate character combinations, which can consist only of the characters 40 (octal) to 77 (octal), inclusive.
t...	Terminating character combinations, which can consist only of the characters 100 (octal) to 176 (octal) inclusive.

The format of an escape sequence that begins with the SS3 character is:

SS3 i

SS3	The 8-bit single shift character, 217 octal, or the 7-bit combination ESC O (33 + 117 octal).
i	A single character in the range 40 (octal) to 176 (octal), inclusive.

11.5.2 Receiving Escape Sequences

Two prerequisites must be satisfied before your task can receive escape sequences:

- The task must request the escape sequences by issuing an IO.ATT function and invoking the subfunction bit TF.ESQ.
- You must have made the terminal capable of generating escape sequences. Do this by issuing the Set Multiple Characteristics request.

If you have not satisfied these prerequisites, the driver treats the ESC character as a line terminator.

11.5.3 Characteristics of Escape Sequences

Escape sequences always act as line terminators. That is, an input buffer can contain other characters that are not part of an escape sequence, but the last characters in the buffer are always part of an escape sequence.

The driver does not echo escape sequences. However, if a DELETE sequence (not from the terminal) is in progress, the driver closes the sequence with a backslash when an escape sequence is begun.

The driver does not recognize escape sequences in unsolicited input streams. Also, the driver does not recognize them in a Read with Special Terminators (subfunction bit TF.RST) or in a Read All (subfunction bit TF.RAL).

11.5.4 Escape Sequence Format Violations

A violation of the format defined in Section 11.5.1 causes the terminal driver to abandon the sequence and to return the error IE.IES.

11.5.4.1 Delete Character--DEL (177) - The DELETE character is not valid within an escape sequence. If it occurs at any point within an escape sequence, the driver abandons the entire sequence and deletes it from the input buffer.

11.5.4.2 Control Characters - The reception of any character in the range 0 through 37 or 200 through 237 (with four exceptions) is a syntax violation that terminates the read with an error (IE.IES). The four exceptions are the control characters CTRL/Q, CTRL/S, CTRL/X, and CTRL/O. The Executive handles these characters normally even when an escape sequence is in progress.

11.5.4.3 Full Buffer - A syntax error results when an escape sequence is terminated by running out of read buffer space, rather than by receipt of a final character. The driver returns the error IE.PES.

For example, suppose a task issues an IO.RLB request with a buffer length of 2, and the terminal operator enters the following characters:

```
ESC ! A
```

The buffer contains "ESC !", word 1 of the I/O Status Block contains IE.PES, and word 2 of the IOSB contains the value 2.

The driver treats the A as unsolicited input.

11.6 VERTICAL FORMAT CONTROL

Table 11-16 summarizes all characters used for vertical format control on the terminal. You can specify any one of these characters as the value of the vfc parameter in the IO.WLB, IO.WVB, IO.WBT, IO.CCO, or IO.RPR functions.

Table 11-16: Vertical Format Control Characters

Octal Value	Character	Meaning
040	blank	**Single space.** Write one line feed, print the contents of the buffer, and write a RETURN. Normally, printing immediately follows the last line printed.
060	0	**Double space.** Write two line feeds, print the contents of the buffer, and write a RETURN. Normally, the driver prints the buffer contents two lines below the last line printed.
061	1	**Page eject.** If the terminal supports FORM FEEDs, write a form feed, print the contents of the buffer, and write a RETURN. If the terminal does not support FORM FEEDs, the driver simulates the FORM FEED character either by writing four line feeds to a CRT terminal, or by writing enough line feeds to advance the paper to the top of the next page on a printing terminal.
053	+	**Overprint.** Write the contents of the buffer and write a RETURN, normally overprinting the previous line.
044	$	**Prompting output.** Write one line feed and print the contents of the buffer. This mode of output is intended for use with a terminal to which a prompting message is written and input is read on the same line.
000	null	**Internal Vertical Format.** Write the buffer contents without adding vertical format control characters.

The driver interprets all other vertical format control characters as blanks (040). Your task can determine the buffer width by issuing a Get LUN Information directive and examining word 5 returned in the buffer.

It is possible to lose track of where you are in the input buffer if wrap-around is enabled for the terminal. For example, while deleting text on a wrapped line, the cursor does not back up to the previous line.

11.7 TYPE-AHEAD BUFFERING

The terminal driver either immediately processes characters that it receives, or it stores the characters in the type-ahead buffer. The type-ahead buffer allows characters to be temporarily stored and retrieved in a first-in, first-out (FIFO) manner.

The driver uses the type-ahead buffer as follows:

Store in buffer:

The driver stores an input character in the type-ahead buffer if one or more of the following conditions is true:

- There is at least one character presently in the type-ahead buffer.
- The character input requires echo, and the output line to the terminal is presently busy writing a character.
- No Read operation is in progress, no unsolicited input AST is specified, and the terminal is attached. The driver does not echo a character when storing it in the buffer. Instead, the driver defers echoing a character until retrieving it from the buffer, since the driver is unaware of the read mode (for example, read-without-echo) until that time.

NOTE

Depending on the terminal mode and the presence of a Read function, Read subfunctions, and an unsolicited input AST, the driver might process CTRL/C, INTERRUPT/DO, CTRL/O, CTRL/Q, CTRL/S, and CTRL/X characters immediately, without storing them in the type-ahead buffer.

Retrieve from buffer:

When the driver becomes ready to process input, or when a task issues a read request, the driver attempts to retrieve a character from the buffer. If this attempt is successful, the driver processes and echoes the character if required.

The driver then continues to retrieve and process characters until one of the following conditions is true:

- The buffer is empty.
- The driver becomes unable to process another character.
- A read request terminates, leaving the terminal either attached or slaved.

Flush the buffer:

The driver flushes (clears) the buffer upon receiving CTRL/X OR INTERRUPT/DO.

An exception is that CTRL/X does not flush the buffer if read-pass-all or read-with-special-terminators is in effect.

If the buffer becomes full, each character that cannot be entered causes a BELL character to be echoed to the terminal.

If a character is input and echo is required, but the transmitter section is busy with an output request, the driver holds the input character in the type-ahead buffer until output (transmitter) completion occurs.

11.8 FULL-DUPLEX OPERATION

While in full-duplex mode, the driver attempts to simultaneously service one Read request and one Write request. The driver performs Attach, Detach and Set Multiple Characteristics functions with the line in an idle state (not executing a Read or a Write request).

11.9 INTERMEDIATE INPUT AND OUTPUT BUFFERING

Input buffering for checkpointable tasks exists in the terminal driver private pool. As each buffer becomes full, the driver automatically allocates a new buffer and links it to the previous

buffer. The Executive then transfers characters from these buffers to the task buffer. The driver deallocates the buffers once the transfer has been completed.

If the driver fails to allocate the first input buffer, it transfers the characters directly into the task buffer. If the driver successfully allocates the first buffer, but a subsequent buffer allocation fails, the input request terminates with the error code IE.NOD. In this case, the I/O Status Block contains the number of characters actually transferred to the task buffer.

Your task can then update the buffer pointer and byte count and reissue a Read request to receive the rest of the data. The type-ahead buffer ensures that no input data is lost as long as the type-ahead buffer is not full.

All terminal output is buffered. The driver allocates and links to a list as many buffers as required. If the driver cannot obtain enough buffers for all output data, it performs the transfer as a number of partial transfers, using available buffers for each partial transfer. This is transparent to the requesting task. If the driver cannot allocate any buffers, the request terminates with the error code IE.NOD.

The unconditional output buffering serves the following purposes:

- It reduces time spent at system state.
- It enables task checkpointing during the transfer to the terminal (if all output fits in one buffer list).

11.10 TERMINAL-INDEPENDENT CURSOR CONTROL

The terminal driver responds to task I/O requests for cursor positioning without the task requiring information about the type of terminal in use.

You specify cursor position in the vfc parameter of the IO.WLB or IO.RPR function. To use the vfc parameter to define cursor position, the high byte must be nonzero. The driver in this case interprets the low byte as column number (x coordinate) and the high byte as line number (y coordinate).

If the high byte of the vfc parameter is 0, the driver intreprets the parameter simply as a vertical format control (vfc) character.

The driver defines the home position, the upper-left corner of the display, as 1,1. Depending upon the terminal type, the driver writes cursor-positioning commands appropriate for the terminal in use to move the cursor to the specified position. If the most significant bit of the line number is set, the driver clears the display before positioning the cursor.

When defining cursor position in an IO.WLB function, you can use the TF.RCU subfunction to save the current cursor position. When included in this manner, TF.RCU causes the driver to:

1. Save the current cursor position.

2. Position the cursor and write the specified buffer.

3. Restore the cursor to the original (saved) position once the output transfer has been completed.

11.11 PROGRAMMING SUGGESTIONS

The following suggestion can help you code particular operations for the terminal driver.

11.11.1 Using IO.WVB Instead of IO.WLB

Use IO.WVB instead of IO.WLB when writing to a terminal. If the write actually goes to a terminal, the Executive converts the IO.WVB request to IO.WLB. However, if the LUN has been redirected to an appropriate device--a disk, for example--the driver will reject the IO.WVB function because a file is not open on the LUN. This prevents privileged tasks from overwriting block zero of the disk.

Note that the Executive strips any subfunction bits specified in an IO.WVB request (such as TF.CCO, TF.WAL, TF.WBT) when converting the IO.WVB to IO.WLB.

CHAPTER 12

VIRTUAL TERMINAL DRIVER

Virtual terminals are software constructs that can provide noninteractive terminal I/O support for offspring tasks.

Offspring tasks spawned by or connected to the parent task that created a virtual terminal can perform terminal I/O operations with the virtual terminal in the same manner as with physical terminals. Virtual terminals differ from physical terminals in that read and write operations directed to TI: are mediated by the parent task instead of a physical device.

The Executive creates a virtual terminal when requested by parent tasks with the Create Virtual Terminal (CRVT$) directive. Virtual terminals are eliminated either when the parent issues the ELVT$ directive or when the parent exits. All offspring tasks whose TI: point to the virtual terminal being eliminated are aborted by the Executive. See Sections 8.15 and 8.23 for details on CRVT$ and ELVT$.

Your task must establish user context by calling the login task for each virtual terminal it creates. Section 12.5 describes how to call the login task.

A special case of the virtual terminal is the *null* virtual terminal, described in Section 12.6.

12.1 GET LUN INFORMATION MACRO FOR VIRTUAL TERMINAL DRIVER

Word 2 of the buffer filled by the Get LUN Information system directive (the first characteristics word) contains the following information for virtual terminals. A setting of 1 indicates that the described characteristic is true for virtual terminals.

Table 12-1: Get LUN Information for Virtual Terminal Driver

Bit	Setting	Meaning
0	1	Record-oriented device
1	1	Carriage-control device
2	1	Terminal device
3	0	File-structured device
4	0	Single-directory device
5	0	Sequential device
6	0	Reserved
7	0	User-mode diagnostics supported
8	0	Reserved
9	0	Unit software write-locked
10	0	Input spooled device
11	0	Output spooled device
12	0	Pseudo device
13	0	Device mountable as a communications channel
14	0	Device mountable as a FILES-11 volume
15	0	Device mountable

Words 3 and 4 are undefined. Word 5 specifies the maximum byte count (that is, maximum buffer size) to which offspring requests will be truncated; this value is specified by the parent task in the Create Virtual Terminal system directive, as described in Section 8.15.

12.2 QIO MACRO FOR VIRTUAL TERMINAL DRIVER

Table 12-2 lists the standard and device-specific functions of the QIO macro that are valid for virtual terminals.

NOTE

The length of a buffer in a QIO request to a virtual terminal cannot exceed 20000-100 (octal), or 8128 (decimal) bytes.

Table 12-2: QIO Functions for Virtual Terminals

Format	Function
Standard Functions	
QIO$C IO.ATT,...	Attach device
QIO$C IO.DET,...	Detach device
QIO$C IO.KIL,...	Cancel I/O request
QIO$C IO.RLB,...,<stadd,size>	Read logical block
QIO$C IO.RVB,...,<stadd,size>	Read virtual block (effects IO.RLB)
QIO$C IO.WLB,...,<stadd,size,stat>	Write logical block
QIO$C IO.WVB,...,<stadd,size,stat>	Write virtual block (effects IO.WLB)
Device-Specific Functions	
QIO$C IO.STC,...,<cb,sw2,sw1>	Set terminal characteristics (enable/disable intermediate I/O buffering), or return I/O completion status to offspring task
QIO$C SF.GMC,...,<stadd,size>	Get multiple characteristics

Format	Function
QIO$C IO.GTS,...,<stadd,size>	Get terminal support
QIO$C IO.RPR,...,<stadd,size, [tmo],pradd,prsize,vfc>	Read logical block after prompt
QIO$C SF.SMC,...,<stadd,size>	Set multiple characteristics

Parameters Shown in Table 12-2

size

The size of the data buffer in bytes (must be greater than 0, and less than or equal to 17700 octal). The buffer must be located within the address space of the parent or offspring task issuing the I/O request.

stadd

The starting address of the data buffer. The address must be word aligned for SF.GMC, IO.GTS, and SF.SMC; otherwise, it may be aligned on a byte boundary.

stat

The I/O completion successful status code, specified by the parent task, issued by the virtual terminal driver in response to an offspring task's read request.

cb

Characteristic bits to be set, selecting the following virtual terminal functions:

cb Value	Bits Set	Function
0	none	Enable intermediate buffering in Executive pool
1	0	Return specified virtual terminal I/O completion status to requesting offspring task
2	1	Disable intermediate buffering
3	0 and 1	Return status for offspring write request

sw1

The code for I/O completion status.

NOTE

The sw2 and sw1 parameters are valid in the IO.STC function only when cb=1 or cb=3.

tmo

An optional timeout count.

vfc

A character for vertical format control. See Table 11-16. This argument is supplied for compatibility with QIOs to physical terminals.

pradd

The starting address of the prompt buffer.

prsize

The size of the prompt buffer in bytes. The buffer must be located within the address space of the offspring task issuing the I/O request.

12.2.1 Standard QIO Functions

12.2.1.1 IO.ATT - This I/O function can be issued by offspring tasks to attach the virtual terminal. (It is invalid for parent tasks to issue IO.ATT.) Attaching a virtual terminal prevents other offspring tasks from executing I/O operations with the virtual terminal. However, parent task I/O requests are always serviced when issued.

This function is invalid for null virtual terminals.

12.2.1.2 IO.DET - This I/O function can be issued by offspring tasks to detach the virtual terminal, making it available for use by other offspring tasks connected to the same parent task. (It is invalid for parent tasks to issue IO.DET.)

This function is invalid for null virtual terminals.

12.2.1.3 IO.KIL - Parent and offspring tasks can issue IO.KIL to cancel I/O requests. An offspring task issuing IO.KIL can result in IE.ABO being returned to the parent task.

This function is invalid for null virtual terminals.

12.2.1.4 IO.RLB, IO.RVB, IO.WLB, IO.WVB - These read and write functions execute requested I/O operations with virtual terminals in the same manner as with terminals described in Chapter 2, except as follows:

- The virtual terminal driver returns the tmo parameter of an offspring task's IO.RLB or IO.RVB request, or the vfc parameter of an offspring task's IO.WLB or IO.WVB request as a stack parameter on entry to the appropriate AST for the parent task.

- The virtual terminal driver returns I/O completion status to the offspring task in response to successful completion of the offspring task's IO.RLB or IO.RVB request. However, the actual I/O completion status values returned are specified for data transfers in the third parameter word of the parent task's IO.WLB or IO.WVB response, or in the second and third parameters of the parent task's IO.STC function response when no data transfer is desired.

12.2.2 Device-Specific QIO Functions

12.2.2.1 IO.STC - The IO.STC function can be issued by parent tasks to enable/disable offspring task I/O buffering in secondary pool, or to force an appropriate I/O completion status for an offspring task read I/O request when no data transfer is desired. Both of these applications for the IO.STC function are described as follows.

Parent tasks can use IO.STC to enable (or disable) intermediate buffering in secondary pool. Intermediate buffering, when enabled, is performed on offspring task virtual terminal read and write requests when the offspring task is checkpointable.

Thus, offspring tasks can be stopped for virtual terminal I/O and checkpointed in a manner similar to that when you use physical terminals. Whenever the virtual terminal driver determines that

it should not use intermediate buffering, offspring tasks that issue terminal requests become locked in memory until I/O completion; transfers occur directly between parent task and offspring task buffers without intermediate buffering in secondary pool.

In addition to the conditions that permit intermediate buffering (when specified), one condition can disable intermediate buffering of the parent task. If the buffer size specified in the Create Virtual Terminal directive exceeds the system maximum size of 512 (decimal) bytes, then intermediate buffering is disabled.

The second application for IO.STC is to allow the virtual terminal driver to return an appropriate I/O completion status in response to an offspring task read request. I/O status returned in this manner allows successful completion of the offspring task's request when the parent task determines that no data transfer is desired; this condition can occur, for example, when no data is available for input to the offspring task by the virtual terminal driver. When you use the IO.STC function in this manner, you must include the three parameters, <cb,sw2,sw1>, as follows:

cb

A value of 1 is specified to indicate that the I/O completion status return to the offspring task is desired.

NOTE

If the virtual terminal is operating in full duplex mode, a cb value of 1 returns status for an offspring read request, and a cb value of 3 returns status for an offspring write request.

sw2

This parameter is the second word returned in the I/O completion status indicating the number of bytes read upon successful completion of an offspring task's read request. However, because no data transfer actually occurs, the value specified is 0; the byte count of 0 specified in this function is legal (and desired), whereas a byte count of 0 in write operations is invalid (and results in an error being returned to the parent task).

sw2

This parameter specifies the status code to be returned to the offspring task by the virtual terminal driver in the first word of the I/O completion status. This value is returned in the high byte and a value of +1 is returned in the low byte of the status word. Typical values, and the status that each represents, are:

Code	Value	Completion Status Indicated
IS.SUC	+ 1	Successful completion
IS.CR	15	Read terminated by a carriage return
IS.ESC	33	Read terminated by an escape character
IS.ESQ	233	Read terminated by an escape sequence

12.2.2.2 SF.GMC - The Get Multiple Characteristics function returns information on terminal characteristics. This function can be issued by both the parent and the offspring tasks. The virtual terminal driver returns the characteristics that were set by the previous corresponding SF.SMC request. However, only the full duplex mode (TC.FDX) characteristic affects the operation of the virtual terminal driver. The SF.GMC function is provided only to maintain transparency to the offspring task. Valid virtual terminal characteristics are listed in Table 12-3.

This function is invalid for null virtual terminals.

12.2.2.3 IO.GTS - The Get Terminal Support function returns a four-word buffer of information specifying which features are a part of the virtual terminal driver. The virtual terminal driver provides the IO.GTS function only to maintain transparency to the offspring task.

Table 11-6 lists the options returned by the full duplex terminal driver. Of those listed, the virtual terminal driver returns:

Word 1 -- F1.BUF, F1.RPR, F1.UTB, and F1.VBF

Word 2 -- F2.SCH and F2.GCH

This function is invalid for null virtual terminals.

12.2.2.4 IO.RPR - The Read After Prompt (IO.RPR) function can be issued only by the offspring task. When the offspring task issues this function, the function appears to the parent task as a separate write request followed by a read request.

See Section 11.2.2.8 for details.

12.2.2.5 SF.SMC - The SF.SMC function allows a task to set and reset the characteristics of a terminal. Both the parent and the offspring tasks may issue this function. The parent task may set virtual terminals to full duplex operation by using the SF.SMC function with the characteristics bit TC.FDX.

When in full duplex mode, the virtual terminal driver attempts to process the offspring task's read and write requests simultaneously. To ensure that these operations are overlapped, the parent task should minimize the amount of time it spends in AST state.

The virtual terminal driver defaults to half duplex mode.

This function is invalid for null virtual terminals.

Table 12-3 lists the characteristics that either the parent or the offspring task may set. Note that the default value for all the virtual terminal characteristics is 0.

Table 12-3: Virtual Terminal Characteristics

Symbolic	Octal Value	Meaning, If Asserted
TC.FDX	64	Full duplex mode
TC.SCP	12	Terminal is a scope
TC.SMR	25	Uppercase conversion disabled
TC.TTP	10	Terminal type

12.3 STATUS RETURNS FOR VIRTUAL TERMINAL DRIVER

The error and status conditions listed in Tables 12-4 and 12-5 are returned by the virtual terminal driver described in this chapter. The SE.NIH error is returned by the SF.GMC and SF.SMC functions. For this error, the low byte of the first word in the I/O status block contains IE.ABO. The second word in the I/O status block contains an offset (starting at 0) pointing to the erroneous byte in the stadd buffer.

Table 12-4: Virtual Terminal Status, Offspring Task Requests

Symbol	Description
--	Successful completion of an offspring task read request results in an I/O completion status specified in a parent task QIO parameter being returned. Typically, the status information returned simulates a subset of I/O returns normally produced by the terminal driver.
IS.SUC	**Successful completion.** The operation specified in the QIO directive was completed successfully. The second word of the I/O status block indicates the number of bytes transferred on a write operation.
IE.IFC	**Invalid function code.** The offspring task attempted a read or a write function and the parent task did not specify an AST address in its response to the requested I/O function, or the offspring task issued an IO.STC or other invalid function.
IE.ABO	**Request terminated.** The offspring task issued IO.KIL or the parent task eliminated the virtual terminal unit.
IE.SPC	**Invalid address space.** Part or all of the buffer specified for a read or write request was outside of the task's address space, or a byte count greater than 17700 (octal), or equal to 0, was specified.
IE.UPN	**Insufficient dynamic storage.** The driver could not allocate an AST block to notify the parent task of an offspring task request, or the driver could not allocate an intermediate buffer in the Executive pool.

Symbol	Description
SE.NIH	**Not in handler.** An offspring task attempted to assert TC.FDX, or a terminal characteristic other than one of those in Table 12-3 was specified.

Table 12-5: Virtual Terminal Status, Parent Task Requests

Symbol	Description
IS.SUC	**Successful completion.** The operation specified in the QIO directive was completed successfully. The second word of the I/O status block indicates the number of bytes transferred on a read or write operation.
IE.EOF	**End of file encountered.** The IO.STC function was completed successfully.
IE.BAD	**Bad parameters.** The parent task specified a buffer size that exceeded the system maximum of 512 (decimal) bytes
IE.DUN	**Device not attachable.** An IO.ATT or IO.DET function was issued by the parent task.
IE.IFC	**Invalid function code.** A read, write, or IO.STC function was issued without a pending offspring task request. This status can occur if the offspring task cancels a pending read or write request. This function code is also returned when IO.STC is issued to enable intermediate buffering on a virtual terminal unit whose buffer size, specified in the Create Virtual Terminal directive, exceeds the system maximum of 512 (decimal) bytes.
SE.NIH	**Not in handler.** A terminal characteristic other than one of those in Table 12-3 was specified in an SF.GMC or SF.SMC request.

12.4 TASK STACK FORMATS, AST ROUTINES

For tasks that use virtual terminals, the stack format can appear as follows during entrance to an AST service routine:

- **Offspring Read/Write QIO Occurs**

  As a result of virtual terminal offspring read or write operations, a parent AST routine may be entered with the following values on the task stack:

  SP+14 - Event flag mask word
  SP+12 - PS of task prior to AST
  SP+10 - PC of task prior to AST
  SP+06 - Task's DSW
  SP+04 - Third parameter word (Vertical Format Control, VFC) of the offspring task
  SP+02 - Byte count of offspring request
  SP+00 - Virtual terminal unit number (low byte); I/O subfunction code of offspring request (high byte)

- **Offspring Attach/Detach QIO Occurs**

  If the Attach/Detach AST routine is entered for a virtual terminal attach, the task's stack contains the following values:

  SP+14 - Event flag mask word
  SP+12 - PS of task prior to AST
  SP+10 - PC of task prior to AST
  SP+06 - Task's DSW
  SP+04 - Second word of offspring task name
  SP+02 - First word of offspring task name
  SP+00 - Virtual terminal unit number (low byte); I/O subfunction code of offspring request (high byte)

12.5 LOGIN FOR VIRTUAL TERMINALS

In order to establish user context for a virtual terminal, the parent task must spawn (SPWN$) the login task, whose installed name is ...$LO. You must specify the unit number of the virtual terminal in the unit parameter of the SPWN$ directive. Additionally, you must supply the following command line to the login task:

LOGIN/QUIET username password

The username can be a maximum of 14 characters, and the password can be up to 6 characters in length.

12.6 NULL VIRTUAL TERMINALS

A null virtual terminal is a special case of the standard virtual terminal. Its purpose is to allow a task to establish a TI: with a known user environment for background operations.

You use a null virtual terminal in situations where a parent task does not require access to an offspring's terminal I/O stream. When such access is required, use a standard virtual terminal instead.

The system treats a null virtual terminal as a null device.

Allowable I/O functions on LUNs assigned to a null virtual terminal are IO.WLB, IO.WVB, IO.RLB, IO.RVB, and IO.RPR. Write operations always return success (IS.SUC) and read operations (including IO.RPR) return end-of-file (IE.EOF) in the I/O Status Block. Other normal virtual terminal functions are invalid, and thus return IE.IFC.

To create a null virtual terminal, invoke the CRVT$ system directive with the following arguments:

CRVT$ [iast],[oast],[aast],mlen

iast This parameter (input ast address), is ignored, but a placeholder is required. You can specify a null value.

oast This parameter (output ast address), is ignored, but a placeholder is required. You can specify a null value.

aast This parameter (attach ast address), is ignored, but a placeholder is required. You can specify a null value.

mlen This parameter specifies the buffer length. It must be -1 to indicate that you want to create a null virtual terminal.

Three pathways lead to elimination of the null virtual terminal:

1. Parent task issues ELVT$ directive.

2. Parent task exits (ELVT$ functionality issued on behalf of parent).

3. Offspring count of virtual terminal goes to 0.

Note that the last case applies only to null virtual terminals. For normal virtual terminals, only the first or second case leads to elimination of the virtual terminal.

When the system eliminates a virtual terminal, the system logout task automatically logs out the virtual terminal, if necessary. No intervention by either parent or offspring to effect logout is required.

CHAPTER 13

THE COMMUNICATION DRIVER

The Professional communication driver, called XKDRV, allows you to use the communication port in asynchronous mode. The driver provides the following features:

- Full duplex operation
- Input buffering
- Unsolicited event ASTs
- Transfer length of up to 8128 bytes
- Optional timeout on solicited input
- Optional XON/XOFF support
- Modem support
- Mini-Exchange support
- Parity, framing, and overrun error reporting

Note that DIGITAL supplies a separate driver, XTDRV, to handle communication lines provided by the Telephone Management System. See the *Telephone Management System Programmer's Manual* for details.

13.1 GET LUN INFORMATION FOR COMMUNICATION DRIVER

Word 2 of the buffer filled by the Get LUN Information system directive (the first characteristic word) contains the information noted in Table 13-1 for the driver. A setting of 1 indicates that the described characteristic is true.

Table 13-1: Get LUN Information for Communication Driver

Bit	Setting	Meaning
0	1	Record-oriented device
1	1	Carriage-control device
2	0	Terminal device
3	0	File structured device
4	0	Single-directory device
5	0	Sequential device
6	0	Mass storage device
7	0	User-mode diagnostics supported, device dependent
8	0	Device supports 22-bit direct addressing
9	0	Unit software write-locked
10	0	Input spooled device
11	0	Output spooled device
12	0	Pseudo device
13	0	Device mountable as communications channel
14	0	Device mountable as a FILES-11 volume
15	0	Device mountable

Words 3 and 4 of the buffer are undefined. Word 5 indicates the size of the internal input ring buffer.

13.2 QIO MACRO FOR COMMUNICATION DRIVER

Table 13-2 lists the standard and device-specific functions of the QIO macro that are valid for the communication driver.

Table 13-2: QIO Functions for Communication Driver

Format	Function
Standard Functions	
QIO$C IO.ATT,...	Attach device
QIO$C IO.DET,...	Detach device
QIO$C IO.KIL,...	Cancel I/O requests
QIO$C IO.RLB,...,<stadd,size[,tmo]>	Read logical block (read input into buffer)
QIO$C IO.RVB,...,<stadd,size[,tmo]>	Read virtual block (read input into buffer)
QIO$C IO.WLB,...,<stadd,size,vfc>	Write logical block (send contents of buffer)
QIO$C IO.WVB,...,<stadd,size,vfc>	Write virtual block (send contents of buffer)
Device-Specific Functions	
QIO$C IO.ANS,...,<stadd,size>	Initiate a connection in answer mode, either in response to a ringing line, or if a connection already exists
QIO$C IO.ATA,...,<ast[,param2]>	Attach device, specify unsolicited event AST
QIO$C IO.BRK,...,<type>	Send a BREAK
QIO$C IO.CON,...,<stadd,size[,tmo]>	Dial and connect
QIO$C IO.HNG,...	Hang up a line

Format	Function
QIO$C IO.LTI,...,<stadd,size[,param3]>	Connect for unsolicited event ASTs while detached
QIO$C IO.ORG,...,<stadd,size[,tmo]>	Initiate a connection in originate mode, assuming the line has already been connected
QIO$C IO.RAL,...,<stadd,size[,tmo]>	Read logical block, pass all bits
QIO$C IO.RNE,...,<stadd,size[,tmo]>	Read logical block, do not echo
QIO$C IO.TRM,...,<stadd,1>	Unload driver*
QIO$C IO.UTI,...	Disable unsolicited event ASTs while detached
QIO$C IO.WAL,...,<stadd,size>	Write logical block, pass all bits
QIO$C SF.GMC,...,<stadd,size>	Get multiple characteristics
QIO$C SF.SMC,...,<stadd,size>	Set multiple characteristics

Parameters Shown in Table 13-2

ast The entry point for an unsolicited event AST.

param2 An optional user-specified AST parameter (1 byte).

param3 An optional user-specified AST parameter (1 byte).

size The size of the stadd data buffer, in bytes. The specified size must be greater than 0 and less than or equal to 8128. The buffer must be within the task's address space.

---

* Use of IO.TRM is restricted. See Section 13.2.2.10.

stadd The starting address of the data buffer. The address can be byte aligned.

type Either 0 to indicate a break or 1 to indicate a long space.

tmo An optional timeout count you can use in conjunction with TF.TMO on any Read request, or on a Connect (IO.CON) or Originate (IO.ORG) request.

Specify a timeout as follows:

```
.BYTE     x,y
```

where x is the number of ten-second intervals, up to 255 (decimal), and y is the number of one-second intervals, also up to 255. The longest possible timeout interval that you can specify is 255 seconds. If the timeout value is larger than 255 seconds, the driver uses 255 seconds. Section 13.6 describes the effect of the timeout parameters on specific requests.

vfc A character for vertical format control. See Table 11-16 in the chapter describing the terminal driver for a list of the characters.

13.2.1 Using Subfunction Bits

You can select many device-specific functions supported by the communication driver by using subfunction bits. To select one or more subfunctions, you logically OR the subfunction(s) together with a QIO function.

For example, the following QIO request IO.RLB uses two subfunction bits to perform a Read All Bits operation (TF.RAL) with a timeout period (TF.TMO).

```
QIO$C IO.RLB!TF.RAL!TF.TMO,...,<stadd,size,tmo>
```

Table 13-3 describes the subfunction bit symbols.

Table 13-3: Subfunction Bit Symbols

Symbolic	Subfunction
TF.AST	Unsolicited-input-character AST
TF.RAL	Read all bits
TF.RNE	Read with no echo [1]
TF.TMO	Operation will time out
TF.WAL	Write all bits

Note to Table 13-3

1. The communication driver ignores the TF.RNE subfunction bit, since it handles a Read Logical Block (IO.RLB) identically to a Read with No Echo (IO.RNE or IO.RLB!TF.RNE). The driver does not echo input characters.

Table 13-4 shows which subfunction bits you can logically OR with the QIO functions.

Table 13-4: Subfunction Bits Allowed for Driver Requests

Function	Equivalent Subfunction	Allowed Subfunction Bits
Standard Functions		
IO.ATT	None	TF.AST
IO.DET	None	None
IO.KIL	None	None
IO.RLB	None	TF.RAL
IO.RVB	None	TF.RAL

Function	Equivalent Subfunction	Allowed Subfunction Bits
IO.WLB	None	TF.WAL
IO.WVB	None	TF.WAL
Device-Specific Functions		
IO.ANS	None	None
IO.ATA	IO.ATT!TF.AST	None
IO.BRK	None	None
IO.CON	None	TF.TMO
IO.HNG	None	None
IO.LTI	None	None
IO.ORG	None	TF.TMO
IO.RAL	IO.RLB!TF.RAL	TF.TMO
IO.RNE	IO.RLB!TF.RNE	TF.TMO
IO.TRM	None	None
IO.WAL	IO.WLB!TF.WAL	None
SF.GMC	None	None
SF.SMC	None	None

13.2.2 Device-Specific QIO Functions

This section describes each of the device-specific QIO functions.

13.2.2.1 IO.ANS - The Answer function establishes a connection in answer mode, either in response to a ringing line, or if a connection already exists. If a connection is not completed within 30 seconds, the driver returns an IE.DNR error. The buffer address is required, but the driver does not use it.

The format is:

```
QIO$C IO.ANS,...,<stadd,size>
```

13.2.2.2 IO.ATA and IO.ATT!TF.AST - IO.ATA is a variation of the Attach function, and is equivalent to IO.ATT logically ORed with the subfunction bit TF.AST. The function specifies an asynchronous system trap (AST) to process unsolicited events.

When an unsolicited event occurs, the resulting AST serves as notification of the unsolicited event. Upon entry to the AST, the top word of the task stack contains the event type (low byte) and, if specified, param2 (high byte), as shown in Table 13-5. Your task must remove the top word from the stack before exiting the AST.

Table 13-5: Task Stack Format

Current Stack Pointer	Contents
SP+10	Event flag mask word
SP+06	PS of task prior to AST
SP+04	PC of task prior to AST
SP+02	Task's directive status word
SP+00	The low byte contains the event type. The high byte contains param2 (IO.ATA) or param3 (IO.LTI), if you specified that optional parameter in the QIO request.

See Section 13.5 for more information on unsolicited events.

The format of IO.ATA is:

```
QIO$C   IO.ATA,...,<ast[,param2]>
```

13.2.2.3 IO.BRK - The IO.BRK function causes the driver to send either a break or a long space. If parameter 1 is zero, the driver sends a break. If parameter 1 is one, the driver sends a long space.

On the communication port, the duration of a break is approximately 235 milliseconds, and the duration of a long space is approximately 3.5 seconds.

The format is:

```
QIO$C IO.BRK,...,<type>
```

13.2.2.4 IO.CON - The IO.CON function causes the driver to dial and connect a line in originate mode.

If you do not specify TF.TMO, the request completes either when a connection is established or after 60 seconds.

The format is:

```
QIO$C   IO.CON,...,<stadd,size[,tmo]>
```

where stadd is the address of the telephone number to dial.

13.2.2.5 IO.HNG - The IO.HNG function causes the driver to hang up a phone line.

The format is:

```
QIO$C IO.HNG,...
```

13.2.2.6 IO.LTI - The IO.LTI function causes the driver to deliver unsolicited event notification ASTs to a specified task, if another task has not attached the driver.

The format is:

```
QIO$C   IO.LTI,...,<stadd,size[,param3]>
```

where stadd is the address of a three-word buffer of the form:

```
.WORD   AST_address
.RAD50  /first_half_of_task_name/
.RAD50  /second_half_of_task_name/
```

When an unsolicited event occurs, the resulting AST serves as notification of the unsolicited event. Upon entry to the AST, the top word of the task stack contains the event type (low byte) and, if specified, param3 (high byte). Your task must remove the top word from the stack before exiting the AST. Table 13-5 earlier in this chapter describes the task stack format. Also, see Section 13.5 for more information on unsolicited events.

13.2.2.7 IO.ORG - The IO.ORG function initiates a connection in originate mode, assuming the line has already been connected. The buffer address is required, but the driver does not use it.

The format is:

```
QIO$C IO.ORG,...,<stadd,size[,tmo]>
```

13.2.2.8 IO.RAL and IO.RLB!TF.RAL - IO.RAL is equivalent to IO.RLB logically ORed with the subfunction bit TF.RAL.

The Read All function causes the driver to pass all bits to the requesting task when the value of TC.FSZ is 8 or 9. The driver does not mask out the high-order bit. Use this function to temporarily bypass the setting of the TC.8BC characteristic.

Note that, unlike the terminal driver, this function does not cause the communication driver to pass CTRL/Q or CTRL/S to the requesting task. You must set the characteristic TC.BIN to enable the the driver to return CTRL/Q and CTRL/S to your task.

The format is:

```
QIO$C IO.RAL,...,<stadd,size[,tmo]>
```

13.2.2.9 IO.RNE and IO.RLB!TF.RNE - IO.RNE is equivalent to IO.RLB logically ORed with the subfunction bit TF.RNE.

Note that the communication driver treats the functions IO.RNE and IO.RLB exactly alike. Consequently, the driver ignores the subfunction bit TF.RNE when processing IO.RLB!TF.RNE; it does not echo input characters.

The format is:

```
QIO$C IO.RNE,...,<stadd,size[,tmo]>
```

13.2.2.10 IO.TRM - The IO.TRM function causes the system to unload the communication driver.

NOTE

This function is provided only for compatibility with programs written to run on P/OS V2.0A and earlier systems. For new programs, we strongly suggest that you use the PROLOD utility to unload a driver.

PROLOD is described in the *Guide to Writing a P/OS I/O Driver and Advanced Programmer's Notes.*

If you use IO.TRM, note the following:

- Your task must be attached in order to issue this function. The system does unload the communication driver when your task detaches the device.
- This function requires a buffer address and count; however, the system does not modify the buffer.

The format is:

```
QIO$C IO.TRM,...,<stadd,1>
```

13.2.2.11 IO.UTI - The IO.UTI function disables unsolicited event notification while the driver is not attached.

The format is:

```
QIO$C IO.UTI,...
```

13.2.2.12 IO.WAL and IO.WLB!TF.WAL - IO.WAL is equivalent to IO.WLB logically ORed with the subfunction bit TF.WAL.

Note that the communication driver treats the functions IO.WAL and IO.WLB exactly the same. Consequently, the driver ignores the subfunction bit TF.WAL when processing IO.WLB!TF.WAL.

The format is:

```
QIO$C IO.WAL,...,<stadd,size>
```

13.2.2.13 SF.GMC - The Get Multiple Characteristics function returns driver characteristics information. It complements the SF.SMC function.

The format is:

```
QIO$C       SF.GMC,...,<stadd,size>
```

where stadd is the starting address of a data buffer whose length in bytes is the value given in the size parameter. Each word in the buffer has the form:

```
.BYTE       charname
.BYTE       0
```

where charname is one of the characteristic bit names given in Table 13-6. The value returned in the high byte of each byte-pair is the value of that characteristic.

Table 13-6: Driver Characteristics for SF.GMC and SF.SMC

Symbol	Valid Values	Meaning
TC.ARC	0-9.	Auto-answer ring count (0 means don't answer).
TC.BIN	0,1	Enable or disable XON/XOFF support.
TC.CTS	0,1	Resume or suspend output.
TC.EPA	0,1	Odd or even parity (if TC.PAR is specified).
TC.FSZ	Note 1	Character width including parity (if any).
TC.PAR	0,1 Note 1	Enable parity checking and generation.
TC.RSP	Note 2	Receiver speed (bits-per-second).
TC.STB	1,2	Number of stop bits.
TC.TBF	Note 3	Input ring buffer count or flush.
TC.TRN	Note 4	Set translate table.
TC.XMM	0,1	Disable or enable maintenance mode.
TC.XSP	Note 2	Transmitter speed (bits-per-second).
TC.8BC	0,1 Note 1	Pass 8-bit characters on input and output.
XT.MTP	Note 5	Modem type.

Note 1 to Table 13-6

TC.FSZ is the frame size of a character. It is the number of data bits per character, plus 1 if parity is enabled.

TC.FSZ and TC.PAR interact with each other to determine the number of data bits returned to the task. Table 13-7 shows the relationship of these characteristics.

Table 13-7: TC.FSZ and TC.PAR Relationship

TC.FSZ	TC.PAR	Number of Data Bits Returned to Task
9	1	8
8	0	8
8	1	7
7	0	7
7	1	6
6	0	6
6	1	5
5	0	5

Two combinations do not appear in Table 13-7: TC.FSZ=9 with TC.PAR=0, and TC.FSZ=5 with TC.PAR=1. These two combinations are invalid, and the driver returns an error if you use either combination. To avoid this problem, always set the value of TC.FSZ first. The driver automatically enables or disables parity if the value of TC.FSZ is 9 or 5.

If the value of TC.FSZ is 8 or 9, the driver further modifies the number of data bits it returns to the task by the value of TC.8BC. If TC.8BC is set to 1, the driver returns all 8 data bits to the task. If TC.8BC is set to 0, the driver returns only 7 data bits to the task. Setting TC.BIN to a value of 1 or using the IO.RAL function overrides the value of TC.8BC.

Note 2 to Table 13-6

Table 13-8 shows TC.RSP and TC.XSP values and their corresponding baud rates.

Table 13-8: Receiver and Transmitter Speed Values

TC.RSP or TC.XSP Symbolic	Decimal Value of Symbolic	Actual baud rate (bits per second)
S.50	2	50
S.75	3	75
S.110	5	110
S.134	6	134.5
S.150	7	150
S.300	9	300
S.600	10	600
S.1200	11	1200
S.1800	12	1800
S.2000	13	2000
S.2400	14	2400
S.3600	15	3600
S.4800	16	4800
S.7200	17	7200
S.9600	18	9600
S.EXTA	19	External clocking
S.19.2	21	19200

Note 3 to Table 13-6

When you use the TC.TBF characteristic with SF.GMC, the driver returns the number of unprocessed characters in the input buffer. If there are more than 255 characters in the buffer, the driver returns the value 255. When you use TC.TBF with SF.SMC, the driver flushes the input buffer.

Note 4 to Table 13-6

The TC.TRN characteristic allows you to specify a translate table. The *translate table* is collection of data that consists of the following:

- A list of translations to perform on particular phone number characters, typically to remove format characters such as parentheses, hyphens, and blank spaces.
- The start sequence of the modem in use.
- The end sequence of the modem in use.

The format of the translate table follows.

```
.BYTE TC.TRN,count1,count2,count3
.BYTE chr1,rep_chr1
.             ;Translation section.
.             ;First character is character
.             ;to translate. Second character
.             ;is the replacement character.
.BYTE chrn,rep_chrn
              ;End of translate section.
.BYTE ss1,ss2,...
              ;ss1,ss2... are the
              ;start characters.
.BYTE es1,es2,...
              ;es1,es2... are the
              ;end characters.
.EVEN
```

In the above format, count1 is the length of the translation section, count2 is the length of the start sequence, and count3 is the length of the end sequence.

If a character in the telephone number matches a character in the translation section, the driver converts the telephone number character into the corresponding replacement character. If the corresponding replacement character is 0 (binary), the driver ignores the character from the telephone number.

If you specify the start sequence string, the driver sends that string to the autodialer before the phone number. If you specify the end sequence string, the driver sends that string to the autodialer after the phone number.

Note the following regarding the translate table and use of TC.TRN:

- The maximum size of the start and end sequences is 16 (decimal) bytes.
- The maximum size of the translate table is 80 (decimal) bytes.
- Any or all sections of the translate table can be empty.
- Start and end sequences are not translated using the translate table.
- TC.TRN is a write-only parameter; the driver returns an SE.NIH error if you attempt to use TC.TRN with the SF.GMC function.
- The characteristic in the SF.SMC function following TC.TRN must begin on a word boundary.

Note 5 to Table 13-6

Table 13-9 shows the possible values for the XT.MTP characteristic.

Table 13-9: Modem Type Values (XT.MTP)

XT.MTP Symbol	Decimal Value	Modem Type
XTM.NO	-1	No modem, hardwired line
XTM.FS	0	USFSK 0..300 baud Bell 103J (TMS)
XTM.PS	1	DPSK 1200 baud Bell 212 (TMS)
XTM.US	10	Mini-Exchange

13.2.2.14 SF.SMC - This function enables a task to set and reset the characteristics of the driver. Set Multiple Characteristics is the complementary function to SF.GMC. The format is:

```
QIO$C   SF.SMC,...,<stadd,size>
```

where stadd is the starting address of a data buffer whose length in bytes is given in the size parameter. Each word in the buffer has the form:

```
.BYTE charname
.BYTE value
```

where charname is one the bit names given in Table 13-6 and "value" is a value in the range given in Table 13-6.

13.3 STATUS RETURNS FOR COMMUNICATION DRIVER

Table 13-10 lists error and status conditions that the communication driver returns to the low byte of the first word of the I/O Status Block.

The SF.GMC and SF.SMC functions return the SE.xxx codes as described in Sections 13.2.2.13 and 13.2.2.14. When the driver returns any of these codes, the low byte in the first word of the I/O Status Block contains IE.ABO. The high byte contains the SE.xxx error code.

The second IOSB word contains an offset (starting from 0) to the byte in error in the QIO's stadd buffer.

Table 13-10: Communication Driver Status Returns

Symbol	Description
IS.SUC	**Successful completion.** The operation specified in the QIO directive completed successfully. If the operation involved reading or writing, you can examine the second word of the I/O Status Block to determine the number of bytes processed. The input buffer contains those bytes.
IS.PND	**I/O request pending.** The driver has not yet executed the operations specified in the QIO directive. The I/O Status Block contains all zeros.

STATUS RETURNS FOR COMMUNICATION DRIVER

Symbol	Description
IS.TMO	**Successful Read with time-out.** A time-out terminated the input from the communication port when the tmo parameter is a nonzero value. The input buffer contains the bytes read.
IE.ABO	**Operation aborted.** An IO.KIL request cancelled the specified I/O operation while in progress or while in the I/O queue. The second word of the I/O Status Block indicates the number of bytes placed in the buffer before the kill occurred.
IE.ALC	**Allocation failure.** The total size of the phone number specified by an IO.CON request, plus the start and end sequences, is larger than the driver's internal buffer.
IE.BAD	**Bad parameters.** The size of the data buffer in the QIO request exceeds 8128 bytes.
IE.BCC	**Framing error.** The read I/O request was terminated because of a framing error. The buffer returned does not include the character on which the error occurred. It is returned in the high byte of the first word of the I/O status block.
IE.CNR	**Connection rejected.** A connection already exists, or the connection attempt was rejected by a Mini-Exchange.
IE.DAA	**Device already attached.** The issuing task already attached the physical device unit specified in the IO.ATT function. This code indicates that the issuing task has already attached the desired physical device unit, not that another task attached the unit. If the attach specified TF.AST, the subfunction bit has no effect.
IE.DAO	**Data overrun.** The read I/O request was terminated due to a data overrun condition. Data has been lost.
IE.DNA	**Device not attached.** The issuing task did not attach the physical device unit specified in an IO.DET or IO.TRM function. This code has no bearing on the attachment status of other tasks.

STATUS RETURNS FOR COMMUNICATION DRIVER

Symbol	Description
IE.DNR	**Device not ready.** The physical device specified in the QIO directive was not ready to perform the desired I/O operation. The driver returns this code to indicate that an attempt was made to perform a function on a line connected to a modem without carrier present, or to indicate that a connection was not established within the time-out period specified by an IO.CON, IO.ANS, or IO.ORG request.
IE.IFC	**Invalid function.** An I/O request specified a function code that is invalid for the communication port.
IE.OFL	**Device off-line.** The physical device-unit associated with the LUN specified in the QIO directive was not online.
IE.VER	**Parity error.** The read I/O request has been terminated due to a character parity error. The character with the error is returned in the high byte of the first word in the I/O status block.
SE.NIH	**Not in handler.** In an SF.GMC or SF.SMC request, you specified a characteristic that is not implemented in the communication driver. For a list of implemented characteristics, see Table 13-6.
SE.VAL	**Invalid characteristic value.** You specified an invalid characteristic in an SF.SMC request. For a list of valid characteristics, see Table 13-6.

13.4 FULL-DUPLEX OPERATION

The communication driver attempts to simultaneously service one Read request and one Write request. Note that, unlike the terminal driver, the SF.SMC function is not blocked until the line is idle. Resetting characteristics during I/O operations can cause unpredictable results.

13.5 UNSOLICITED EVENT PROCESSING

If a task attaches for unsolicited event ASTs (IO.ATA), the driver dispatches an AST whenever any of the events listed in Table 13-11 occurs.

Table 13-11: Unsolicited Event Types

Symbol	Decimal Value	Event Type
XTU.CD	2	Carrier detect
XTU.CL	4	Carrier loss
XTU.OF	8	XOFF received
XTU.ON	10	XON received
XTU.RI	12	Ring
XTU.UI	0	Unsolicited input

Upon entering an AST, the driver places the event type in the low byte of the top word of the stack, and param2 (for IO.ATA) or param3 (for IO.LTI) in the high byte. Note that the driver processes an XTU.UI event differently from the others.

13.5.1 XTU.UI Event Type Processing

If the event type is XTU.UI (unsolicited input), the AST becomes disabled until the task issues a Read request. Once the Read request has completed, the AST is re-enabled for new unsolicited events.

13.6 EFFECT OF TIMEOUT ON QIO REQUEST

The optional timeout parameter on Read, IO.CON, and IO.ORG requests affects the action of the request. The following sections describe each request when you issue it with a timeout value.

13.6.1 Timeout on Read Requests (IO.RLB!TF.TMO)

tmo = 0

The request completes immediately after the driver has transferred as many characters as are available, less than or equal to the size parameter. The driver returns the number of bytes transferred in the second I/O status word.

tmo <> 0

The request completes after the timeout period or when the driver has transferred the requested number of bytes. The driver returns the number of bytes transferred in the second word of the I/O Status Block.

13.6.2 Timeout on IO.CON Request (IO.CON!TF.TMO)

tmo = 0

The request completes immediately. If carrier is not present after 60 seconds, the driver drops DTR and RTS.

tmo <> 0

The request completes after the timeout period, or after a connection is established. If the timeout period expires, the driver drops DTR and RTS and returns an IE.DNR error in the first word of the I/O Status Block. If carrier comes up before the timeout period expires, the driver returns IS.SUC in the first word of the I/O Status Block.

13.6.3 Timeout on IO.ORG Request (IO.OAG!TF.TMO)

tmo = 0

The request completes immediately. If carrier is down, the driver drops DTR and RTS and returns an IE.DNR error in the I/O Status Block. If carrier is up, the driver returns IS.SUC in the I/O Status Block.

tmo <> 0

The request completes after the timeout period, or after a connection is established. If the timeout period expires, the driver drops DTR and RTS and returns an IE.DNR error in the I/O Status Block. If carrier is up, or if carrier comes

up before the timeout period expires, the driver returns IS.SUC in the I/O Status Block.

13.7 XON/XOFF SUPPORT

If your task requests XON/XOFF support (TC.BIN = 0), the driver transmits an XOFF whenever the ring buffer is three-quarters filled, and transmits an XON whenever the buffer empties below the one-quarter point. Because of this, your task should not pass XON/XOFF control characters to the driver for transmission.

If the driver receives an XOFF, it blocks transmission. If your task is attached for unsolicited event ASTs, the driver dispatches an XTU.OF event. In any case, the TC.CTS parameter reflects the XON/XOFF state of the line.

If your task does not request XON/XOFF support (TC.BIN = 1), and the value of XT.MTP is XTM.NO (no modem), the driver uses the Clear to Send and Request to Send lines in place of XON/XOFF control characters. Consequently, state changes of this line cause unsolicited event ASTs and modify the value of TC.CTS.

The driver responds to state changes on the Clear to Send line to control the transmission of data. The driver uses the Request to Send line to signal the remote transmitter either to transmit (RTS high) or not to transmit (RTS low).

APPENDIXES

APPENDIX A

SUMMARY OF I/O FUNCTION AND SUBFUNCTION CODES

This appendix summarizes the I/O function and subfunction codes and their values for all bundled P/OS device drivers. Section A.1 describes the I/O function codes. Section A.2 describes the I/O subfunction codes.

A.1 I/O FUNCTION CODE VALUES

A table for each device lists the available function codes, the word value in octal, and the high and low bytes in octal. Each table presents the codes in alphabetical order.

You can refer to the functions symbolically by invoking the system macros FILIO$ (standard I/O functions) and SPCIO$ (special I/O functions), or by allowing them to be defined at task-build time from the system object library.

Table A-1: Function Code Values, Communication Driver (XKDRV)

Code	Word	Byte 1	Byte 0	Description
IO.ANS	015420	033	020	Initiate connection in answer mode
IO.ATA	001410	003	010	Attach with ASTs
IO.ATT	001400	003	000	Attach a device to a task
IO.BRK	003200	006	200	Send short or long break
IO.CON	015400	033	000	Dial telephone and originate
IO.DET	002000	004	000	Detach a device from a task
IO.HNG	003000	006	000	Hangup dial-up line
IO.KIL	000012	000	012	Kill current request
IO.LTI	007400	017	000	Connect for unsolicited event ASTs while detached
IO.ORG	015410	033	010	Initiate connection in originate mod

Code	Word	Byte 1	Byte 0	Description
IO.RAL	001010	002	010	Read passing all data bits
IO.RLB	001000	002	000	Read logical block
IO.RNE	001020	002	020	Read without echo
IO.RVB	010400	021	000	Read virtual block
IO.TRM	002410	005	010	Termination function
IO.UTI	011420	023	020	Disable unsolicited event ASTs while detached
IO.WAL	000410	001	010	Write passing all characters
IO.WLB	000400	001	000	Write logical block
IO.WVB	011000	022	000	Write virtual block
SF.GMC	002560	005	160	Get multiple characteristics
SF.SMC	002440	005	040	Set multiple characteristics

Table A-2: Function Code Values, Disk Drivers (DZDRV and DWDRV)

Code	Word	Byte 1	Byte 0	Description
IO.ATT	001400	003	000	Attach a device to a task
IO.DET	002000	004	000	Detach a device from a task
IO.KIL	000012	000	012	Kill current request
IO.RLB	001000	002	000	Read logical block
IO.RPB	001040	002	040	Read physical block
IO.RVB	010400	021	000	Read virtual block
IO.WLB	000400	001	000	Write logical block
IO.WPB	000440	001	040	Write physical block
IO.WVB	011000	022	000	Write virtual block

Table A-3: Function Code Values, Terminal Driver (TTDRV)

Code	Word	Byte 1	Byte 0	Description
IO.ATA	001410	003	010	Attach with ASTs
IO.ATT	001400	003	000	Attach a device to a task
IO.CCO	000440	001	040	Write with cancel CTRL/O

Code	Word	Byte 1	Byte 0	Description
IO.DET	002000	004	000	Detach a device from a task
IO.GTS	002400	005	000	Get terminal support
IO.KIL	000012	000	012	Kill current request
IO.RAL	001010	002	010	Read passing all data bits
IO.RLB	001000	002	000	Read logical block
IO.RNE	001020	002	020	Read without echo
IO.RPR	004400	011	000	Read with prompt
IO.RSD	006030	014	030	Read special data
IO.RST	001001	002	001	Read with special terminator
IO.RTT	005001	012	001	Read with terminator table
IO.RVB	010400	021	000	Read virtual block
IO.WAL	000410	001	010	Write passing all characters
IO.WBT	000500	001	100	Write with breakthrough
IO.WLB	000400	001	000	Write logical block
IO.WSD	005410	013	010	Write special data
IO.WVB	011000	022	000	Write virtual block
SF.GMC	002560	005	160	Get multiple characteristics
SF.SMC	002440	005	040	Set multiple characteristics

Table A-4: Function Code Values, TMS Driver (XTDRV)

Code	Word	Byte 1	Byte 0	Description
IO.ANS	015420	033	020	Initiate connection in answer mode
IO.ATA	001410	003	010	Attach with ASTs
IO.ATT	001400	003	000	Attach a device to a task
IO.BRK	003200	006	200	Send short or long break
IO.CON	015400	033	000	Dial telephone and originate
IO.DET	002000	004	000	Detach a device from a task
IO.HLD	003100	006	100	Hang up but leave line on hold
IO.HNG	003000	006	000	Hang up dial-up line
IO.KIL	000012	000	012	Kill current request
IO.LTI	007400	017	000	Connect for unsolicited event ASTs while detached
IO.ORG	015410	033	010	Initiate connection in originate mode
IO.RAL	001010	002	010	Read passing all characters
IO.RLB	001000	002	000	Read logical block
IO.RNE	001020	002	020	Read without echo
IO.RVB	010400	021	000	Read virtual block

Code	Word	Byte 1	Byte 0	Description
IO.UTI	011420	023	020	Disable unsolicited event ASTs while detached
IO.WAL	000410	001	010	Write passing all characters
IO.WLB	000400	001	000	Write logical block
IO.WVB	011000	022	000	Write virtual block
IO.WSD	005410	013	010	Write special data to modify voice unit indicators
SF.GMC	002560	005	160	Get multiple characteristics
SF.SMC	002440	005	040	Set multiple characteristics

A.2 I/O SUBFUNCTION CODE VALUES

I/O subfunction codes are bit values with which you can logically OR particular function codes to obtain specific results (usually to create a function synonym).

A table for each device groups the available subfunction codes according to the I/O functions that you can use them with. Within each group, the subfunctions are ordered alphabetically. The bit value for each subfunction is shown in octal.

Table A-5: Subfunction Code (Bit) Values, XKDRV and XTDRV

Code	Value	Description
Use with IO.ATT:		
TF.AST	010	Specify ASTs in attach
Use with IO.RLB:		
TF.RAL	010	Read pass all
TF.RNE	020	Read with no echo
Use with IO.COM, IO.ORG, any READ:		
TF.TMO	200	Read with timeout

Code	Value	Description
Use with IO.WLB		
TF.WAL	010	Write pass all

Table A-6: Subfunction Code (Bit) Values, TTDRV

Code	Value	Description
Use with IO.RLB:		
TF.RTT	040	Read with terminator table
TF.RAL	010	Read pass all
TF.RNE	020	Read with no echo
TF.RST	001	Read with special terminators
TF.TMO	200	Read with timeout
Use with IO.RPR:		
TF.BIN	002	Send prompt as "pass all"
TF.RTT	040	Read with terminator table
TF.RAL	010	Read pass all
TF.RNE	020	Read with no echo
TF.RST	001	Read with special terminators
TF.TMO	200	Read with timeout
TF.XOF	100	Send XOF after prompt
Use with IO.WLB:		
TF.CCO	040	Cancel CTRL/O
TF.RCU	001	Restore cursor position
TF.WAL	010	Write pass all
TF.WBT	100	Break through read
TF.WMS	020	Write suppressible message

Code	Value	Description
Use with IO.ATT:		
TF.AST	010	Specify ASTs in attach
TF.ESQ	020	Recognize escape sequences
TF.NOT	002	Notification only for type-ahead
TF.XCC	001	Do not trap CTRL/C

Table A-7: Subfunction Code (Bit) Values, DZDRV and DWDRV

Code	Value	Description
Use with IO.RLB, IO.WLB:		
IQ.X	001	Inhibit retry attempts for error recovery (use on all I/O functions)

APPENDIX B

SUMMARY OF DSW AND IO STATUS CODES

This appendix summarizes the P/OS status codes:

- Section B.1 summarizes DSW codes that the system returns in the Directive Status Word (DSW).
- Section B.2 summarizes the standard I/O status codes that the system returns in the I/O Status Block (IOSB).
- Section B.3 groups all I/O status codes according to which device generates them.

B.1 STATUS CODES RETURNED IN DIRECTIVE STATUS WORD (DSW)

The symbols listed in the following tables are associated with the directive status codes returned by the Executive. They are determined (by default) at task-build time. To include these in a MACRO-11 program, use the following two lines of code:

```
.MCALL  DRERR$
DRERR$
```

Table B-1 lists the success codes; Table B-2 lists the error codes. The tables list both the decimal value and the octal word value for each code.

Table B-1: DSW Success Codes

Symbol	Decimal	Octal	Description
IS.CLR	+00	000000	Event flag was clear (from clear event flag directive)
IS.SUC	+01	000001	Operation complete -- success
IS.SET	+02	000002	Event flag was set (from set event flag directive)
IS.SPD	+02	000002	Task was suspended
IS.SUP	+03	000003	Logical name superseded

Table B-2: DSW Error Codes

Symbol	Decimal	Octal	Description
IE.UPN	-01	177777	Insufficient dynamic storage; see IE.NDR
IE.INS	-02	177776	Specified task not installed
IE.PTS	-03	177775	Partition too small for task
IE.UNS	-04	177774	Insufficient dynamic storage for send
IE.ULN	-05	177773	Unassigned LUN
IE.HWR	-06	177772	Device handler not resident
IE.ACT	-07	177771	Task not active
IE.ITS	-08	177770	Directive inconsistent with task state
IE.FIX	-09	177767	Task already fixed/unfixed
IE.CKP	-10	177766	Issuing task not checkpointable
IE.TCH	-11	177765	Task is checkpointable
IE.RBS	-15	177761	Receive buffer is too small
IE.PRI	-16	177760	Privilege violation
IE.RSU	-17	177757	Resource in use
IE.NSW	-18	177756	No swap space available
IE.ILV	-19	177755	Invalid vector specified
IE.ITN	-20	177754	Invalid table number
IE.LNF	-21	177753	Logical name not found
IE.AST	-80	177660	Directive issued/not issued from AST
IE.MAP	-81	177657	Invalid mapping specified
IE.IOP	-83	177655	Window has I/O in progress
IE.ALG	-84	177654	Alignment error
IE.WOV	-85	177653	Address window allocation overflow
IE.NVR	-86	177652	Invalid region ID
IE.NVW	-87	177651	Invalid address window ID
IE.ITP	-88	177650	Invalid TI parameter

Symbol	Decimal	Octal	Description
IE.IBS	-89	177647	Invalid send buffer size (greater than 255.)
IE.LNL	-90	177646	LUN locked in use
IE.IUI	-91	177645	Invalid UIC
IE.IDU	-92	177644	Invalid device or unit
IE.ITI	-93	177643	Invalid time parameters
IE.PNS	-94	177642	Partition/region not in system
IE.IPR	-95	177641	Invalid priority (greater than 250.)
IE.ILU	-96	177640	Invalid LUN
IE.IEF	-97	177637	Invalid event flag (greater than 64.)
IE.ADP	-98	177636	Part of DPB out of user's space
IE.SDP	-99	177635	DIC, DPB size, or subfunction invalid

B.2 I/O STATUS CODES (STANDARD)

I/O status codes are the low-byte values returned in the first word of the I/O status block on completion of an I/O function. This section summarizes all the standard codes available under P/OS. (See Section B.3 for a summary of the device-specific codes returned by the drivers bundled with P/OS.)

In your MACRO-11 program, you can refer to the codes symbolically by invoking the system macro IOERR$.

Table B-3 lists the success codes. Table B-4 lists the error codes. Octal values listed in both tables consist of the low-order byte of the complete word value (two's complement of the decimal number).

Table B-3: I/O Success Codes, Standard

Symbol	Decimal	Octal	Description
IS.PND	00	000	Operation pending
IS.SUC	+01	001	Operation complete, success
IS.RDD	+02	002	Floppy disk successful completion of a read physical, and deleted data mark was seen in sector header

Symbol	Decimal	Octal	Description
IS.DAO	+02	002	Successful but with data overrun (not to be confused with IE.DAO)
IS.TMO	+02	002	Successful completion on read terminated by timeout
IS.TNC	+02	002	(PCL) successful transfer but message truncated (receive buffer too small)
IS.CHW	+04	004	(IBM COMM) Data read was result of IBM host chained write operation
IS.BV	+05	005	(A/D READ) At least one bad value was read (remainder may be good); bad channel is indicated by a negative value in the buffer

Table B-4: I/O Error Codes, Standard

Symbol	Decimal	Octal	Description
IE.BAD	-01	377	Bad parameters
IE.IFC	-02	376	Invalid function code
IE.DNR	-03	375	Device not ready
IE.VER	-04	374	Parity error on device
IE.ONP	-05	373	Hardware option not present
IE.SPC	-06	372	Invalid user buffer
IE.DNA	-07	371	Device not attached
IE.DAA	-08	370	Device already attached
IE.DUN	-09	367	Device not attachable
IE.EOF	-10	366	End of file detected
IE.EOV	-11	365	End of volume detected
IE.WLK	-12	364	Write attempted to locked unit
IE.DAO	-13	363	Data overrun
IE.SRE	-14	362	Send/receive failure
IE.ABO	-15	361	Request terminated
IE.PRI	-16	360	Privilege violation
IE.RSU	-17	357	Sharable resource in use
IE.OVR	-18	356	Invalid overlay request
IE.BYT	-19	355	Odd byte count (or virtual address)
IE.BLK	-20	354	Logical block number too large
IE.MOD	-21	353	Invalid UDC module number
IE.CON	-22	352	UDC connect error

Symbol	Decimal	Octal	Description
IE.NOD	-23	351	Caller's nodes exhausted
IE.DFU	-24	350	Device full
IE.IFU	-25	347	Index file full
IE.NSF	-26	346	No such file
IE.LCK	-27	345	Locked from read/write access
IE.HFU	-28	344	File header full
IE.WAC	-29	343	Accessed for write
IE.CKS	-30	342	File header checksum failure
IE.WAT	-31	341	Attribute control list format error
IE.RER	-32	340	File processor device read error
IE.WER	-33	337	File processor device write error
IE.ALN	-34	336	File already accessed on LUN
IE.SNC	-35	335	File ID, file number check
IE.SQC	-36	334	File ID, sequence number check
IE.NLN	-37	333	No file accessed on LUN
IE.CLO	-38	332	File was not properly closed
IE.NBF	-39	331	OPEN--no buffer space available for file
IE.RBG	-40	330	Invalid record size
IE.NBK	-41	327	File exceeds space allocated, no blocks
IE.ILL	-42	326	Invalid operation on file descriptor block
IE.BTP	-43	325	Bad record type
IE.RAC	-44	324	Invalid record access bits set
IE.RAT	-45	323	Invalid record attributes bits set
IE.RCN	-46	322	Invalid record number--too large
IE.2DV	-48	320	Rename--two different devices
IE.FEX	-49	317	Rename--new file name already in use
IE.BDR	-50	316	Bad directory file
IE.RNM	-51	315	Can't rename old file system
IE.BDI	-52	314	Bad directory syntax
IE.FOP	-53	313	File already open
IE.BNM	-54	312	Bad file name
IE.BDV	-55	311	Bad device name
IE.BBE	-56	310	Bad block on device
IE.DUP	-57	307	ENTER--duplicate entry in directory
IE.STK	-58	306	Not enough stack space (FCS or FCP)
IE.FHE	-59	305	Fatal hardware error on device
IE.NFI	-60	304	File ID was not specified
IE.ISQ	-61	303	Invalid sequential operation
IE.EOT	-62	302	End of tape detected
IE.BVR	-63	301	Bad version number
IE.BHD	-64	300	Bad file header
IE.OFL	-65	277	Device off line
IE.BCC	-66	276	Block check, CRC, or framing error
IE.NNN	-68	274	No such node
IE.NFW	-69	273	Path lost to partner

Symbol	Decimal	Octal	Description
IE.DIS	-69	273	Path lost to partner (same as IE.NFW)
IE.BLB	-70	272	Bad logical buffer
IE.NDR	-72	270	No dynamic space available; see also IE.UPN
IE.URJ	-73	267	Connection rejected by user
IE.NRJ	-74	266	Connection rejected by network
IE.EXP	-75	265	File expiration date not reached
IE.BTF	-76	264	Bad tape format
IE.NNC	-77	263	Not ANSI "D" format byte count
IE.NDA	-78	262	No data available
IE.IES	-82	256	Invalid escape sequence
IE.PES	-83	255	Partial escape sequence
IE.ALC	-84	254	Allocation failure
IE.ULK	-85	253	Unlock error
IE.WCK	-86	252	Write check failure
IE.DSQ	-90	246	Disk quota exceeded
IE.IQU	-91	245	Inconsistent qualifier usage
IE.RES	-92	244	Circuit reset during operation
IE.TML	-93	243	Too many links to task
IE.NNT	-94	242	Not a network task
IE.TMO	-95	241	Timeout on request; see also IS.TMO
IE.CNR	-96	240	Connection rejected
IE.UKN	-97	237	Unknown name
IE.MII	-99	235	Media inserted incorrectly
IE.SPI	-100	234	Spindown ignored

B.3 I/O STATUS CODES (DEVICE SPECIFIC)

Each driver that is bundled with P/OS returns a subset of the standard I/O status codes. This section lists the status codes returned by each driver. Also listed are subcodes for any additional status information a driver returns in the high byte of the status word. (For example, the terminal driver returns input terminator information in the high byte of an IS.SUC return.)

NOTE

Subcodes returned with IS.SUC are word values. Subcodes returned with IE.ABO, and all I/O status codes, are byte values.

Table B-5: I/O Status Codes for the Terminal Driver, TTDRV

Symbol	Decimal	Octal	Description
IS.SUC	+01	001	Successful termination
IS.TMO	+02	002	Successful read terminated by timeout
IE.BAD	-01	377	Bad parameter
IE.IFC	-02	376	Invalid function
IE.DNR	-03	375	Device not ready
IE.SPC	-06	372	Invalid user buffer
IE.DNA	-07	371	Device not attached
IE.DAA	-08	370	Device already attached
IE.EOF	-10	366	End of file detected
IE.ABO	-15	361	Request terminated
IE.NOD	-23	351	Buffer allocation failure
IE.IES	-82	256	Invalid escape sequence
IE.PES	-83	255	Partial escape sequence

Table B-6: Full-Word Subcodes for IS.SUC Return in TTDRV

Symbol	Octal Byte 0	Octal Byte 1	Octal Word	Description
IS.CR	001	015	006401	Carriage return was terminator
IS.ESC	001	033	015401	Escape (ALTMODE) was terminator
IS.CC	001	003	001401	CTRL/C (INTERRUPT/DO) terminator
IS.ESQ	001	233	115401	Escape sequence was terminator
IS.PES	001	200	100001	Partial escape sequence terminator
IS.EOT	001	004	002001	EOT was terminator, block mode input
IS.TAB	001	011	004401	Tab was terminator, forms mode input

Table B-7: High-Byte Subcodes for IE.ABO Return in TTDRV

Symbol	Decimal	Octal	Description
SE.ICN	01	001	Invalid characteristic name
SE.FIX	02	002	Attempt to change fixed characteristic
SE.BIN	03	003	Invalid value for binary characteristic
SE.VAL	04	004	Invalid value for nonbinary characteristic
SE.TER	05	005	Invalid terminal type
SE.SPD	06	006	Invalid speed for interface
SE.SPL	07	007	Invalid split speed for interface
SE.PAR	08	010	Invalid parity type for interface
SE.LPR	09	011	Other invalid line parameters
SE.NSC	10	012	Interface has no settable characteristics
SE.UPN	11	013	No space to save default characteristics
SE.NIH	12	014	Characteristic not assembled in handler
SE.ATA	13	015	Terminal attached with ast notification
SE.NAT	14	016	Terminal not attached
SE.IAA	15	017	Invalid AST address specified

Table B-8: I/O Status for TMS Driver, XTDRV

Symbol	Decimal	Octal	Description
IS.PND	+00	000	Success, I/O request pending
IS.SUC	+01	001	Successful completion
IS.TMO	+02	002	Success, request timed out
IE.BAD	-01	377	Invalid digit or call issued for line 3
IE.DNR	-03	375	Device not ready
IE.DNA	-07	371	Device not attached
IE.DAA	-08	370	Device already attached to calling task
IE.DAO	-13	363	Data overrun
IE.ABO	-15	361	Operation aborted
IE.RSU	-17	357	Shared resource in use
IE.FHE	-59	305	Fatal hardware error on device
IE.OFL	-65	277	Device off line
IE.ALC	-84	254	Allocation failure
IE.TMO	-95	241	Could not connect within timeout period
IE.CNR	-96	240	Connection rejected

Table B-9: High-Byte Subcodes for IE.ABO Return in XTDRV

Symbol	Decimal	Octal	Description
SE.VAL	+04	004	Invalid value for nonbinary characteristic
SE.NIH	+12	010	Characteristic not assembled in handler

Table B-10: I/O Status for Communication Driver, XKDRV

Symbol	Decimal	Octal	Description
IS.PND	+00	000	Success, I/O request pending
IS.SUC	+01	001	Successful completion
IS.TMO	+02	002	Success, request timed out
IE.BAD	-01	377	Bad parameters
IE.IFC	-02	376	Invalid function
IE.DNR	-03	375	Device not ready
IE.DNA	-07	371	Device not attached
IE.DAA	-08	370	Device already attached to calling task
IE.ABO	-15	361	Operation aborted
IE.OFL	-65	277	Device off line
IE.ALC	-84	254	Allocation failure
IS.TMO	-95	241	Successful completion on a read
IE.CNR	-96	240	Connection rejected

Table B-11: High-Byte Subcodes for IE.ABO Return in XKDRV

Symbol	Decimal	Octal	Description
SE.VAL	+04	004	Invalid value for nonbinary characteristic
SE.NIH	+12	010	Characteristic not assembled in handler

Table B-12: I/O Status for Disk Drivers, DZDRV and DWDRV

Symbol	Decimal	Octal	Description
IS.PND	+00	000	Operation pending
IS.SUC	+01	001	Operation complete, success
IE.IFC	-02	376	Invalid function
IE.DNR	-03	375	Device not ready
IE.VER	-04	374	Parity error on device
IE.SPC	-06	372	Invalid user buffer
IE.WLK	-12	364	Write attempted to locked unit
IE.ABO	-15	361	Operation aborted
IE.PRI	-16	360	Privilege violation
IE.OVR	-18	356	Invalid overlay request
IE.BYT	-19	355	Odd byte count (or virtual address)
IE.BLK	-20	354	Logical block number too large
IE.NOD	-23	351	Caller's nodes exhausted
IE.ALN	-34	336	File already accessed on LUN
IE.NLN	-37	333	No file accessed on LUN
IE.BBE	-56	310	Bad block on device
IE.FHE	-59	305	Fatal hardware error on device
IE.OFL	-65	277	Device off line
IE.WCK	-86	252	Write check failure
IE.MII	-99	235	Media inserted incorrectly

APPENDIX C

CONFIGURATION TABLE VALUES

This appendix describes the values contained in the configuration table return buffer after a call to the P/OS system directive WIMP$. For a complete description of the WIMP$ system directive, see Section 8.88.

In this appendix, references to PC350 apply to the PC325. Also, unless specified otherwise, all values shown apply to both the PC325/350 and the PC380.

C.1 CONFIGURATION TABLE

Table C-1: Configuration Table Values

Byte Offset (Decimal)	Description
0	**Table length in bytes.** This word indicates the number of bytes in the table.
2	**Serial number ROM ID.** This value is the identification number of the serial number ROM part itself; it is not the contents of the serial number ROM. Normally this value is 31 (octal).
4	**High word of serial number.** If the checksum of the serial number ROM ID is correct, then this word contains the high 16-bits of the 48-bit identification number. Otherwise, this word contains the value zero.

Byte Offset (Decimal)	Description
6	**Middle word of serial number.** If the checksum of the serial number ROM ID is correct, then this word contains the middle 16-bits of the 48-bit identification number. Otherwise, this word contains the value zero.
8	**Low word of serial number.** If the checksum of the serial number ROM ID is correct, then this word contains the low 16-bits of the 48-bit identification number. Otherwise, this word contains the value zero.
10	**Number of option slots.** This word value represents the maximum number of option slots.
12	**Device section length in bytes.** This value is the byte length of the device section of the configuration table, excluding the device section length itself. Normally, this value is 170 (octal).
14-44	**Slot ID numbers and error/status values.** These items indicate what options are present in the Professional's option slots. For each slot, there is a 2 word entry. The first word contains the identification number of an option in the slot. The second word contains an error value in the low byte and a status value in the high byte. For a description of various ID numbers and error values that apply to the slot options, see Table C-3.
46-48	**Keyboard ID and error/status.** These word values apply to either the keyboard or another input device connected to the Professional's keyboard port. The first word contains the identification number of the input device. In the second word, the low byte contains the error value for the device; the high byte contains the status value. For a list of error codes, see Table C-3.

Byte Offset (Decimal)	Description
50-52	**Base processor ID and error/speed multiplier.** The first word contains the base processor identification number. For the J-11 processor (PC380), the ID is 35 (octal); for the F-11 processor (PC350), the ID is 11 (octal). In the second word, the low byte contains the error number for the CPU. The high byte contains a speed multiplier s, where: s = (J-11 time/F-11 time) - 1 The value s is a rounded integer whose value cannot be used for accurate timing. The value of the speed multiplier for the PC380 is 1; for the PC350 it is 0.
54-56	**Primary memory ID and error/size.** The first word contains the identification number for primary memory. Primary memory is the memory resident in the system module. It does not include memory connected through an option slot. This value is normally 33 (octal). In the second word, the low byte contains an error number for the primary memory; the high byte contains the total primary memory size in 32-kilobyte units.
58-60	**Base system ROM ID and error/operation mode.** The first word contains the identification number for the base system ROM. For the PC380, the ID of the base system ROM is 37 (octal); for the PC350, the ID is 26 or 27 (octal). In the second word, the low byte word contains an error number for the base system ROM, and the first bit of the high byte contains operation mode information: **Mode** — **Bit** Customer — 0 Console — 1 The remaining bits in the high byte are reserved.

Byte Offset (Decimal)	Description
	Customer mode is the default. In customer mode, the system sets the printer port to 4800 baud for use by a printer. In console mode, the system sets the printer port to 9600 baud for debugging with a BCC08 console cable and a second terminal. The Professional enters console mode automatically if the BCC08 cable is connected to the printer port at boot time.
62-64	**Video monitor ID and error/status.** The first word contains the video monitor identification number, if a video monitor is present. Normally this value is 32 (octal). If a monitor is not present, then the value of the first word is 0. The low byte of the second word contains the error value for the video monitor; the high byte contains the status value.
66-68	**Audio device ID and error/status.** These words are reserved for an audio device. The first word will contain the identification number of the audio device. The low byte of the second word will contain the error value for the audio device; the high byte will contain the status value.
70-72	**Keyboard interface ID and error/status.** The first word contains the identification number of the keyboard interface or other input device interface. This value is normally 14 (octal). The low byte of the second word contains the error value for the interface. The high byte contains the status value.
74-76	**Printer interface ID and error/status.** The first word contains the identification number of the printer interface. This value is normally 17 (octal). The low byte of the second word contains the error value for the interface; the high byte contains the status value. For a list of error codes, see Table C-3.

Byte Offset (Decimal)	Description
78-80	**Console interface ID and error/status.** The first word contains the identification number of the console interface, if it is present. If the console interface is not present, this value is 0. The console interface ID is normally 27 (octal). Note that both the console interface ID and the base system ROM ID can be 27 without any conflict. The low byte of the second word contains the error value for the interface; the high byte contains the status value.
82-84	**Communication interface ID and error/status.** The first word contains the identification number of the communication interface. This value is normally 21 (octal). The low byte of the second word contains the error value for the interface; the high byte contains the status value. For a list of error codes, see Table C-3.
86-88	**Time of day clock ID and error/status.** The first word contains the identification number of the clock. This value is normally 23 (octal). The low byte of the second word contains the error value for the clock. For the high byte, when the first bit is clear, the system correctly updated the time and date since the last power-up. When the first bit of the high byte is set, power to the clock was inconsistent, and the time and date may be invalid. For a list of error codes, see Table C-3.
90-92	**Nonvolatile RAM ID and error/status.** The first word contains the identification number of the nonvolatile RAM (NVR). This value is normally 24 (octal).

Byte Offset (Decimal)	Description
	The low byte of the second word contains the error value for the NVR. For the high byte, when the first bit is clear, the data in the NVR is valid since the last power-up. When the first bit of the high byte is set, the power to the NVR was inconsistent and its data may be invalid. For a list of error codes, see Table C-3.
94-96	**Floating point unit ID and error/status.** The first word contains the identification number of the floating point unit. For the PC380, this value is 10012 (octal); for the PC350, the value is 00012 (octal). The low byte of the second word contains the error value for the floating point unit; the high byte contains the status value. For a list of error codes, see Table C-3.
98-100	**Interrupt controller ID and error/status.** The first word contains the identification number of the interrupt controller. For the PC380, this value is 36 (octal); for the PC350, the value is 25 (octal). The low byte of the second word contains the error value for the floating point unit; the high byte contains the status value. For a list of error codes, see Table C-3.
102-132	**Reserved.**
134	**Soft restart address.** This value is an address in the I/O page to which software can transfer execution reboot the system without powering the system off and on, and without running the power-up self test. The device specified at offset 140 (decimal) of the configuration table is the boot device.
136	**Offset value into boot block.** This value is an offset value into the boot block that executes during system boot. This offset is useful in avoiding unwanted instructions, such as RESET, at the beginning of a bootblock. The system initializes this value to 0.

Byte Offset (Decimal)	Description
138	**Booted device ID.** This word is the identification number of the device that booted the system.
140	**Booted device unit number, slot, and type.** The low byte of this word contains the physical unit number of the device that booted the sytem. For the high byte, bits 0 through 4 contain the physical slot number of the boot device, and bits 5 through 7 contain device type information, as follows:

Device Type	Bits 7 6 5	
Removable media	0 0 0	
User specified	0 0 1	(PC380 only)
Local	0 1 0	(PC380 only)
Remote	0 1 1	(PC380 only)
Any	1 0 0	
Reserved	1 0 1	
Reserved	1 1 0	
Reserved	1 1 1	

142	**Current boot search return address.** This address in the I/O page allows software to return to the system's boot search state, in which the system searches for a boot device. This is useful in the event of a boot failure, or as a means of booting more than one operating system.
144	**Error flag for ROM diagnostics.** This flag indicates the result of the power-up self test. If the word value is 0, self test did not detect faults in the system. A nonzero value indicates that at least one fault was detected during self test.
146	**Additional information section length.** This value is the byte length of the additional information section of the configuration table, excluding the additional information length itself. The length of the additional information section in the PC380 is 14 (octal) bytes; in the PC350 it is 0.
148-158	**Display numbers 2 through 7.** The system displays the values contained in these words in the event of a software crash.

Byte Offset (Decimal)	Description
160	**Scratch memory starting address.** In the event of a software crash, this entry contains the starting 64-byte boundary of 32 kilobytes of scratch memory, for use by the base system ROM.
162	**Return PC.** This entry contains a virtual return address following a software crash. In the event of a software crash, the operating system can use the return PC to regain control.

C.2 DEVICE ID AND ERROR NUMBERS

Table C-2 summarizes the device codes for the Professional. Unused numbers are omitted.

Table C-2: Summary of Device Codes

ID (Octal)	Device
000001	LK200 keyboard
000011	PC350 base processor (F-11)
000012	PC350 floating point processor (FPP)
000013	Reserved
000014	LK200 keyboard interface
000015	Reserved
000016	Reserved
000017	Printer port
000020	Reserved
000021	Communication port

ID (Octal)	Device
000023	Time of day clock
000024	Nonvolatile RAM
000025	PC350 interrupt controller
000026	PC350 base system ROM V1.0 (first release)
000027	Maintenance console port or base system ROM V2.0 (IVIS)
000030	Option present register
000031	Serial number ROM
000032	Monitor attachment
000033	Primary memory
000034	Option RAM (256KB extended memory, MSC11-CK)
000035	PC380 base processor (J-11)
000036	PC380 interrupt controller
000037	PC380 base system ROM V1.0 (first release)
000041	Telephone Management System (TMS)
000042	Ethernet controller
000043	Z80/CPM softcard
000045	PC350 IVIS base module set
000046	IDLDR IEEE option (real-time interface)
000047	KANJI font module
000050	PC380 bitmap video base module
000060	DECtouch module (DTM)
000061	MS-DOS board
000064	Quad Serial Line Unit (SLU)
000075	PC350 IVIS controller

ID (Octal)	Device
000401	RD-Series 5 1/4" Winchester disk controller
001002	PC350 bitmap video base module
001403	PC350 bitmap video extension
002004	RX50 5 1/4" floppy disk controller
003006	PC350 IVIS system module (no ROM)
010012	PC380 floating point processor (FPP)
010050	PC380 bitmap video extension
011002	PC350 IVIS bitmap base module
011403	PC350 IVIS bitmap video extension
040001	Tempest shielded keyboard

Table C-3 describes various device ID and error values that can appear in the configuration table. Errors are listed in order of ascending device ID code.

Note that an error value of 0 for any device indicates that the device is functioning properly.

Table C-3: Error Values for Devices

ID (Octal Word)	Error Code (Octal Byte)	Description
Any	-2	The system located a device in the option slot but determined (by reading the device's ROM) that it is not functioning properly. The system error display will contain the three-digit octal error 376.

DEVICE ID AND ERROR NUMBERS

ID (Octal Word)	Error Code (Octal Byte)	Description
Any except 0	-4	The system located a device in the option slot but determined that it is not functioning properly. This error indicates that, although the device is present, it is not registered in the option present register. The system error display for this error will contain the three digit octal error 374.
Any except 0, -2	-1 (word value)	The system located a device in the option slot but determined that it is an untested device.
-2	-3	The system located a device in the option slot but determined that it is not functioning properly. This error indicates that the device ID remained at zero for too long a period of time. You should replace the board. The system error display for this error will contain the three digit octal error 375.
-2	-1	The system located a device in the option slot but determined that it is not functioning properly. This error indicates that a timeout trap occurred while the system attempted to read the ID. If tightening the cables does not eliminate the error, replace the board. The system error display for this error will contain the three digit octal error 377.
0	-1 (word value)	No device is present in the option slot. (Not an error.)
1	60	The keyboard is not functioning properly. You should replace it.

ID (Octal Word)	Error Code (Octal Byte)	Description
1	75	The keyboard is not functioning properly--a key is stuck. The configuration status byte for the keyboard contains the code for the key that is stuck. Note that the maintenance services keyboard test also reports the code for the stuck key. If you cannot loosen the key, you must replace the keyboard.
12 (PC350)	27 through 50	The PC350 floating point unit is not functioning properly. The system detected errors in certain floating-point instructions. You should replace the floating point unit. Refer to ID 10012 for the PC380 floating point unit.
14	4	The keyboard interface is not functioning properly. The UART chip (2661) did not respond within the allotted time. You should replace the system module.
14	6	The keyboard interface is not functioning properly. A status error occurred. You should replace the system module.
14	10	The keyboard interface is not functioning properly. A data compare error occurred. You should replace the system module.
14	26	The keyboard interface is not functioning properly. The system could not locate the UART chip (2661), a nonexistent memory trap occurred. You should replace the system module.
17	3	The printer interface is not functioning properly. The system could not locate the UART chip (2661), a nonexistent memory trap occurred. You should replace the system module.

ID (Octal Word)	Error Code (Octal Byte)	Description
17	5	The printer interface is not functioning properly. The UART chip (2661) did not respond within the allotted time. You should replace the system module.
17	7	The printer interface is not functioning properly. A status error occurred. You should replace the system module.
17	11	The printer interface is not functioning properly. A data compare error occurred. You should replace the system module.
21	12	The serial communication interface is not functioning properly. A data response error occurred. You should replace the system module.
21	13	The serial communication interface is not functioning properly. The USART chip (7201) did not respond. You should replace the system module.
23	20	The time-of-day device is not functioning properly. It did not interrupt. You should replace the system module.
23	21	The time-of-day device is not functioning properly. It did not interrupt within the proper time frame. You should replace the system module.
23	25	The time-of-day device is not functioning properly. It did not respond; a nonexistent memory trap occurred. You should replace the system module.

ID (Octal Word)	Error Code (Octal Byte)	Description
24	22	The Nonvolatile RAM is not functioning properly. It did not respond. You should replace the system module. **NOTE** ID 24, error 22 can in unusual cases result from a floating point error as well.
24	23	The Nonvolatile RAM is not functioning properly. A data compare error occurred. You should replace the system module.
25	1	The interrupt controller is not functioning properly. Either all of the channels did not interrupt, or there was a nonexistent memory (NXM) trap when addressing the controllers. You should replace the system module.
25	2	The interrupt controller is not functioning properly. The system detected multiple interrupts when only one was allowed. You should replace the system module.
34	Any	There is a problem with one or more memory modules.
41	Any	There is a problem with the TMS controller.
42	112 or less	There is a problem with the DECNA module.
43	Any	There is a problem with the CP/M module.
46	1	There is a problem with the Real-Time Interface (RTI) module, serial line unit 1.

ID (Octal Word)	Error Code (Octal Byte)	Description
46	2	There is a problem with the Real-Time Interface (RTI) module, serial line unit 2.
50	1	The PC380 bit-map video controller is not functioning properly. The "transfer done" bit in the CSR failed.
50	2	The PC380 bit-map video controller is not functioning properly. A register initialization failure occurred.
50	3	The PC380 bit-map video controller is not functioning properly. A plane 1 memory failure occurred.
50	4	The PC380 bit-map video controller is not functioning properly. A vertical retrace failure occurred.
50	5	The PC380 bit-map video controller is not functioning properly. A counter register failure occurred.
50	6	The PC380 bit-map video controller is not functioning properly. An X, Y pattern, and/or plane 1 control register failure occurred.
50	7	The PC380 bit-map video controller is not functioning properly. A scroll register failure occurred.
50	10	The PC380 EBO is not functioning properly. The bit-map video detected the presence of the EBO board, but the system could not find the board in the slot.
50	103	The PC380 EBO is not functioning properly. A plane 2 memory failure occurred.
50	106	The PC380 EBO is not functioning properly. A plane 2 control failure occurred.

ID (Octal Word)	Error Code (Octal Byte)	Description
50	107	The PC380 EBO is not functioning properly. A plane 2 scroll register failure occurred.
50	203	The PC380 EBO is not functioning properly. A plane 3 memory failure occurred.
50	206	The PC380 EBO is not functioning properly. A plane 3 control failure occurred.
50	207	The PC380 EBO is not functioning properly. A plane 3 scroll register failure occurred.
401	1	The hard disk drive is not functioning properly. The system could not locate the "operation ended" bit.
401	2	The hard disk drive is not functioning properly. An internal power-up error occurred.
401	3	The hard disk drive is not functioning properly. A sector, cylinder, or head select register failed.
401	4	The hard disk drive is not functioning properly. The busy bit did not go away.
401	5	The hard disk drive is not functioning properly. Either the drive is not ready or a seek is not yet complete.
401	6	The hard disk drive is not functioning properly. The RESTORE command did not cause an "A" interrupt.
401	7	The hard disk drive is not functioning properly. The RESTORE command did not cause the operation to be ended.
401	10	The hard disk drive is not functioning properly. The RESTORE command caused the error bit to be set.

ID (Octal Word)	Error Code (Octal Byte)	Description
401	30	The hard disk drive is not functioning properly. The ID was not found.
401	31	The hard disk drive is not functioning properly. An ID field CRC error occurred.
401	32	The hard disk drive is not functioning properly. A data field CRC error occurred.
401	33	The hard disk drive is not functioning properly. An unexpected operation end interrupt occurred.
401	34	The hard disk drive is not functioning properly. An invalid operation end interrupt occurred.
401	35	The hard disk drive is not functioning properly. An unexpected DRQ interrupt occurred.
401	36	The hard disk drive is not functioning properly. An invalid DRQ interrupt occurred.
401	37	The hard disk drive is not functioning properly. A restore command time out occurred.
1002	1	The PC350 bit-map video controller is not functioning properly The "transfer done" bit in the CSR failed.
1002	2	The PC350 bit-map video controller is not functioning properly. A register initialization failure occurred.
1002	3	The PC350 bit-map video controller is not functioning properly. A plane 1 memory failure occurred.
1002	4	The PC350 bit-map video controller is not functioning properly. A vertical retrace failure occurred.

DEVICE ID AND ERROR NUMBERS

ID (Octal Word)	Error Code (Octal Byte)	Description
1002	5	The PC350 bit-map video controller is not functioning properly. A counter register failure occurred.
1002	6	The PC350 bit-map video controller is not functioning properly. An X, Y pattern, and/or plane 1 control register failure occurred.
1002	7	The PC350 bit-map video controller is not functioning properly. A plane 1 scroll register failure occurred.
1002	10	The PC350 EBO is not functioning properly. The bit-map video detected the presence of the EBO board, but the system could not find the board in the slot.
1403	1	The PC350 extended bit-map video controller is not functioning properly. There is a failure in the cable connection.
1403	2	The PC350 extended bit-map video controller is not functioning properly. Improper register initialization occurred.
1403	3	The PC350 extended bit-map video controller is not functioning properly. A plane 2 memory failure occurred.
1403	4	The PC350 extended bit-map video controller is not functioning properly. A plane 2 control register failure occurred.
1403	5	The PC350 extended bit-map video controller is not functioning properly. A plane 3 memory failure occurred.

ID (Octal Word)	Error Code (Octal Byte)	Description
2004	270	The RX50 module is not functioning properly. The system attempted to access a nonspecified sector number.
2004	300	The RX50 module is not functioning properly. The lower nibble of RAM failed to pass the memory test.
2004	310	The RX50 module is not functioning properly. The higher nibble of RAM failed to pass the memory test.
2004	320	The RX50 module is not functioning properly. No index pulse was detected.
2004	330	The RX50 module is not functioning properly. The drive speed is not within the limit.
2004	340	The RX50 module is not functioning properly. The system detected a bad format or a blank diskette.
2004	350	The RX50 module is not functioning properly. A stepping error occurred.
2004	354	The RX50 module is not functioning properly. An attempt was made to set unsupported disk parameters.
2004	360	The RX50 module is not functioning properly. The Phase Lock Loop (PLL) frequency is not within the limits.
2004	364	The RX50 module is not functioning properly. An attempt was made to read a sector containing a deleted data mark.
2004	370	The RX50 module is not functioning properly. A data buffer is bad.
2004	210	The RX50 module is not functioning properly. An attempt was made to write to a nonRX50 formatted diskette.

ID (Octal Word)	Error Code (Octal Byte)	Description
10012 (PC380)	27 through 50	The PC380 floating point unit is not functioning properly. The system detected errors in certain floating-point instructions. You should replace the floating point unit. Refer to ID 12 for the PC350 floating point unit.
10050	1	The PC380 extended bit-map video controller is not functioning properly. There is a failure in the cable connection.
10050	2	The PC380 extended bit-map video controller is not functioning properly. Improper register initialization occurred.
10050	3	The PC380 extended bit-map video controller is not functioning properly. A plane 2 memory failure occurred.
10050	4	The PC380 extended bit-map video controller is not functioning properly. A plane 2 control register failure occurred.
10050	5	The PC380 extended bit-map video controller is not functioning properly. A plane 3 memory failure occurred.
10050	6	The PC380 extended bit-map video controller is not functioning properly. A plane 3 control register failure occurred.
10050	7	The PC380 EBO is not functioning properly. A plane 1 scroll register failure occurred.
10050	103	The PC380 EBO is not functioning properly. A plane 2 memory failure occurred.

APPENDIX D

DIRECTIVE IDENTIFICATION CODES

Directive Identification Codes (DICs) are used to identify each directive. The DIC appears in the low byte of the first (or only) word in the Directive Parameter Block (DPB). The DPB length (in words) appears in the high byte of the first DPB word. Thus, both bytes make up the word format shown below:

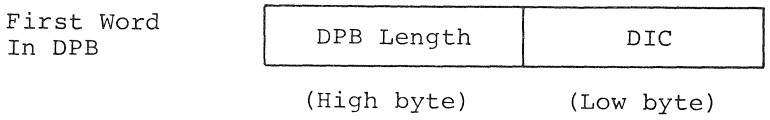
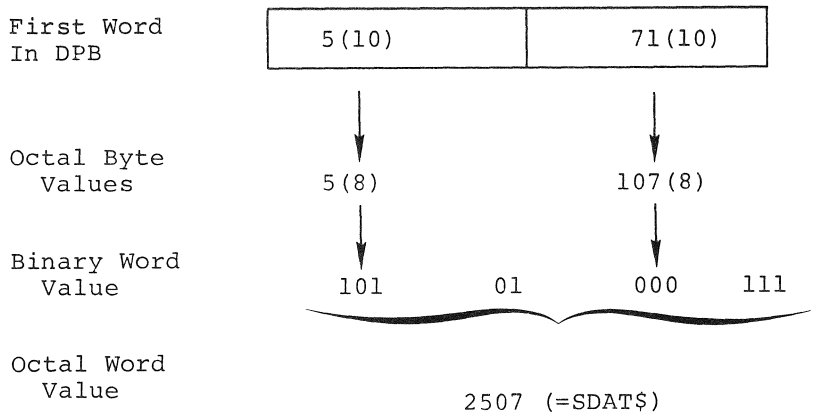


Table D-1 contains a listing of directives arranged in numerical sequence, according to the octal value for the first DPB word. In addition, the DIC and DPB lengths are included as decimal values as they appear in Chapter 8.

This list can be used as a software debugging aid to quickly identify directives based on the octal value of the first word in a DPB. An example for the SDAT$ directive is provided below, to illustrate how the octal value is obtained:



DIRECTIVE IDENTIFICATION CODES

Octal Value for DPB First Word	Directive (Macro Call)	DIC (Decimal)	Values for DPB Length (Decimal)
1457	RSUM$	47	3
1475	STIM$	61	3
1523	ABRT$	83	3
1531	EXTK$	89	3
1547	SVDB$	103	3
1551	SVTK$	105	3
1605	USTP$	133	3
1611	STLO$	137	3
1617	CNCT$	143	3
1647	SREX$	167	3
1657	SWST$	175	3
1715	CPCR$	205	3
2007	ALUN$	7	4
2011	ALTP$	9	4
2101	GPRT$/GREG$	65	4
2113	RCVD$	75	4
2115	RCVX$	77	4
2213	RCST$	139	4
2223	EMST$	147	4
2427	MRKT$	23	5
2505	SREF$	69	5
2507	SDAT$	71	5

DIRECTIVE IDENTIFICATION CODES

Octal Value for DPB First Word	Directive (Macro Call)	DIC (Decimal)	Values for DPB Length (Decimal)
2625	CRVT$	149	5
2717	DLOG$	207	5
2717	ACHN$	207	5
2717	SDIR$	207	5
3113	VRCD$	75	6
3115	VRCX$	77	6
3213	VRCS$	139	6
3317	GDIR$	207	6
3413	RQST$	11	7
3601	CINT$	129	7
3615	SDRC$	141	7
3717	CLOG$	207	7
3717	FSS$	207	7
4107	VSDA$	71	8
4215	VSRC$	141	8
4615	SDRP$	141	9
4717	TLOG$	207	9
5421	RUN$	17	11
6001	QIO$	1	12
6003	QIOW$	3	12
6413	SPWN$	11	13
6717	PFCS$	207	13

INDEX

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Please cut along this line.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify) _____

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code _____
or
Country

-- Do Not Tear - Fold Here and Tape ----------

digital

No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

Professional Workstations Publications
DIGITAL EQUIPMENT CORPORATION
146 Main Street, MLO21-2/T76
Maynard, Massachusetts 01754-2571

--- Do Not Tear - Fold Here ----------

Cut Along Dotted Line