WICAT Multi-user Control System

WMCS

Programmer's Reference Manual

188-190-305 C

May 1985

• Software •
Publications

WICATsystems

Copyright © 1983 by WICAT Systems Incorporated
All Rights Reserved
Printed in the United States of America

Receipt of this manual must not be construed as any kind of commitment, on the part of WICAT Systems Incorporated, regarding delivery or ownership of items manufactured by WICAT.

This manual is subject to change without notice.

first printing	August 1983
second printing	April 1984
third printing	May 1985

Typographical Conventions Used in this Publication

Bold facing indicates what you should type.

Square brackets, [], indicate a function key, the name of which appears in uppercase within the brackets. For example, [RETRN], [CTRL], etc.

Underlining is used for emphasis.

Information about this Manual

Review the following items before you read this publication:

1. WMCS Introductory User's Manual
2. WMCS User's Reference Manual

The subject of this manual

WMCS system calls and the Keyed Sequential Access Method (KSAM) are described for the system programmer's ongoing use of the WMCS operating system.

The audience for whom this publication was written

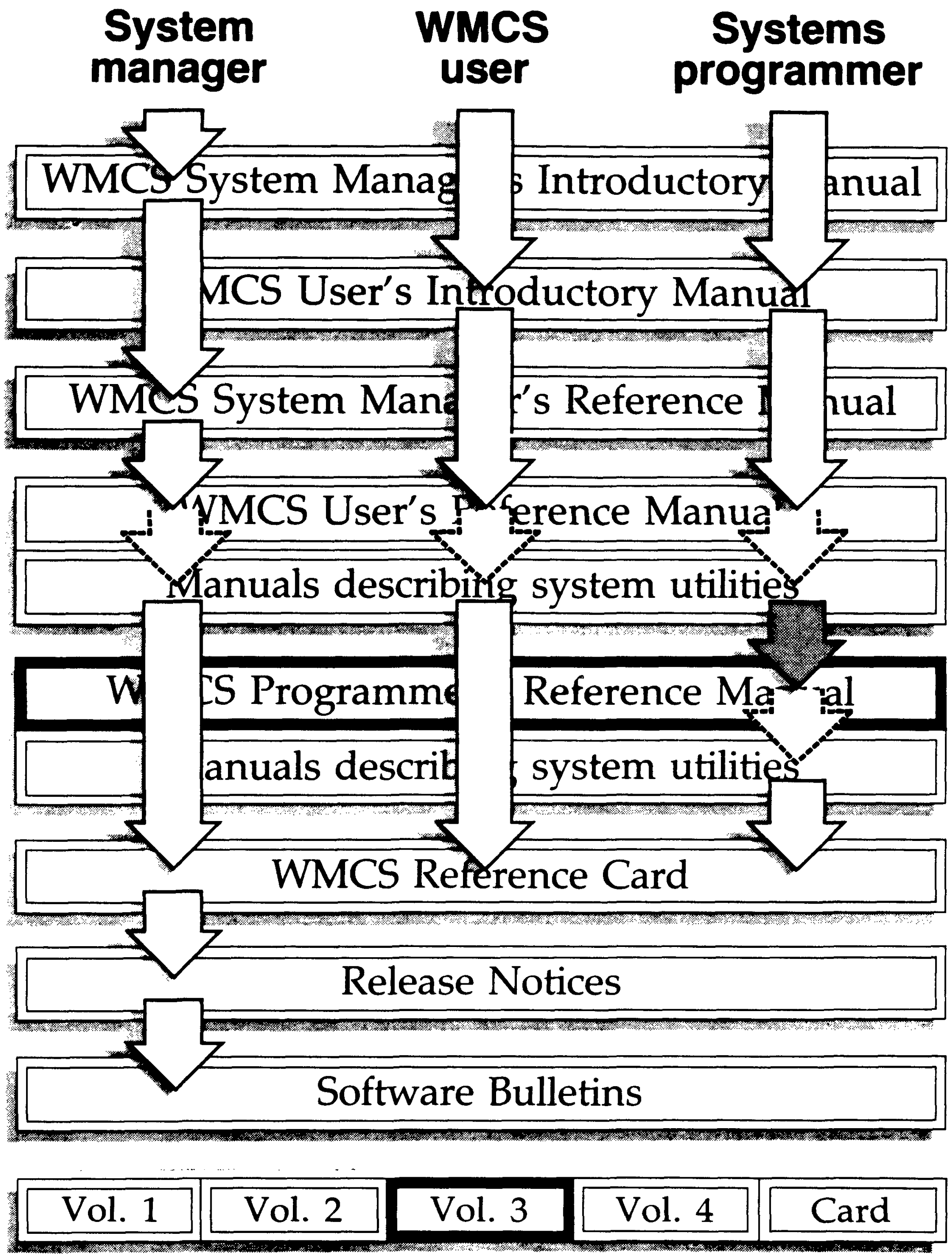
Programmers who understand programming fundamentals and who have read the WMCS Introductory User's Manual and the WMCS User's Reference Manual.

Related publications

The chart on the following page lists other publications about the WMCS and the order in which they should be read.

Reader's Guide to WMCS Publications

Instructions: Determine the audience to which you belong and then read *only* the publications at an *arrowhead.* Dotted arrowheads indicate optional reading.



System manager
WMCS user
Systems programmer
WMCS System Manager's Introductory Manual
WMCS User's Introductory Manual
WMCS System Manager's Reference Manual
WMCS User's Reference Manual
Manuals describing system utilities
WMCS Programmer's Reference Manual
Manuals describing system utilities
WMCS Reference Card
Release Notices
Software Bulletins
Vol. 1
Vol. 2
Vol. 3
Vol. 4
Card

Table of Contents

CHAPTER 1

INTRODUCTION

The Multiuser Control System (MCS) is a general purpose, interactive, multi-user operating system developed by WICAT Systems, Inc. for it's family of MC68000-based computer systems. The MCS makes available, on a microcomputer, features previously available only on large mini, and mid-sized computer systems.

1.1 FEATURES OF THE MCS

The operating system is divided logically into two parts:

1. **The scheduler.**

   Based upon the priority and status assigned to each process, or task, the scheduler gives each process a share of the processor resource.

2. **System service calls.**

   System service calls are executed only as they are called for (explicitly) by a process. They are therefore considered extensions of the process. When a process calls the operating system, the process continues its execution within the MCS; it is as though the system service calls were a set of reentrant, callable subroutines. Hence, the system calls are not an overhead function like the scheduler (that is not part of any process and does not contribute to the accomplishment of any user task).

These are the major features of the MCS:

1. System configuration does not require a complicated system generation procedure. For example, device drivers can be added and removed using the _MOUNT and _DISMNT system calls.

2. The amount of available memory is the only limitation on the number of files that can be open simultaneously, the number of processes that can be active simultaneously, the number of devices that can be concurrently mounted, etc.

3. A prioritized scheduling algorithm.

4. The text portion of processes is automatically shared by multiple invocations of the same image file.

5. Memory can be allocated and deallocated dynamically. Each process has its own address space. The MCS address space is protected from all processes, and processes are protected from one another by the hardware memory management.

6. A hierarchichal file structure.

7. Disk devices support a user-definable disk cache with read-ahead capability.

8. Multiple versions of files.

9. Logical I/O, i.e., disk files and devices are accessed uniformly.

10. Logical names are fully integrated into the MCS.

11. A multi-keyed (Keyed Sequential Access Method, KSAM) file access program is provided in addition to standard random and sequential access methods.

12. Interprocess communication includes named pipes, mail, shared memory, and event flags.

13. General purpose record locking.

14. User-assignable, interactive terminal characteristics. The standard XONXOFF protocol is supported, and reads from the terminal can use any of several edit modes including raw data and line reads. Time outs are supported so that processes do not hang while waiting for input.

Chapter 2

Directory of WMCS System Calls

This chapter lists the WMCS system calls by function. For a complete alphabetical listing of these system calls, see Appendix A in this manual.

Process Creation

This set of system calls provides the mechanism for process creation and termination. There are two forms of process creation under the WMCS:

1. Forking. The child process is executed parallel to the parent process.
2. Spawning. The parent process hibernates until the execution of the child process is complete.

Several parameters can be specified during the creation of a process, e.g., the scheduling priority for the new process; its standard input, output, and error files; a name and a command line.

_CLONE	Make a duplicate of an existing process.
_CRPRCS	This simplified version of the create process system call assumes default values for many of the parameters associated with the _CRPROC System Call.
_CRPROC	Create a new process. This is the standard system call for process creation.
_EXPROC	Terminates, i.e., removes from the system, the specified process. Any open files are closed automatically and all memory assigned to the process is made available to the system.

_MULCRPS	Allows the creation of multiple copies of a process by means of a single image file. This is useful in quickly bringing up a single application on several terminals simultaneously.

Process Control

These system calls are used to manage the attributes of processes executing in the system. Note that privileges are required before a process can affect processes that do not have the same user id code (UIC). Also, privileges are required to change a processes priority, timeslice, operating mode or privileges.

_ALARM	Sets or resets a timer so that, if the specified time is reached during the life of the process, the process is terminated.
_CHSUPER	Change to supervisor mode. If the calling process has the correct privilege, its processing mode is changed to supervisor. This allows the process to execute privileged instructions and to access memory outside of its logical address space. After successful execution of this system call, the process has virtually unlimited access to the system.
_CHUSER	Change the processing mode of the calling process to user (this is the inverse of _CHSUPER).
_CRSHDP	Use this system call to enable or inhibit the crash display (stack dump) which normally appears when a process performs an illegal operation.
_CTRLC	Enables or disables the use of [CTRL] c to terminate the process.
_EXITRTN	This system call can be used in place of _SETEXIT to define an exit handler. If the process uses _SETEXIT to define an exit handler, it must use an RTR or RTE instruction to return from the exit handler. With _EXITRTN the process can return from the exit handler with the standard subroutine return statement, RTS. This allows processes written in high level languages to define exit handlers from which they can return.
_GENGY	Returns, to the calling process, the PID of the specified ancestor process.

_GETATTR	Returns the current process attributes.
_GETEXIT	Returns, to the calling process, the address of the current exit handler.
_GETPCB	Returns, to the calling process, the Process Control Block (PCB) of the specified process.
_GETPID	Returns the Process Identification number (PID) of the process whose name is specified as part of this system call.
_GETPNAM	Returns the name of the process assigned to the PID specified as part of this system call.
_GETPRI	Returns the priority level of the process assigned to the PID specified as part of this system call.
_GETPRV	Returns the privilege mask assigned to the process whose PID is specified as part of this system call.
_GETTMSL	Returns the timeslice assigned to the process whose PID is specified as part of this system call. The timeslice is the maximum amount of time a process is allowed to run before it is interrupted so that another process can run.
_HIBERN	Suspends the specified process. Use WAKE to cause the suspended process to resume. Note that a suspended process cannot wake itself.
_ORIGPRV	This system call returns the privilege a process has, not including any privileges with which it may have been installed.
_PIDLST	Returns the PIDs of all processes on all priorities and the total number of processes running on your machine.
_PRCLST	Returns the PIDs of those processes assigned to the priority level designated as part of this system call.
_PRIRAT	Assigns the scheduling ratios for each priority level. The scheduling ratio determines the number of processes at a particular priority level that will be executed for each process at the next lower level.
_SETATTR	Set process attributes.
_SETEXIT	Allows a process to specify the execution of a procedure or subroutine before the termination of the calling process. This is particularly useful in recovering from errors.

_SETPNAM	Changes the name of the process assigned to the PID specified as part of this system call.
_SETPRI	Changes the priority level of the specified process.
_SETPRV	Changes the privileges assigned to the specified process.
_SETRTM	Immunizes the specified process from interruptions by the scheduler, i.e., with the real-time mode flag set, the process runs until it either relinquishes the CPU, or is blocked due to input or output not being received in time.
_SETTMSL	Adjusts the timeslice associated with the specified process. The timeslice is the maximum amount of time a process is allowed to run before being interrupted so that another process can run.
_SETTRP	Allows a user process to take advantage of the MC68000 trap instructions, and to handle certain hardware exceptions, e.g., a divide-by-zero.
_WAIT	Suspends the designated process for a specified period.
_WAKE	Wakes a hibernated process.
_WAKEC	Decrements the hibernate count of the specified process. If the hibernate count goes to zero, the specified process is awakened. Contrast this with _WAKE.

File System

One of the major functions of the file system is to insulate user processes from the details of physically accessing I/O devices. It is also advantageous if a program can read from and write to terminals, printers, etc., using those system calls used to access files on a disk.

These capabilities are referred to as device-independent I/O, or logical I/O. In other words, this allows the program to manipulate files without having to consider most of the particular characteristics of the device on which the file is located. The WMCS provides logical I/O for reading and writing devices, files, as well as named pipes (interprocess).

_CHDIR	Designates the working-default device and directory for the calling process.

_CLOSE	Closes the specified file, i.e., makes the file inaccessible to read and write operations.
_CREATE	Creates a file and assigns it the attributes specified as part of this system call, then opens the file.
_CREATES	This simplified version of _CREATE creates a file and assigns it default values for many of the _CREATE parameters.
_DELETE	Deletes the specified file.
_DUPLUN	Copy the LUN from _OPEN or _CREATE (similar to re-opening).
_FRDWAIT	Waits for the completion of a fast read. A fast read means that one or more sectors are read directly into the logical address space assigned to the process (bypassing the disk cache). Inasmuch as this happens asynchronously, this system call allows the calling process to verify that the data are available before the data are accessed.
_GETDIR	Returns a string containing the name of the default device and the name of the default directory for the calling process.
_GETFCB	Returns the FCB of a file opened by the calling process.
_GETFID	Returns the file ID, to the calling process, of the specified process.
_GETFNAM	A process can use this system call to determine the name of an open file.
_GETFRSZ	Returns the record size of an open file.
_GETPOS	Returns the relative record position (relative to the front of the file) of the next record in the file to be read or written.
_LOCK	Allows the process to lock records within a specified file to be used exclusively by the calling process.
_OPEN	Makes the specified file accessible to read and/or write operations. A file can be opened for read and/or write operations and (optionally) for exclusive access by the calling person. Files can also be shared.

_READ	Reads records from an open file into the specified buffer.
_RENAME	Changes the name of the specified file.
_SETFCB	Allows the calling process to modify the File Control Block of an open file.
_SETFID	Allows the calling process to specify the file ID of an open file.
_SETFRSZ	Allows the calling process to change the record size of an open file.
_SETPOS	Allows the process to position a file. This is not required for random access to files. _READ and _WRITE allow the specification of those records to be transferred.
_UNLOCK	Unlocks records in an open file.
_WRITE	Writes records from the specified buffer to the file.

Device Control

The following set of system calls allow a process to mount, dismount, access and set attributes on devices.

_ALLOC	Use this system call to allocate or reserve a device for the exclusive use of a process and its subprocesses.
_DEALLOC	Deallocate a device that was allocated using the _ALLOC system call.
_DISMNT	Removes the device from the cognizance of the WMCS.
_FLUSH	Flushes I/O buffers to the device. Any modified sectors or FCBs are written to the device.
_GETALC	This system call allows a process to retrieve the names of devices allocated to a specified process.
_GETDNAM	Returns the name of the nth device in the list of mounted devices.
_GETDST	Returns the device table and device status block for the specified device.

_GETREL	Given the name of a rotored device, this system call will retrieve the names of all devices assigned to that rotor.
_GETRTR	This system call allows a process to retrieve the names of all currently defined rotor lists.
_GIODST	This system call is the same as _GETDST except that it requires the logical unit number (lun) of an open file on the device instead of the device name. This is a more efficient mechanism than _GETDST.
_MEMMNT	Makes a device or pipe known to the file system and, if necessary, loads the device driver from a specified location in system memory.
_MOUNT	Makes a device or pipe known to the file system and, if necessary, loads the device driver from the specified file.
_PHYSIO	Allows the process to perform physical I/O operations on the device, e.g., reading and writing sectors.
_PHYSOP	Allows the process to perform physical I/O operations on the device, e.g., reading and writing sectors.
_SETDST	Allows the process to update the device status block of the specified device, and is used to establish such device characteristics as baud rate.
_SETRTR	Defines a rotor list and assigns the names of the devices that are associated with the rotor.
_SIODST	Similar to _SETDST except that the device whose status is to be set is specified by a logical unit number of an open file on the device instead of the device name. This is more efficient than _SETDST. Allows the process to update the device status block of the specified device, and is used to establish such device characteristics as baud rate.
_SKIP	Allows the process to position the tape at the beginning or end of the tape, or to skip files.

KSAM

The Keyed Sequential Access Method (KSAM) is an optional WMCS module that can be included in the system's configuration when the system is booted. It allows files to be accessed on the basis of key values, an access method that enhances the standard random access provided by the file system. Its more important features include:

Multi-user access to KSAM files
Record-level locking
Deadlock detection
Multiple keys
Segmented keys

Each KSAM file is maintained as two separate files; one containing the keys and the second containing the data.

Keys can be composed of any number of signed or unsigned bytes, words, or longwords (up to a maximum of 255 bytes).

Programs can find a record containing a specified key value, or read a file in ascending or descending order for any key.

"KSAM file" in the following list of KSAM system calls refers to the KSAM key file and the KSAM data file.

_KCLALL	Closes all KSAM files opened by the calling process.
_KCLOSE	Closes a KSAM file.
_KCREAT	Creates a KSAM file. This includes a definition of all the key fields in the records constituting the new file.
_KDELET	Deletes the current record from the KSAM file.
_KFIND	Finds the record that contains the specified key value.
_KFLUSH	Writes all modified KSAM buffers to the disk.
_KINFO	Returns information about an open KSAM file.
_KMOVFB	Positions (logically) the KSAM file at its beginning or end, according to the specified key.
_KOPEN	Opens an existing KSAM file for access.
_KREAD	Retrieves a record from an open KSAM file.

_KUNLCK	Unlocks the specified KSAM records.
_KUPDAT	Replaces a KSAM record, in an open KSAM file, with the specified record.
_KWRITE	Adds a new record to an open KSAM file.

Memory Control

The following system calls allow the process to manage the system's memory.

_ALLMEM	Adds a new page of memory to the process, or allows the process to share a page of memory with another process.
_DEFMEM	Define a named shareable memory segment. Once a named memory segment has been defined, other processes may request that that segment be mapped into their address space.
_FREMEM	Deallocates a page of memory, i.e., returns the page to the system's pool of available memory.
_GETFRE	Assigns the amount of available memory to the calling process.
_GETMLST	This system call allows a process to retrieve the names of currently defined named shared memory segments.
_MAPPHYS	This system call allows a process to map any physical segment of memory into its logical address space.
_PROTMEM	Sets or clears the write-protect flag on a page of logical memory.
_RDPMEM	Allows the process to read the values stored in the specified locations in physical memory.
_SHRMEM	Maps the specified named shareable memory segment to the logical address space of the calling process.
_UDEFMEM	Removes the definition for the specified named shareable memory segment from the operating system. This is the inverse of _DEFMEM.
_USHRMEM	Removes the memory associated with the specified named shareable memory segment from the logical address space of the calling process. This is the inverse of _SHRMEM

_WTPMEM	Allows the process to write values to the specified locations in physical memory.

Logical Names

The following system calls allow processes to assign, deassign, and retrieve logical names. A logical name is a string equivalence.

_ASSIGN	Assigns a logical name in the logical name table for the specified process.
_GASSIGN	Assigns a logical name in the system's logical name table.
_GETGLB	Allows the process to retrieve the nth logical name from the system's logical name table.
_GETLOG	Allows the process to retrieve the nth logical name from the logical name table for the specified process.
_TRANPID	Returns the Equivalence assigned to the specified Name. Note that this is similar to _TRANS except that this system call uses the logical name table of a specified process and its parents, instead of the logical name table of the calling process.
_TRANS	Returns the Equivalence assigned to the specified Name. First, the logical name table for the calling process is searched. If no Equivalence is found, the logical name table for the parent of the calling process is searched. The search continues until an Equivalence is found, or until there are no more parent processes. At that time, the system's logical name table is searched. If no Equivalence is found, the original string is returned as the translation.

Ownership

The following system calls are used to find out or specify the ownership of files, devices, or processes (all files, devices, and processes have an owner). Ownership is determined by a User Identification Code, or UIC. The UIC is composed of an owner ID and a group ID.

_DEFDUIC	Establishes the default device ownership. Whenever the device is not being referenced by any process the user identification code (UIC) of the device is set to this value.
_GETDUIC	Returns the UIC for the specified device.
_GETFUIC	Returns the UIC for the specified file.
_GETUIC	Returns the UIC for the specified process.
_SETDUIC	Assigns a UIC to the specified device (this changes the ownership of the file).
_SETFUIC	Set file UIC. This changes the ownership of the file.
_SETMUIC	Assigns a UIC to the specified named memory segment.
_SETUIC	Assigns a UIC to the specified process (this changes the ownership of the specified process).

Protection

The following system calls are used to find out or assign device and file protection. Protection is actually a matter of the access privileges granted (to a process) by the owner of the device or file. Processes fall into four categories, based on the owner of the process and the process's privilege mask:

1. A process created by the owner of the file or device.
2. Processes created by members of the same group to which the owner belongs.
3. Processes with SYSTEM privilege.
4. All other processes, i.e., the Public.

_DEFDPRT	Establishes the default protection to be applied to a device. Whenever the device is not being referenced by any process, the protection mask will be set to this value.
_DEFPROT	Establishes the default protection to be assigned to files or devices created, by the specified process, when protection is not specified.

_GETDPRT	Returns the protection flag word associated with the specified device.
_GETFPRT	Returns the protection flag word associated with the specified file.
_GETPROT	Returns a default protection mask associated with the calling process.
_SETDPRT	Assigns the specified value as the protection flag word for the designated device.
_SETFPRT	Assigns the specified value as the protection flag word for the designated file.
_SETMPRT	Assigns the specified value as the protection flag word for the designated memory segment.

Interprocess Communication

These system calls signal events and send messages between cooperating processes.

_ANDEVNT	Waits for the logical AND of event flags. The calling process is suspended until all of the specified bits are set in the event flag word of the specified process.
_CLREVNT	Clears the specified bits in the event flag word of specified process.
_GETEVNT	Transfers the event flag word of the specified process to the calling process.
_GMAIL	Returns a message (sent to the specified process) to the calling process.
_OREVNT	Waits for the logical OR of event flags. The calling process is suspended until any of the specified bits are set in the event flag word of the specified process.
_SETEVNT	Sets bits in the event flag word of the specified process.
_SMAIL	Allows the calling process to send mail to another process.

Installed Files

An installed file is an image file that must execute with more privileges than the parent process may have. In other words, an installed process executes with privileges that the user running the process does not possess.

Furthermore, a device driver can be installed, meaning that a process with no privileges can mount a device using that driver.

The following system calls allow processes to install and remove privileged files.

_DEINST	Removes a privileged file from the list of installed files.
_GETINST	Retrieves the definition of the nth installed file.
_INSTALL	Installs the specified privileged file.

Information

These system calls are used to set the system clock and to get the system's time of day, the system's tick clock, and the WMCS version banner.

_ERRNO	The WMCS will pass control to the exit handler of a process when the process is about to be terminated. _ERRNO is used to determine the cause of the termination, or the abort reason code.
_GETTIC	Returns the value of the system's tick clock, which shows how much time has elapsed since the system was booted. The time is expressed as the number of .01 seconds that have elapsed.
_GETTIM	Returns the date and time according to the system's time-of-day clock.
_SETTIM	Sets the system's time-of-day clock.
_VERSION	Returns a string containing the WMCS version banner, which contains a copyright notice and the revision number of the version of the WMCS running on your system.

Floating Point

_MAPFP	A process uses this system call to map the physical address of a hardware floating point device into its logical address space.

Networking

The following system calls allow processes to execute on a remote computer.

_CONNECT	Make a connection to a remote machine.
_DCONALL	Break all remote connections.
_DCONIDLE	Break all idle remote connections.
_DISCONN	Break a remote connection.
_GETNNAM	Get a nodename from a site ID.
_GETNSID	Get a site ID from a nodename.
_RNIDLST	Return a list of all known remote ID numbers.
_RSIDLST	Get a list of site IDs from a remote network.
_SIDLST	Return a list of all known site ID numbers on the current network.

Important Features of the System Call Library

The system call library is a set of procedures that allow programs written in C, FORTRAN, Assembler and Pascal to call the WMCS. The interface (system call name, parameter definition, parameter sequence, etc.) is uniform for each language.

Furthermore, a set of system table definitions is released with the WMCS. These files contain the structure or record definitions of all WMCS tables for Assembler, C, and Pascal. These files can be included in any program that refers to system tables, to provide up-to-date definitions. Note that this is particularly useful for systems integrators who write device drivers.

CHAPTER 3

DICTIONARY OF SYSTEM CALLS

Set alarm clock

Description:

Sets an alarm clock. When the system clock becomes greater than or equal to the specified value, the process will be terminated. Time is in the 64 bit system time format (absolute time or delta time).

The absolute time format of the date and time within these 8 bytes is as follows, where byte 0 is the most significant byte.

Bytes	Description
0,1	The current year (counted from A.D. 0). Example, 1983.
2,3	The day of the year (1..365 or 1..366)
4	The hour of the day (0..23)
5	The minute of the hour (0..59)
6	The second of the minute (0..59)
7	The fraction of a second (in 100'ths of a second) (0..99)

For delta time, the most significant long word is (-1). The least significant long word is a negative number whose absolute value is the number of ticks (.01 seconds per tick) from the current time.

Alarm clocks may be set only for the current process.

There can be only one alarm time per process. When _alarm is called, the previous setting is replaced with the new value.

Setting the alarm time to 0 resets (disables) the alarm clock.

Related Privileges:

None.

Parameters:

mstime - Most significant 32 bits of clock value
lstime - Least significant 32 bits of clock value

Diagnostics:

None.

See Also:

_wait - Pause for a period of time

Assembler Calling Sequence:

```
push    mstime                    ;value - most significant time bits
push    lstime                    ;value - least significant time bits
jsr     _alarm                    ;set alarm clock
```

C function declaration:

```
                                  /* set alarm clock */
void                              /* no result */
_alarm(mstime,lstime)
        long mstime;              /* most significant time bits */
        long lstime;              /* least significant time bits */
```

Fortran Subroutine Declaration:

```
c                                 ! set alarm clock
        subroutine alarm(mstime, lstime)
            integer*4 mstime      ! most significant time bits
            integer*4 lstime      ! least significant time bits
```

Pascal procedure declaration:

```
procedure _alarm(                 {** Set alarm clock }
        mstime  : longint;        {** most significant time bits }
        lstime  : longint         {** least significant time bits }
); external;
```

Allocate a device.

Description:

This system call is used to allocate a device for the exclusive use of a specified process and any spawned subprocesses of that process (see _crproc). Once a device is successfully allocated, other processes may not access that device except to read the device status (_getdst) and to flush the cache buffers (_flush).

A device may be allocated to at most one process. Subprocesses of the process to which the device is allocated will be able to access the device as though it were allocated to them.

To be successfully allocated, the specified device must not be currently referenced by any process. That is, the device must not be open by any other process. _getdst, _setdst, _physop, and other device operations also cause a device to be momentarily referenced. If the device is a virtual circuit (X.25) the device may not have an incoming session pending.

The device to be allocated may reside on any node to which the process has access.

The calling process also specifies the intended use (read operations, and/or write operations) of the device. The specified process must have access to the device for the intended use before the device can be alloced. For instance, if the intended use of the device is for read operations, the specified process must have read privilege to the device.

The calling process must have access to the process to which the device will be allocated. For instance, a process which does not have either world or group privilege may allocate a device to itself, or to any other process with the same user identification code.

If the specified device name is the name of a rotor list, this system call will select a device from the list that is currently not in use and to which the specified process has appropriate access (read privilege/write privilege).

The time out parameter is used to specify the maximum amount of time the calling process is willing to wait for the specified device (or a suitable device from the specified rotor list) to become available for allocation. The specified device (or the suitable devices on the specified rotor list) is checked once per second until it becomes available, or the time out expires. Note that if the specified process does not have access to the device (according to the specified intended use) the time out does not apply. That is, a non-zero status is returned to the calling process immediately, without waiting for the time out to expire.

Related Privileges:

none	- Allows allocation of devices to a process with the same user identification code (UIC) as the calling process.
group	- Allows allocation of devices to processes which have the same group id as the calling process.
world	- Allows allocation of devices to any process whatsoever.

Parameters:

pid	- Process IDentification number of the process to which a device will be allocated. The pids 0 and -1 have special meaning. 0 refers to the calling process, -1 refers to the parent of the calling process.
timout	- Should the specified device not be currently available, this svc will poll once per second until the specified timeout expires. This parameter is the number of 1/100th second ticks to wait for a device to become available.
access	- This parameter specifies the intended use for the allocated device by the specified process. The format of this parameter is the same as the mode parameter used by the open and create svcs except that bits 2-31 are reserved and should be zero. bit 0 = read access. 1=access desired, 0=no access. bit 1 = write access. 1=access desired, 0=no access. bit 2-31 = reserved. Should be 0.

dname	- Address of a null terminated string identifying the specific device (or rotor list name) which is to be allocated. This string will be translated automatically by WMCS into its logical equivalent. The string may contain up to 93 significant characters followed by a null, but must translate to a valid device or rotor list name of not more than 27 characters (16-character nodename with two underscores and an 8 character devicename with one underscore and a null).
alcnam	- Address of a 27-character string buffer which will contain the null terminated name of the successfully allocated device.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
erremptyrtrlst	(18)	The specified rotor list is empty.
errnamenull	(80)	The specified name must not be null.
errnoname	(82)	The specified name does not exist.
errtimeout	(128)	The request was not completed within the specified time.
errinvdevnam	(130)	The specified devicename is syntactically incorrect.
errundevnam	(131)	The MCS does not recognize the devicename. is the device mounted?
errnoreadpriv	(144)	The process does not have Read Privilege for the file.
errnowritepriv	(145)	The process does not have Write Privilege for the file.

See Also:

_dealloc	- Deallocate an allocated device.
_getalc	- Get names of allocated devices.
_getrel	- Get names of rotor list elements.
_getrtr	- Get rotor list names.
_setrtr	- Assign device names to a rotor list.

Assembler Calling Sequence:

```
push    pid                     ; value - process id
push    timout                  ; value - time out
push    access                  ; value - access mode
push    dname                   ; address - device name
push    alcnam                  ; address - allocated device
push    status                  ; address - result of the operation
jsr     _alloc                  ; Allocate a device
```

C Function Declaration:

```
                                /* Allocate a device */
long                            /* returns result of the operation */
_alloc(pid,timout,access,dname,alcnam)
        long    pid;            /* process id */
        long    timout;         /* time out */
        long    access;         /* access mode */
        char    dname[94];      /* device name */
        char    alcnam[27];     /* allocated device */
```

FORTRAN Subroutine Declaration:

```
c                               ! Allocate a device
        subroutine _alloc(pid,timout,access,dname,alcnam,status)
            integer*4   pid    ! process id
            integer*4   timout ! time out
            integer*4   access ! access mode
            character*94 dname ! device name
            character*27 alcnam ! allocated device
            integer*4   status ! result of the operation
```

Pascal Procedure Declaration:

```
procedure _alloc(               {** Allocate a device}
        pid     : longint;      {** process id}
        timout  : longint;      {** time out}
        access  : longint;      {** access mode}
        devnam  : string[93];   {** device name}
    var alcnam  : string[26];   {** allocated device}
    var status  : longint       {** result of the operation}
); external;
```

Allocate dynamic memory.

Description:

Allocate a 4K byte page of memory to the calling process, or share a 4K byte page of memory with another process.

For successful page allocation the address of the page must be on a 4K byte boundary; it must reside in the first 2 megabytes of address space (locations $000000 through $1FE000); and that address must not have been allocated previously by the process receiving the page. Note that for security reasons the process cannot allocate memory in the highest page of the logical address space, i.e. at location $1FF000.

Unless the process has writephys privilege, only pages owned by the calling process can be shared with another process.

To share a page, the value of the pid parameter is the process id of a process other than the calling process, i.e. the pid of the process to receive the page (receiving process). The value of the adr parameter specifies the address of a valid page of memory within the calling process. The page shared will have the same logical address in both the sharing process and the receiving process. For successful sharing, the receiving process must not have a page of logical memory already allocated at the specified address.

If the value of the pid parameter is zero or the process id of the calling process, a new page is allocated to the calling process.

Related Privileges:

none	- Can allocate memory to the current process or can share memory with processes with the same owner id and group id (uic)
group	- Can share memory with any process with the same group id.
world	- Can share memory with any process whatsoever.
writephys	- Can request that an unowned page of memory, assigned to the current process be shared with another process.

Parameters:

pid	- Process ID of which process is to receive the memory. 0 is used for the current process, -1 for the parent of the current process.

adr	- Logical address in the 2 megabyte logical address space. Adr must be aligned on 4K byte boundary.
prot	- Protection. 0 indicates no protection; 1 page is write protected; other values reserved.
timout	- The wait count is in 100'ths of a second and represents the amount of time to wait for a page to become available before returning an error.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
errprcsnotfnd	(2)	The specified process is not in the system process table.
errinvadr	(4)	The logical address, for the memory requested, is invalid.
errmemalloc	(5)	The process requested a logical page that was already allocated.
errnonowned	(6)	The process tried to affect a page in memory it did not own.
errnomemavail	(7)	All available memory has been allocated.
errtimeout	(128)	A request was not completed within the specifie time.

See Also:

_fremem - Deallocate a page of memory
_getfre - Get amount of available memory
_protmem- Change memory page protection

Assembler Calling Sequence:

```
push    pid                     ;value - process id
push    adr                     ;value - address of new page
push    prot                    ;value - protection
push    timout                  ;value - time out
push    status                  ;address - result of the operation
jsr     _allmem                 ;allocate dynamic memory
```

C Function Declaration:

```
                                /* allocate dynamic memory */
long                            /* returns result of the operation */
_allmem (pid, adr, prot, timout)
        long pid;               /* process id */
        long adr;               /* address of new page */
        long prot;              /* protection */
        long timout;            /* time out */
```

Fortran Subroutine Declaration:

```
c                                        ! allocate dynamic memory
        subroutine allmem(pid, adr, prot, timout, status)
                integer*4 pid     ! process id
                integer*4 adr     ! address of new page
                integer*4 prot    ! protection
                integer*4 timout  ! time out
                integer*4 status  ! result of the operation
```

Pascal Procedure Declaration:

```
procedure _allmem(                 {** allocate dynamic memory}
        pid     : longint;         {** process id}
        adr     : longint;         {** address of new page}
        prot    : longint;         {** protection}
        timout  : longint;         {** time out}
    var status  : longint          {** result of the operation}
); external;
```

Wait for AND of event flags

Description:

Suspend process execution until the logical AND of one or more event flags is true. When all of the specified event flags are simultaneously set (1's) the process is resumed.

Related Privileges:

none	- allows waiting on event flags of any process with the same owner id and group id (uic) as the calling process.
group	- allows waiting on event flags of processes with the same group id but a different owner id than the calling process.
world	- allows waiting on event flags of processes whose owner id and group id (uic) are other than those of the calling process.

Parameters:

pid	- Process ID of the process whose event flags are to be waited on. A 0 indicates the current process; -1 indicates the parent of the current process.
efmask	- Event flag mask. The mask of all bits which must simultaneously be set high for control to return to the calling process.
timout	- The wait count in 100'ths of a second. Represents the amount of time to wait for the specified event flags to be set before giving up. NOTE that time outs are not implemented in MCS 4.1.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
errprcsnotfnd	(2)	The specified process is not in the system process table.
errtimeout	(128)	A request was not completed within the specified time.

See Also:

_clrevnt - Clear event flags
_getevnt - Read event flags
_orevnt - Wait for OR of event flags
_setevnt - Set event flags

Assembler Calling Sequence:

```
push    pid                         ;value - process id
push    efmask                      ;value - event flag mask
push    timout                      ;value - time out
push    status                      ;address - result of the operation
jsr     _andevnt                    ;wait for AND of event flags
```

C function declaration:

```
                                    /* wait for AND of event flags */
long                                /* returns result of the operation */
_andevnt (pid, efmask, timout)
        long pid;                   /* process id */
        long efmask;                /* event flag mask */
        long timout;                /* time out */
```

Fortran Subroutine Declaration:

```
c                                   ! wait for AND of event flags
        subroutine andevn(pid, efmask, timout, status)
            integer*4 pid           ! process id
            integer*4 efmask        ! event flag mask
            integer*4 timout        ! time out
            integer*4 status        ! result of the operation
```

Pascal Procedure Declaration:

```
procedure _andevnt(                 {** wait for AND of event flags}
        pid     : longint;          {** process id}
        efmask  : longint;          {** event flag mask}
        timout  : longint;          {** time out}
    var status  : longint           {** result of the operation}
); external;
```

Assign a logical name.

Description:

Creates, deletes or replaces a logical name in the current process's translation table, or in the table of any other process.

Abbreviations are allowed in logical names. An asterisk (*) in the logical name is a marker that indicates the minimum string that is a recognized abbreviation of the logical name. Abbreviations are recognized only during logical name translation (see _trans). For example, if the logical name is "PR*INT", a translation of any of the strings "PR", "PRI", "PRIN", or "PRINT" will return the equivalence.

The values of the parameters lname and equiv determine whether an entry in the logical name table of the specified process is created, removed, or replaced.

To create a new logical name, the lname parameter must contain a logical name which does not match any existing logical names in the logical name table of the specified process and the equiv parameter must not be null.

To remove a logical name assignment, the lname parameter must contain a logical name which matches a logical name found in the logical name table of the specified process and the equiv parameter must be null.

To replace the equivalent string associated with a logical name the lname parameter must contain a logical name which matches an existing logical name found in the logical name table of the specified process and the equiv parameter must not be null.

If the lname parameter contains a logical name which does not match any existing name found in the logical name table and the equiv parameter is null, or if the lname parameter is null, this system call has no effect.

If the assignment is made in the current process's translation table, it (the assignment) is not in effect after the current process terminates. If the assignment is made in another process's translation table, it persists for the life of that process.

Related Privileges:

none	- Allows creation or replacement of a logical name in the translation table of any process with the same owner id and group id (uic) as the calling process.

group	- Allows creation or replacement of a logical name in the translation table of another process with the same group id as the calling process.
world	- Allows creation or replacement of a logical name in the translation table of any other process.

Parameters:

lname	- Address of null terminated string containing the logical name to be added, replaced or deleted from the logical name table of the specified process. This string may contain up to 93 characters plus a null.
equiv	- Address of null terminated string containing the equivalent to which the logical name translates. It this parameter contains a null string, the logical name represented in parameter lname is removed from the logical name table. This string may contain up to 93 characters plus a null.
pid	- The process id of the process for which this logical name will be in effect. 0=current process, -1=parent process.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
errprcsnotfnd	(2)	The specified process is not in the system process table.
errnomemavail	(7)	All available memory has been allocated.

See Also:

_gassign	- Assign a global logical name
_getglb	- Retrieve a global logical name
_getlog	- Retrieve a logical name
_gengy	- Get pid of ancestor process
_trans	- Translate a logical name

Assembler Calling Sequence:

```
push    lname                   ;address - logical name
push    equiv                   ;address - translation string
push    pid                     ;value - process id
push    status                  ;address - result of the operation
jsr     _assign                 ;assign a logical name
```

C function declaration:

```
                                /* assign a logical name */
```

```
    long                                /* returns result of the operation */
    _assign (lname, equiv, pid)
            char lname[94];             /* logical name */
            char equiv[94];             /* translation string */
            long pid;                   /* process id */
```

Fortran Subroutine Declaration:

```
    c                                   ! assign a logical name
            subroutine assign(lname, equiv, pid, status)
                character*94 lname      ! logical name
                character*94 equiv      ! translation string
                integer*4 pid           ! process id
                integer*4 status        ! result of the operation
```

Pascal Procedure Declaration:

```
    procedure _assign(                  {** assign a logical name}
            lname   : string[93];       {** logical name}
            equiv   : string[93];       {** translation string}
            pid     : longint;          {** process id}
        var status  : longint           {** result of the operation}
    ); external;
```

Set default device and directory.

Description:

Used by a process to change its default directory. Any subsequent file references that do not have an explicit path name will be assumed to be in this directory. In essence, the named path becomes the current working directory.

Unless the process has bypass privilege, it must have read privilege to the new default device, execute privilege to all directories up to the new default directory and read privilege to the default directory.

If the devdir is specified in fcb.seq number format, the process must have read privilege to the new default device and read privilege to the new default directory.

Related Privileges:

None	- Successful if process has access to the device and directories as described above.
altuic	- Successful if the owner of image file for the current process has access to the device and directory as described above.
bypass	- Allows the process to set the default to any mounted device and directory independent of the file protection.
system	- Successful if the system has access to the device and directory as described above.

Parameters:

devdir	- Address of a null terminated string which contains the new default device and directory specification. This string will be translated automatically by the MCS to its logical equivalence. This parameter may have up to 93 characters (the null makes 94)
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errnomemavail	(7)	All available memory has been allocated.
errinvdevnam	(130)	The specified devicename is syntactically incorrect.
errundevnam	(131)	The MCS does not recognize the devicename. Is the device mounted?
errnoexecpriv	(143)	The process does not have Execute Privilege

		for the file.
errnoreadpriv	(144)	The process does not have Read Privilege for the file.
errinvdirfle	(148)	The specified directory is not a directory.
errinvdirstr	(149)	The specified directory name is syntactically incorrect.
errinvcloper	(173)	The device class is inappropriate for the operation.
errinvdirdev	(174)	Directories do not exist on the specified device.
errdirnotfnd	(177)	The specified directory does not exist.
errinvseqnum	(178)	The file's FCB.SEQ number in the directory file is incorrect.
		Device integrity errors.

See Also:

_create - Create a file
_delete - Delete a file
_getdir - Get default device and directory
_open - Open a file
_rename - Rename a file
_setfprt- Set file protection

Assembler Calling Sequence:

```
    push    devdir                   ;address - new device/directory
    push    status                   ;address - result of the operation
    jsr     _chdir                   ;set default device and directory
```

C function declaration:

```
                                     /* set default device and directory */
    long                             /* returns result of the operation */
    _chdir(devdir)
            char devdir[94];         /* new device/directory */
```

Fortran Subroutine Declaration:

```
    c                                ! set default device and directory
            subroutine chdir(devdir, status)
                character*94 devdir  ! new device/directory
                integer*4 status     ! result of the operation
```

Pascal Procedure Declaration:

```
    procedure _chdir(                {** set default device and directory}
            devdir  : string[93];    {** new device/directory}
        var status  : longint        {** result of the operation}
    ); external;
```

Change to supervisor mode.

Description:

Sets the supervisor bit (bit 13) in the CPU status register. Allows execution of privileged instructions.

If the call is successful, the system returns control to the process with the CPU in supervisor mode. The process will continue in supervisor mode until the process changes the status register back to user mode.

Note especially that with the change to supervisor mode comes a transition to using the supervisor stack pointer. The supervisor stack is approximately 1700 bytes in length and is located in system memory. Overflowing the system stack will crash the process and probably the system also.

Data that was on the users stack prior to this call will have to be accessed differently while in supervisor mode.

Note that with the processor in supervisor mode, the user has complete access to all hardware features of the system. Indiscriminate memory accesses may lead to unexpected and disastrous results.

Related Privileges:

none	- Process not allowed to change to supervisor mode
chngsuper	- Allows process to change to supervisor mode

Parameters:

status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.

See Also:

_chuser - Change processor mode to user

Assembler Calling Sequence:

```
push    status                    ;address - result of the operation
jsr     _chsuper                  ;change to supervisor mode
```

C function declaration:

```
                                  /* change to supervisor mode */
    long                          /* returns result of the operation */
    _chsuper()
```

Fortran Subroutine Declaration:

```
    c                             ! change to supervisor mode
            subroutine chsupe(status)
                integer*4 status  ! result of the operation
```

Pascal Procedure Declaration:

```
    procedure _chsuper(           {** change to supervisor mode}
        var status  : longint     {** result of the operation}
    ); external;
```

Change processor mode to user.

Description:

Clears the supervisor bit (bit 13) in the CPU status register. Provides high level languages the ability to convert back to user mode from supervisor mode.

Note that with the change to user mode comes a transition to using the user stack pointer.

Note that unless the process is currently in supervisor mode a fatal error will occur when an attempt is made to write to the status register.

Related Privileges:

None.

Parameters:

None.

Diagnostics:

None.

See Also:

_chsuper - Change to supervisor mode

Assembler Calling Sequence:

```
jsr     _chuser                     ;change processor mode to user
```

C function declaration:

```
                                    /* change processor mode to user */
void                                /* no result */
_chuser()
```

Fortran Subroutine Declaration:

```
c                                   ! change processor mode to user
        subroutine chuser
```

Pascal Procedure Declaration:

```
procedure _chuser;                  {** change processor mode to user}
```

```
          external;
```

Create a new process by cloning an existing process

Description:

This call is similar to _crproc except that rather than load the image for the process to be created from some mass storage media specified by a file name, the process is created via copying (cloning) an already existing process specified by a PID.

Each process under control of the operating system must be created by a call to this operating system service routine or to _crproc, _crprcs, or _mulcrps. When a process is created, it is called a child process. The process that created it is called its parent process.

SPAWN and FORK are two different modes of creation. Spawned processes run in series. This means that the parent process hibernates while the child process runs. When the child process terminates, the parent process resumes. The completion status of the child is returned to the parent.

Forked processes run in parallel. The parent process is not hibernated, but continues execution immediately after successful creation of the child process.

The calling process must be able to access the process specified by the PID either via group privilege, world privilege, or the protection allowing public access to it for successful creation of the cloned process.

Without the setpriv privilege, the child may not be given more privileges than the parent has.

The child process is created with the same default device and directory as the parent.

Related Privileges:

none	- Allows the parent process to create a child from an existing process to which the parent has only public access. The child may not be given privileges not possessed by the parent.

group	- Allows the parent process to create a child process with the same group ID but a different owner ID than the parent process has. Also allows the cloning of processes with the same group ID but a different owner ID that the creating process has.
setpriv	- Allows the parent process to give the child process more privileges than those possessed by the parent.
setprior	- Allows the parent process to initiate a child at a higher priority level and/or with a higher timeslice than the parent.
world	- Allows the parent process to create a child with any owner ID and group ID (UIC) whatsoever. Also allows a process to clone a process with any owner ID and group ID (UIC) whatsoever.

Parameters:

mode - Whether the process is spawned or forked. A 0 indicates spawn, 1 indicates fork. All other values are reserved and should not be used.

pid - The process ID of the process to be cloned. The new process will be created on the same site where the process being cloned exists.

pname - Address of a 17 byte null terminated string containing the process name to be given the new process. This string is used for human identification. (16 significant characters plus a null)

priv - The privilege mask contains a bit mask of privileges to be given to the child process. A -1 indicates that the child should receive the same privileges that the parent has. Privileges are bit encoded as follows:

Bit Name	Bit #	Description
pcbpvsetpriv	0	setpriv
pcbpvsystem	1	system
pcbpvreadphys	2	readphys
pcbpvwritephys	3	writephys
pcbpvsetprior	4	setprior
pcbpvchngsuper	5	chngsuper
pcbpvbypass	6	bypass
pcbpvoperator	7	operator
pcbpvaltuic	8	altuic
pcbpvworld	9	world
pcbpvgroup	10	group

pcbpvnetwork	11	network
pcbpvsetattr	12	setattr
	13-31	Reserved. Must be set to zero.

priort - The priority level (0..15) at which the child process will execute. Level 0 is the highest priority. A minus one (-1) in this parameter means to use the same priority as the parent process.

tslice - The timeslice value. The maximum amount of time the child process will be able to run each time it is scheduled. This time is specified in .01 milliseconds. (A timeslice of 100 represents 1 millisecond) A minus one (-1) in this parameter means to use the same timeslice as the parent process.

uic - The user identification code of the child process. The most significant 16 bits represent the owner ID and the least significant 16 bits represent the group ID. A minus one (-1) in this parameter means to use the same UIC as the parent process.

sysin - Address of a 94 byte null terminated string containing the name of the standard input file for the child process. This string will be translated automatically by the WMCS to its logical equivalent. The equivalent string will be assigned the logical name "SYS$INPUT" in the logical name table of the child process. The string passed is NOT checked for validity. It may contain up to 93 significant characters followed by a null.

sysout - Address of a 94 byte null terminated string containing the name of the standard output file for the child process. This string will be translated automatically by the WMCS to its logical equivalent. The equivalent string will be assigned the logical name "SYS$OUTPUT" in the logical name table of the child process. The string passed is NOT checked for validity. It may contain up to 93 significant characters followed by a null.

syserr - Address of a 94 byte null terminated string containing the name of the standard error file for the child process. This string will be translated automatically by the WMCS to its logical equivalent. The equivalent string will be assigned the logical name "SYS$ERROR" in the logical name table of the child process. The string passed is NOT checked for validity. It may contain up to 93 significant characters followed by a null.

cmd - Address of the command line. (up to 3072 bytes) The command line may contain any data whatever to be passed from the parent to the child.

The data appears on the top of the child process's stack as the child process begins. The long word at the top of the child's stack is the length in bytes of the command line. At the location (USP+4) on the child's stack is a long word which contains the starting address of the command line.

cmdlen	- Length of the command line specified in bytes.
chpid	- Address of a long word to receive the PID of the child process. Note that this is only valuable in the case that the child is forked. If the address of the long word is zero, no value is returned.
ccode	- Address of a long word to receive the completion code returned to the parent by the process responsible for terminating the child process. If the child is exited as a result of a system violation (memory violation, illegal instruction, ...) the system supplies the ccode. If the process terminates normally, the process itself supplies the ccode. If the process is exited by another process, the other process supplies the ccode. Note that the ccode will always be zero for processes that are forked. If the address of the long word is zero, no value is returned. Completion codes that may be supplied by the system include:

erralarmexit	(28)	The system clock reached the value specified for _ALARM.
errzerodivtrap	(29)	The process has an undefined trap: Divide-by-zero.
errchktrap	(30)	The process has an undefined trap: CHK Instruction.
errtrapvtrap	(31)	The process has an undefined trap: TRAPV Instruction.
errtracetrap	(32)	The process has an undefined trap: TRACE.
err1010trap	(33)	The process has an undefined trap: 1010 Instruction.
err1111trap	(34)	The process has an undefined trap: 1111 Instruction.
errprivintrap	(35)	The process attempted to execute a privileged instruction.
errillintrap	(36)	The process attempted to execute an illegal instruction.
errbustrap	(37)	The process has a bus error.
erradrtrap	(38)	The process has an address error.
errnonexmem	(39)	The process attempted to access nonexistent memory.
errmemparity	(40)	The process has a memory parity-error.

errwriteprot	(41)	The process attempted to write to a write-protected page in memory.
errundeftrap	(42)	_SETTRP was not used to define a call for a trap other than TRAP 0.
errundefsvc	(43)	The MCS does not recognize the SVC number used by the process.
errcontccode	(255)	[CTRL] c terminated the process.

status - Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
errnomemavail	(7)	All available memory has been allocated.
errinvsiteid	(8)	The specified site ID does not exist.
errnotimfle	(21)	The specified file is not an image file.
errimagebmbad	(53)	(MCS error) The bitmap changed during the creation of the process.
errinvdevnam	(130)	The specified devicename is syntactically incorrect.
errundevnam	(131)	The MCS does not recognize the devicename. Is the device mounted?
errfilnotfnd	(133)	The specified file could not be found.
errreadleof	(140)	The process tried to read past the logical end of a file.
errnoexecpriv	(143)	The process does not have Execute Privilege for the file.
errnoreadpriv	(144)	The process does not have Read Privilege for the file.
errinvfnstr	(147)	The specified filename is syntactically incorrect.
errinvdirfle	(148)	The specified directory is not a directory.
errinvdirstr	(149)	The specified directory name is syntactically incorrect.
errdirnotfnd	(177)	The specified directory does not exist.
errfilopen	(202)	The process tried to simultaneously open more than one tape file.

See Also:

_crprcs	- Simplified create process
_crproc	- Create a new process
_exproc	- Terminate the specified process
_mulcrps	- Multiple create process
_setpnam	- Change process name
_setpri	- Change priority level
_settmsl	- Change scheduling timeslice
_setuic	- Set process UIC

Assembler Calling Sequence:

```
push    mode                    ;value - spawn or fork
push    pid                     ;value - pid of process to clone
push    pname                   ;address - process name
push    priv                    ;value - process privilege
push    priort                  ;value - process priority
push    tslice                  ;value - process timeslice
push    uic                     ;value - user identification code
push    sysin                   ;address - standard input file
push    sysout                  ;address - standard output file
push    syserr                  ;address - standard error file
push    cmd                     ;address - command line
push    cmdlen                  ;value - length of cmd
push    pid                     ;address - childs pid
push    ccode                   ;address - childs completion code
push    status                  ;address - result of the operation
jsr     _clone                  ;clone an existing process
```

C Function Declaration:

```
                                /* clone an existing process */
long                            /* returns result of the operation */
_clone(mode, pid, pname, priv, priort, tslice, uic,
                    sysin, sysout, syserr, cmd, cmdlen, chpid, ccode)
        long mode;              /* spawn or fork */
        long pid;               /* PID of process to clone */
        char pname[17];         /* process name */
        long priv;              /* process privilege */
        long priort;            /* process priority */
        long tslice;            /* process timeslice */
        long uic;               /* user identification code */
        char sysin[94];         /* standard input file */
        char sysout[94];        /* standard output file */
        char syserr[94];        /* standard error file */
        char cmd[3072];         /* command line */
        long cmdlen;            /* length of cmd */
        long *chpid;            /* childs pid */
        long *ccode;            /* childs completion code */
```

FORTRAN Subroutine Declaration:

```
     c                                 ! clone an existing process
             subroutine _clone(mode, pid, pname, priv, priort,
             &           tslice, uic,sysin, sysout, syserr, cmd,
             &           cmdlen, chpid, ccode,status)
                 integer*4 mode       ! execution mode (spawn or fork)
                 integer*4 pid        ! PID of process to clone
                 character*17 pname   ! process name
                 integer*4 priv       ! process privilege
                 integer*4 priort     ! process priority
                 integer*4 tslice     ! process timeslice
                 integer*4 uic        ! user identification code
                 character*94 sysin   ! standard input file
                 character*94 sysout  ! standard output file
                 character*94 syserr  ! standard error file
                 character*(*) cmd    ! command line
                 integer*4 cmdlen     ! length of cmd
                 integer*4 chpid      ! childs pid
                 integer*4 ccode      ! childs completion code
                 integer*4 status     ! result of the operation
```

Pascal Procedure Declaration:

```
     procedure _clone(                {** clone an existing process }
            mode    : longint;        {** spawn or fork}
            pid     : longint;
            pname   : string[16];     {** process name}
            priv    : longint;        {** process privilege}
            priort  : longint;        {** process priority}
            tslice  : longint;        {** process timeslice}
            uic     : longint;        {** user identification code}
            sysin   : string[93];     {** standard input file}
            sysout  : string[93];     {** standard output file}
            syserr  : string[93];     {** standard error file}
            cmd     : ^array_of_char; {** command line}
            cmdlen  : longint;        {** length of cmd}
        var chpid   : longint;        {** childs pid}
        var ccode   : longint;        {** childs completion code}
        var status  : longint         {** result of the operation}
     ); external;
```

Close a file.

Description:

Given a valid logical unit number (lun), close a file. That is, make the file inaccessible to the current process through that lun. If the flush flag is set on a disk device, all disk cache buffers will be written to the device. If the device is a tape, the tape buffer is written to the device.

Any pending errors encountered during asynchronous reads to this file will be returned as warnings to the _close.

If the delete option is specified in the mode parameter, the process must have read and write privilege to the device containing the file, read and write privilege to the directory containing the file, and delete privilege to the file itself in order for the file to be successfully deleted.

Related Privileges:

none	- The file will be closed. Allows optional deletion of the file if the process has privileges to the file as described above. Returns a warning if the process specified delete upon closing and does not have privilege to the file as described above.
bypass	- Allows the process to delete the file upon closing, independent of the process's privilege to the file.
system	- Allows the process to delete the file upon closing if the system has privilege to the file as described above.

Parameters:

lun	- Logical unit number.
mode	- Bit encoded long word specifying action to be taken upon closing. If the bit is a zero, no action is performed. The following descriptions apply when the specified bit is set (1):

15	9	8	7	6	5	4	3	2	1	0
						1	0	1	1	0

Bit Name	Bit #	Description
cldelete	0	Delete - Requests that the file be deleted upon closing. If other processes have the file open, the file will be marked for deletion, no error is returned, and as soon as the file is closed by all processes it will be deleted.
clnotruncfile	1	No truncate - Specifies that when the disk file is closed, the extra physical sectors allocated to the file are not to be released. For tape devices, this bit specifies that the last block written to the tape should be written as a full sized block (as opposed to a variable sized block).
clnodelete	2	No delete - Overrides the delete upon closing request specified by the _open system call.
clforcedwrite	3	Forced write - Writes to the device all data in system buffers associated with this lun. If an error occurs it will be reported as a warning to the calling process. The file is always closed.
clsupalldelete	4	Suppress all deletes - Overrides all deletes that have been set for the file, i.e., opdelete or a delete set by a different process.
clzerodelete	5	Zero delete - Zero each sector of the file before deleting the file. This bit is only valid if the file is being deleted (via cldelete or some other way).
	6-31	Reserved. Must be set to zero.

status - Address of a long word to receive the result of the operation.

Diagnostics:

errinvlfn	(132)	The logical unit number does not correspond to an open file.
errnodelpriv	(146)	The process does not have Delete Privilege for the file.

erropendel	(153)	The specified file is open, has been marked for deletion.
errdelfile	(158)	System files cannot be deleted.
errdevwrtprot	(269)	The specified device is write-protected.

See Also:

_create	- Create a file
_delete	- Delete a file
_frdwait	- Wait for a fast read to complete
_open	- Open a file

Assembler Calling Sequence:

```
push    lun                         ;value - logical unit number
push    mode                        ;value - mode word
push    status                      ;address - result of the operation
jsr     _close                     ;close a file
```

C Function Declaration:

```
                                    /* close a file */
long                                /* returns result of the operation */
_close (lun, mode)
         long lun;                  /* logical unit number */
         long mode;                 /* mode word */
```

Fortran Subroutine Declaration:

```
c                                    ! close a file
         subroutine _close (lun, mode, status)
             integer*4 lun        ! logical unit number
             integer*4 mode       ! mode word
             integer*4 status     ! result of the operation
```

Pascal Procedure Declaration:

```
procedure _close(              {** close a file}
        lun      : longint;     {** logical unit number}
        mode     : longint;     {** type of access requested}
    var status   : longint      {** result of the operation}
); external;
```

Clear event flags.

Description:

Clears the specified event flags of a particular process.

Related Privileges:

None	- Allows clearing event flags of any process with the same owner id and group id as the calling process.
group	- Allows clearing the event flags of any process with the same group id as the calling process.
world	- Allows clearing the event flags of any process whatever.

Parameters:

pid	- Process ID of the process whose event flags are to be cleared. 0 refers to current process; -1 references the parent of the current process.
efmask	- Event flag mask. The 32 bit mask of those flags which are to be cleared. Bits set within the mask correspond to the event flags which will be cleared.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
errprcsnotfnd	(2)	The specified process is not in the system process table.

See Also:

_andevnt	- Wait for AND of event flags
_getevnt	- Read event flags
_orevnt	- Wait for OR of event flags
_setevnt	- Set event flags

Assembler Calling Sequence:

```
push    pid                     ;value - process id
push    efmask                  ;value - event flag mask
push    status                  ;address - result of the operation
jsr     _clrevnt                ;clear event flags
```

C function declaration:

```
                                    /* clear event flags */
    long                            /* returns result of the operation */
    _clrevnt (pid, efmask)
            long pid;               /* process id */
            long efmask;            /* event flag mask */
```

Fortran Subroutine Declaration:

```
    c                               ! clear event flags
            subroutine clrevn(pid, efmask, status)
                integer*4 pid       ! process id
                integer*4 efmask    ! event flag mask
                integer*4 status    ! result of the operation
```

Pascal Procedure Declaration:

```
    procedure _clrevnt(             {** clear event flags}
            pid     : longint;      {** process id}
            efmask  : longint;      {** event flag mask}
        var status  : longint       {** result of the operation}
    ); external;
```

Make a connection to a remote machine.

Description:

This system call is used to establish a logical connection with a remote machine. It does this by allocating a network link (virtual circuit) to the process for use in network communication.

There must be an entry in the remote machine's NETUAF.DAT file matching the UIC of the process requesting the connection for the _connect to succeed.

This SVC does not need to be called prior to accessing other nodes on a network. All SVCs that access other nodes in the network will automatically issue a connect request if the process does not already have an open connection to the node. Use this SVC if you want to ensure that you have a good connection to another node prior to performing any operations on that node. This may simplify error reporting.

Related Privileges:

none	- Process not allowed to access the network.
network	- Process allowed to perform network operations.

Parameters:

siteid	- Site ID of the system with which a connection is being attempted.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
errnomemavail	(7)	All available memory has been allocated.
errinvsiteid	(8)	The specified site ID does not exist.
errremotelogon	(47)	The process was not allowed to logon to the remote system
errnoclass	(185)	The device class handler was not loaded when the system was booted.

See Also:

_disconn - Break a remote connection
_dconall - Break all remote connections
_dconidle - Break all idle remote connections

Assembler Calling Sequence:

```
push    siteid                  ;value - site being connected to
push    status                  ;address - result of the operation
jsr     _connect                ;make a remote connection
```

C Function Declaration:

```
                                /* make a remote connection */
long                            /* returns result of the operation */
_connect(siteid);
         long siteid;           /* site being connected to */
```

FORTRAN Subroutine Declaration:

```
c                               ! make a remote connection
         subroutine _connec(siteid, status)
             integer*4 siteid   ! site being connected to
             integer*4 status   ! result of the operation
```

Pascal Procedure Declaration:

```
procedure _connect(             {** make a remote connection }
        siteid  : longint;      {** site being connected to }
    var status  : longint       {** result of the operation }
); external;
```

Create a file.

Description:

After logical name translation, the specified file is created. Upon successful completion of the create, the file is opened and the logical unit number is returned. If a specific version number is requested, the file is created provided that there is no file with the specified filename and version number. If the version number is 0 or no version number is specified, the new file will be assigned a version number one higher than the previous highest version number on a file with the same name in the specified directory.

Unless the process has bypass privilege, it must have read and write privilege to the device to contain the file, execute privilege to the device to contain the file, execute privilege for all directories in the path leading to the file, and read and write privilege to the directory to contain the file for the file to be successfully created.

If the delete upon closing option is specified in the mode parameter, the process must have read and write privilege to the device containing the file, read and write privilege to the directory containing the file, and delete privilege to the file itself for the file to be successfully deleted.

Related Privileges:

none	- Allows creation only if process has access as described above. The created file may not have a UIC other than that of the calling process.
altuic	- Allows creation if the owner of the image file for the current process has access as described above.
bypass	- Allows the process to create the file independent of the file protection.
group	- Allows the process to create a file with the same group ID but a different owner ID than the calling process.
system	- Allows creation if the system has access as described above.
world	- Allows the process to create a file with any UIC.

Parameters:

fname - Address of a null terminated string containing the name of the file to be created. The string will be translated automatically by WMCS to its logical equivalence. This string may contain up to 93 significant characters followed by a null.

mode - Bit encoded long word specifying the type of access required. The following description explains the options when the specified bit is set (1).

Bit Name	Bit #	Description
opreadacc	0	Read access - Requests permission to read the file.
opwriteacc	1	Write access - Requests permission to write the file.
opreadlock	2	Read lock - Requests permission for exclusive read access to the file.
opwritelock	3	Write lock - Requests permission for exclusive write access to the file.
opdelete	4	Delete - Requests that the file be deleted upon closing.
opappend	5	Append - Specifies that the initial file position be at the logical end of file.
opfastread	6	Fast read - Specifies that the file will be read asynchronously. That is, that control returns to the user process before the data have actually been read. As records are read, they will be transferred directly into the process's logical address space bypassing the device cache. This bit is only valid for disk class devices. Other requirements are 1) Supports only requests for complete sectors only, 2) Process buffer must be on a word boundary, 3) Request cannot cross a 4 Kbyte page boundary. Use the _frdwait system call to determine when asynchronous reads are complete.
opnextfile	7	Open next file - On a tape device, specifies to open the "next" file without regard to the filename.

opnordahead	8	No read ahead - Specifies that read ahead is not to be done on the opened file.
opnotruncfile	9	No truncate - Specifies that when the file is closed the extra physical sectors allocated to the file are not to be released.
cropenifthere	10	Open if there - Specifies that the file will be opened if it exists. Only if it does not exist will it be created. If the file does exist and this bit is set the ftype, prot., uic, fid, mstime, and lstime parameters are ignored. The reclen parameter will specify the record length for this open and does not alter the default record length associated with the file.
cropenshared	11	Open shared - Specifies that if the current process or any ancestor of the current process has a file with the specified name (fname) and with the same access modes currently open, this process will share the file with the ancestor, including the default file position. As the file is read or written, the default position is adjusted for both the current process and the ancestor.
opzerodelete	12	Zero delete - Zero each sector of the file before deleting the file. This bit is only valid if the file is being deleted (via cldelete or some other way).
	13-31	Reserved. Must be set to zeros.

reclen - Default file record length in bytes. Must be between 0 and 65535. In the case of the "open if there" mode and the file exists, this parameter overrides the default record length specified for the file. If a zero or $FFFFFFFF (-1) is used, the file will be created with a record length of 1. In the case that the named file exists and the "open if there" bit is set, a $FFFFFFFF (-1) signifies that the default length of the specified file is to be used.

ftype - A numerically valued field specifying the file type.

File Type	Value	Description
fcbftdata	0	data
fcbftdir	1	directory
fcbftimage	2	image file
fcbftksamdata	3	KSAM data file
fcbftksamkey	4	KSAM key file
fcbftllimage	5	LL image file
fcbftarchcont	6	archive file continuation
	7	reserved
fcbftsystem	8	system file
fcbftarchive	9	archive file
	20-255	reserved
	256-65535	user defined

prot - File protection mask. The least significant 16 bit word of this parameter is divided into 4 nibbles. Each nibble corresponds to a class of users. The bits within each nibble represent the type of access that class of user is granted for this file. If the bit is set (1), the access is granted.

From the least to the most significant nibble the user classes are:

Ownr - file owner
Grp - processes with the same group ID as the owner
Pub - all other processes in the system
Sys - processes with SYSTEM privilege.

```
 Sys  Pub  Grp  Ownr
|----|----|----|----|
|DWRE|DWRE|DWRE|DWRE|
|-------------------|
MSB                LSB
```

From the least to the most significant bits within the nibbles, the access privileges are:

E - Execute access
R - Read access
W - Write access
D - Delete access

The value $FFFFFFFF (-1) is a reserved value that means that the user's default protection mask is to be used.

uic	- The user identification code, specifying the owner of the file. The most significant 16 bits of this parameter contain the owner ID, and the least significant 16 bits contain the group ID. A value of $FFFFFFFF (-1) is a reserved value which means to give the file the same UIC as the calling process.
fid	- The least significant 16 bits of this parameter become the file identification code to be associated with the file.
mstime	- The most significant 32 bits of the file creation date and time in system time format. This parameter may be used to specify a file creation date other than the current date. If the value of this parameter is $FFFFFFFF (-1), the current date (year and day) will be used. Otherwise, the value specified will be used for the creation date. This value is not checked for validity.
lstime	- The least significant 32 bits of the file creation date and time in system time format. This parameter may be used to specify a file creation time other than the current time. If the value of this parameter is $FFFFFFFF (-1), the current time (hour, minute, second and tick) will be used. Otherwise, the value specified will be used for the creation time. This value is not checked for validity.
lun	- Address of a long word to receive the logical unit number of the open file.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
errnomemavail	(7)	All available memory has been allocated.
errinvvernum	(129)	A file's version number cannot be greater than 65535.
errinvdevnam	(130)	The specified devicename is syntactically incorrect.
errundevnam	(131)	The MCS does not recognize the devicename. Is the device mounted?
errfilexists	(134)	The specified version of the file already exists.
errinvreclen	(138)	Edit mode 3 requires that the file's record length be set to one.
errinvfiletype	(139)	The specified file type is reserved for the MCS.
errnoexecpriv	(143)	The process does not have Execute Privilege for the file.

errnoreadpriv	(144)	The process does not have Read Privilege for the file.
errnowritepriv	(145)	The process does not have Write Privilege for the file.
errnodelpriv	(146)	The process does not have Delete Privilege for the file.
errinvfnstr	(147)	The specified filename is syntactically incorrect.
errinvdirfle	(148)	The specified directory is not a directory.
errinvdirstr	(149)	The specified directory name is syntactically incorrect.
errinvcloper	(173)	The device class is inappropriate for the operation.
errdirnotfnd	(177)	The specified directory does not exist.
errdirinvver	(200)	A directory file cannot have a version number greater than one.
errfilopen	(202)	The process tried to simultaneously open more than one tape file.

See Also:

_close	- Close a file
_creats	- Simplified file creation
_defprot	- Set default protection mask
_delete	- Delete a file
_open	- Open a file
_setfprt	- Set file protection

Assembler Calling Sequence:

```
push    fname               ;address - file name
push    mode                ;value - type of access requested
push    reclen              ;value - record length
push    ftype               ;value - file type
push    prot                ;value - protection
push    uic                 ;value - user identification code
push    fid                 ;value - file ID
push    mstime              ;value - day and year
push    lstime              ;value - hour, minute, second, tick
push    lun                 ;address - logical unit number
push    status              ;address - result of the operation
jsr     _create             ;create a file
```

C Function Declaration:

```
                                  /* create a file */
    long                          /* returns result of the operation */
    _create (fname, mode, reclen, ftype, prot, uic, fid, mstime,
                     lstime, lun)
            char fname[94];       /* file name */
            long mode;            /* type of access requested */
            long reclen;          /* record length */
            long ftype;           /* file type */
            long prot;            /* protection */
            long uic;             /* user identification code */
            long fid;             /* file ID */
            long mstime;          /* day and year */
            long lstime;          /* hour, minute, second, tick */
            long *lun;            /* logical unit number */
```

FORTRAN Subroutine Declaration:

```
    c                              ! create a file
            subroutine _create (fname, mode, reclen, ftype, prot,
         &            uic, fid, mstime, lstime, lun, status)
                character*94 fname ! file name
                integer*4 mode     ! type of access requested
                integer*4 reclen   ! record length
                integer*4 ftype    ! file type
                integer*4 prot     ! protection
                integer*4 uic      ! user identification code
                integer*4 fid      ! file ID
                integer*4 mstime   ! day and year
                integer*4 lstime   ! hour, minute, second, tick
                integer*4 lun      ! logical unit number
```

Pascal Procedure Declaration:

```
    procedure _create(            {** create a file}
            fname   : string[93]; {** file name}
            mode    : longint;    {** type of access requested}
            reclen  : longint;    {** record length}
            ftype   : longint;    {** file type}
            prot    : longint;    {** protection}
            uic     : longint;    {** user identification code}
            fid     : longint;    {** file ID}
            mstime  : longint;    {** day and year}
            lstime  : longint;    {** hour, minute, second, tick}
        var lun     : longint;    {** logical unit number}
        var status  : longint     {** result of the operation}
    ); external;
```

Simplified file creation.

Description:

This system call is simplified form of _create. Default values are assumed for the file type (data file), the file protection (uses the user's default protection mask), the uic (becomes the same as that of the calling process), fid (uses 0), creation date and time (uses the current date and time).

After logical name translation, the specified file is created. Upon successful completion of the create, the file is opened and the logical unit number is returned. If a specific version number is requested, the file is created provided that there is no file with the specified file name and version number. If version number 0, or no version number is specified, the new file will be assigned a version number one higher than the previous highest version number on a file with the same name in the specified directory.

Unless the process has bypass privilege, it must have read and write privilege to the device to contain the file, execute privilege for all directories in the path leading to the file, and read and write privilege to the directory to contain the file for the file to be successfully created.

If the delete upon closing option is specified in the mode parameter, the process must have read and write privilege to the device containing the file, read and write privilege to the directory containing the file and delete privilege to the file itself for the file to be successfully deleted.

Related Privileges:

none	- Allows creation only if process has access as described above.
altuic	- Allows creation if the owner of the image file for the current process has access as described above.
bypass	- Allows the process to create the file independent of the file protection.
system	- Allows creation if the system has access as described above.

Parameters:

fname - Address of a null terminated string containing the name of the file to be created. The string will be translated automatically, by the MCS to its logical equivalence. This string may contain up to 93 significant characters followed by a null.

mode - Bit encoded long word specifying the type of access required. The following description explains the options when the specified bit is set (1).

Bit Name	Bit	Description
opreadacc	0	Read access - Requests permission to read the file.
opwriteacc	1	Write access - Requests permission to write the file.
opreadlock	2	Read lock - Requests permission for exclusive read access to the file.
opwritelock	3	Write lock - Requests permission for exclusive write access to the file.
opdelete	4	Delete - Requests that the file be deleted upon closing
opappend	5	Append - Specifies that the initial file position be at the logical end of file.
opfastread	6	Fast read - Specifies that the file will be read asynchronously. That is, that control returns to the user process before the data has actually been read. As records are read, they will be transferred directly into the process's logical address space, bypassing the device cache. This bit is only valid for disk class devices. Other requirements are: 1) Supports requests for complete sectors only, 2) Process's buffer must be on a word boundary, 3) The request cannot cross a 4-Kbyte page boundary. Use the _frdwait system call to determine when asynchronous reads are complete.
opnextfile	7	Open next file - On a tape device, specifies to open open the 'next' file without regard to the file's name.

opnordahead	8	No read ahead - Specifies that read ahead is not to be done on the opened file.
opnotruncfile	9	No truncate - Specifies that when the file is closed the extra physical sectors allocated to the file are not to be released.
cropenifthere	10	Open if there - Specifies that the file will be opened if it exists. Only if it does not exist will it be created. If the file does exist and this bit is set, the ftype, prot, uic, fid, mstime, and lstime parameters are ignored. The reclen parameter will specify the record length for this open and does not alter the default record length associated with the file.
cropenshared	11	Open shared - Specifies that if the current process or any ancestor of the current process has a file with the specified name (fname) and with the same access modes currently open, this process will share the file with the ancestor, including the default file position. As the file is read or written, the default position is adjusted for both the current process and the ancestor.
opzerodelete	12	Zero delete - Zero each sector of the file before deleting the file. This bit is only valid if the file is being deleted (via cldelete or some other way).
	13-31	Reserved. Must be set to zero.

reclen - Default file Record length in bytes. Must be between 0 and 65535. In the case of the 'open if there' mode and the file exists, this parameter overrides the default record length specified for the file.
If a zero or $FFFFFFFF (-1) is used, the file will be created with a record length of 1. In the case that the named file exists and the 'Open if there' bit is set, a $FFFFFFFF (-1) signifies that the default length of the specified file is to be used.

lun	- Address of a long word to receive the logical unit number of the open file.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errnomemavail	(7)	All available memory has been allocated.
errinvvernum	(129)	A file's version number cannot be greater than 65535.
errinvdevnam	(130)	The specified devicename is syntactically incorrect.
errundevnam	(131)	The MCS does not recognize the devicename. Is the device mounted?
errfilexists	(134)	The specified version of the file already exists.
errinvreclen	(138)	Edit mode 3 requires that the file's record length be set to one.
errinvfiletype	(139)	The specified file type is reserved for the MCS.
errnoexecpriv	(143)	The process does not have Execute Privilege for the file.
errnoreadpriv	(144)	The process does not have Read Privilege for the file.
errnowritepriv	(145)	The process does not have Write Privilege for the file.
errnodelpriv	(146)	The process does not have Delete Privilege for the file.
errinvfnstr	(147)	The specified filename is syntactically incorrect.
errinvdirfle	(148)	The specified directory is not a directory.
errinvdirstr	(149)	The specified directory name is syntactically incorrect.
errinvcloper	(173)	The device class is inappropriate for the operation.
errdirnotfnd	(177)	The specified directory does not exist.
errdirinvver	(200)	A directory file cannot have a version number greater than one.
errfilopen	(202)	The process tried to simultaneously open more than one tape file.

See Also:

_close	- Close a file
_create	- Create a file
_defprot	- Set default protection mask
_delete	- Delete a file
_open	- Open a file
_setfprt	- Set file protection

Assembler Calling Sequence:

```
push    fname                   ;address - file name
push    mode                    ;value - type of access requested
push    reclen                  ;value - record length
push    lun                     ;address - logical unit number
push    status                  ;address - result of the operation
jsr     _creats                 ;simplified file creation
```

C Function Declaration:

```
                                /* simplified file creation */
long                            /* returns result of the operation */
_creats (fname, mode, reclen, lun)
        char fname[94];         /* file name */
        long mode;              /* type of access requested */
        long reclen;            /* record length */
        long *lun;              /* logical unit number
```

FORTRAN Subroutine Declaration:

```
c                               ! simplified file creation
        subroutine _creats(fname, mode, reclen, lun, status)
            character*94 fname  ! file name
            integer*4 mode      ! type of access requested
            integer*4 reclen    ! record length
            integer*4 lun       ! logical unit number
            integer*4 status    ! result of the operation
```

Pascal Procedure Declaration:

```
procedure _creats(              {** simplified file creation}
        fname   : string[93];   {** file name}
        mode    : longint;      {** type of access requested}
        reclen  : longint;      {** record length}
    var lun     : longint;      {** logical unit number}
    var status  : longint       {** result of the operation}
); external;
```

Simplified create process.

Description:

This call is identical to _crproc except that several parameters are removed. It uses the default of each of the parameters left out.

Each process under control of the operating system must be created by a call to this operating system service routine (or to _crproc). When a process is created, it is called a child process. The process that created it is called its parent process.

This system call allows spawning of child processes. Spawned processes run in series. This means that the parent process hibernates while the child process runs. When the child process terminates, the parent process resumes. The completion status of the child is returned to the parent.

The calling process must have read privilege to the device containing the image file, execute privilege to all directories in the path leading to the directory containing the image file, read privilege to the directory containing the image file and execute privilege to the file containing the child process image for successful creation of the child process.

If the image file is specified by fcb.seq number then the process must have read privilege to the device containing the image file and execute privilege to the file containing the image.

The child process is created with the same privileges, at the same priority, with the same time slice, with the same user identification code, and the same standard input, output and error files as the parent process.

Related Privileges:

none	- Allows the parent process to create a child from an image file to which the parent has access as described above.
bypass	- Allows the parent process to create a child process independent of the file protection.

Parameters:

fname	- Address of a 94 byte null terminated string specifying the name of the file containing the

process image. This string will be translated automatically by the MCS to its logical equivalent. This string may contain up to 93 significant characters followed by a null.

pname	- Address of a 17 byte null terminated string containing the process name to be given the new process. This string is used for human identification. (16 significant characters plus a null)
cmd	- Address of the command line. The command line may contain up to 3072 significant bytes. The command line may contain any data whatever to be passed from the parent to the child. The data appears on the top of the child process's stack as the child process begins. The long word at the top of the child's stack is the length in bytes of the command line. At the location (USP+4) on the child's stack is a long word which contains the starting address of the command line.
cmdlen	- Length of the command line specified in bytes.
pid	- Address of a long word to receive the pid of the child process. If the address of the long word is zero, no value is returned.
ccode	- Address of a long word to receive the completion code returned to the parent by the process responsible for terminating the child process. If the child is exited as a result of a system violation (memory violation, illegal instruction, ...) the system supplies the ccode. If the process terminates normally, the process itself supplies the ccode. If the process is exited by another process, the other process supplies the ccode. If the address of the long word is zero, no value is returned. Completion codes that may be supplied by the system include:

erralarmexit	(28)	The system clock reached the value specified for _ALARM.
errzerodivtrap	(29)	The process has an undefined trap: Divide-by-zero.
errchktrap	(30)	The process has an undefined trap: CHK Instruction.
errtrapvtrap	(31)	The process has an undefined trap: TRAPV Instruction.
errtracetrap	(32)	The process has an undefined trap: TRACE.
err1010trap	(33)	The process has an undefined trap: 1010 Instruction.
err1111trap	(34)	The process has an undefined trap: 1111 Instruction.
errprivintrap	(35)	The process attempted to execute a privileged instruction.
errillintrap	(36)	The process attempted to execute an

		illegal instruction.
errbustrap	(37)	The process has a bus error.
erradrtrap	(38)	The process has an address error.
errnonexmem	(39)	The process attempted to access nonexistent memory.
errmemparity	(40)	The process has a memory parity-error.
errwriteprot	(41)	The process attempted to write to a write-protected page in memory.
errundeftrap	(42)	_SETTRP was not used to define a call for a trap other than TRAP 0.
errundefsvc	(43)	The MCS does not recognize the SVC number used by the process.
errcontccode	(255)	[CTRL] c terminated the process.

status - Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
errnomemavail	(7)	All available memory has been allocated.
errnotimfle	(21)	The specified file is not an image file.
errimagebmbad	(53)	(MCS error) The bitmap changed during the creation of the process.
errinvdevnam	(130)	The specified devicename is syntactically incorrect.
errundevnam	(131)	The MCS does not recognize the devicename. Is the device mounted?
errfilnotfnd	(133)	The specified file could not be found.
errreadleof	(140)	The process tried to read past the logical end of a file.
errnoexecpriv	(143)	The process does not have Execute Privilege for the file.
errnoreadpriv	(144)	The process does not have Read Privilege for the file.
errinvfnstr	(147)	The specified filename is syntactically incorrect.
errinvdirfle	(148)	The specified directory is not a directory.
errinvdirstr	(149)	The specified directory name is syntactically incorrect.
errdirnotfnd	(177)	The specified directory does not exist.
errfilopen	(202)	The process tried to simultaneously open more than one tape file.
		Device integrity errors

See Also:

_crproc	- Create a new process
_exproc	- Terminate the specified process
_setpnam	- Change process name
_setpri	- Change priority level
_settmsl	- Change scheduling time slice

_setuic - Set process uic

Assembler Calling Sequence:

```
push    fname                       ;address - name of image file
push    pname                       ;address - process name
push    cmd                         ;address - command line
push    cmdlen                      ;value - length of cmd
push    pid                         ;address - childs pid
push    ccode                       ;address - childs completion code
push    status                      ;address - result of the operation
jsr     _crprcs                     ;simplified create process
```

C function declaration:

```
                                    /* simplified create process */
long                                /* returns result of the operation */
_crprcs (fname, pname, cmd, cmdlen, pid, ccode)
        char fname[94];             /* name of image file */
        char pname[17];             /* process name */
        char cmd[3072];             /* command line */
        long cmdlen;                /* length of cmd */
        long *pid;                  /* childs pid */
        long *ccode;                /* childs completion code */
```

Fortran Subroutine Declaration:

```
c                                   ! simplified create process
        subroutine crprcs(fname, pname, cmd, cmdlen, pid,
     &          ccode, status)
            character*94 fname      ! name of image file
            character*17 pname      ! process name
            character*(*) cmd       ! command line
            integer*4 cmdlen        ! length of cmd
            integer*4 pid           ! childs pid
            integer*4 ccode         ! childs completion code
            integer*4 status        ! result of the operation
```

Pascal Procedure Declaration:

```
procedure _crprcs(                  {** simplified create process}
        fname   : string[93];       {** name of image file}
        pname   : string[16];       {** process name}
        cmd     : ^array_of_char;   {** command line}
        cmdlen  : longint;          {** length of cmd}
    var pid     : longint;          {** childs pid}
    var ccode   : longint;          {** childs completion code}
    var status  : longint           {** result of the operation}
); external;
```

Create a new process.

Description:

Each process under control of the operating system must be created by a call to this operating system service routine. When a process is created, it is called a child process. The process that created it is called its parent process.

SPAWN and FORK are two different modes of creation. Spawned processes run in series. This means that the parent process hibernates while the child process runs. When the child process terminates, the parent process resumes. The completion status of the child is returned to the parent.

Forked processes run in parallel. The parent process is not hibernated, but continues execution immediately after successful creation of the child process.

The calling process must have read privilege to the device containing the image file, execute privilege to all directories in the path leading to the directory containing the image file, read privilege to the directory containing the image file and execute privilege to the file containing the child process image for successful creation of the child process.

If the image file is specified by fcb.seq number then the process must have read privilege to the device containing the image file and execute privilege to the file containing the image.

Without the setpriv privilege, the child may not be given more privileges than the parent has.

The child process is created with the same default device and directory as the parent.

Related Privileges:

none	- Allows the parent process to create a child from an image file to which the parent has access as described above. The child may not be given privileges not possessed by the parent.
bypass	- Allows the parent process to create a child process independent of the file protection.
group	- Allows the parent process to create a child process with the same group id but a different owner id than the parent process has.
setpriv	- Allows the parent process to give the child

	process more privileges than those possessed by the parent.
setprior	- Allows the parent process to initiate a child at a higher priority level and/or with a higher time slice than the parent.
world	- Allows the parent process to create a child with any owner id and group id (uic) whatsoever.

Parameters:

mode - Whether the process is spawned or forked. A 0 indicates spawn, 1 indicates fork. All other values are reserved and should not be used.

siteid - The site id of the system on which the process is to be created. If the site id is zero, the process will be created on the same system as the calling process.

fname - Address of a 94 byte null terminated string specifying the name of the file containing the process image. This string will be translated into its logical equivalent. This string may contain up to 93 significant characters followed by a null.

pname - Address of a 17 byte null terminated string containing the process name to be given the new process. This string is used for human identification. (16 significant characters plus a null)

priv - The privilege mask contains a bit mask of privileges to be given to the child process. A -1 indicates that the child should receive the same privileges that the parent has. Privileges are bit encoded as follows:

Bit Name	Bit #	Description
pcbpvsetpriv	0	setpriv
pcbpvsystem	1	system
pcbpvreadphys	2	readphys
pcbpvwritephys	3	writephys
pcbpvsetprior	4	setprior
pcbpvchngsuper	5	chngsuper
pcbpvbypass	6	bypass
pcbpvoperator	7	operator
pcbpvaltuic	8	altuic
pcbpvworld	9	world
pcbpvgroup	10	group
	11-31	Reserved. Must be set to zero

priort - The priority level (0..3) at which the child process will execute. Level 0 is the highest priority. A minus one (-1) in this parameter means to use the same priority as the parent process.

tslice - The time slice value. The maximum amount of time the child process will be able to run each time it is scheduled. This time is specified in .01 milliseconds. (A time slice of 100 represents 1 millisecond) A minus one (-1) in this parameter means to use the same time slice as the parent process.

uic - The user identification code of the child process. The most significant 16 bits represent the owner id and the least significant 16 bits represent the group id. A minus one (-1) in this parameter means to use the same uic as the parent process.

sysin - Address of a 94 byte null terminated string containing the name of the standard input file for the child process. This string will be translated automatically by the MCS to its logical equivalent. The equivalent string will be assigned the logical name "SYS$INPUT" in the logical name table of the child process. The string passed is NOT checked for validity. It may contain up to 93 significant characters followed by a null.

sysout - Address of a 94 byte null terminated string containing the name of the standard output file for the child process. This string will be translated automatically by the MCS to its logical equivalent. The equivalent string will be assigned the logical name "SYS$OUTPUT" in the logical name table of the child process. The string passed is NOT checked for validity. It may contain up to 93 significant characters followed by a null.

syserr - Address of a 94 byte null terminated string containing the name of the standard error file for the

e

child process. This string will be translated automatically by the MCS to its logical equivalent. The equivalent string will be assigned the logical name "SYS$ERROR" in the logical name table of the child process. The string passed is NOT checked for validity. It may contain up to 93 significant characters followed by a null.

cmd - Address of the command line. (up to 3072 bytes) The command line may contain any data whatever to be passed from the parent to the child.

The data appears on the top of the child process's stack as the child process begins. The long word at the top of the child's stack is the length in bytes of the command line. At the location (USP+4) on the child's stack is a long word which contains the starting address of the command line.

cmdlen - Length of the command line specified in bytes.

pid - Address of a long word to receive the pid of the child process. Note that this is only valuable in the case that the child is forked. If the address of the long word is zero, no value is returned.

ccode - Address of a long word to receive the completion code returned to the parent by the process responsible for terminating the child process. If the child is exited as a result of a system violation (memory violation, illegal instruction, ...) the system supplies the ccode. If the process terminates normally, the process itself supplies the ccode. If the process is exited by another process, the other process supplies the ccode. Note that the ccode will always be zero for processes that are forked. If the address of the long word is zero, no value is returned. Completion codes that may be supplied by the system include:

erralarmexit	(28)	The system clock reached the value specified for _ALARM.
errzerodivtrap	(29)	The process has an undefined trap: Divide-by-zero.
errchktrap	(30)	The process has an undefined trap: CHK Instruction.
errtrapvtrap	(31)	The process has an undefined trap: TRAPV Instruction.
errtracetrap	(32)	The process has an undefined trap: TRACE.
err1010trap	(33)	The process has an undefined trap: 1010 Instruction.
err1111trap	(34)	The process has an undefined trap: 1111 Instruction.
errprivintrap	(35)	The process attempted to execute a privileged instruction.
errillintrap	(36)	The process attempted to execute an illegal instruction.
errbustrap	(37)	The process has a bus error.
erradrtrap	(38)	The process has an address error.
errnonexmem	(39)	The process attempted to access nonexistent memory.
errmemparity	(40)	The process has a memory parity-error.
errwriteprot	(41)	The process attempted to write to a write-protected page in memory.
errundeftrap	(42)	_SETTRP was not used to define a call for a trap other than TRAP 0.
errundefsvc	(43)	The MCS does not recognize the SVC number used by the process.
errcontccode	(255)	[CTRL] c terminated the process.

status - Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
errnomemavail	(7)	All available memory has been allocated.
errinvsiteid	(8)	The specified site id does not exist.

errnotimfle	(21)	The specified file is not an image file.
errimagebmbad	(53)	(MCS error) The bitmap changed during the creation of the process.
errinvdevnam	(130)	The specified devicename is syntactically incorrect.
errundevnam	(131)	The MCS does not recognize the devicename. Is the device mounted?
errfilnotfnd	(133)	The specified file could not be found.
errreadleof	(140)	The process tried to read past the logical end of a file.
errnoexecpriv	(143)	The process does not have Execute Privilege for the file.
errnoreadpriv	(144)	The process does not have Read Privilege for the file.
errinvfnstr	(147)	The specified filename is syntactically incorrect.
errinvdirfle	(148)	The specified directory is not a directory.
errinvdirstr	(149)	The specified directory name is syntactically incorrect.
errdirnotfnd	(177)	The specified directory does not exist.
errfilopen	(202)	The process tried to simultaneously open more than one tape file.
		Device integrity errors

See Also:

_crprcs	- Simplified create process
_exproc	- Terminate the specified process
_setpnam	- Change process name
_setpri	- Change priority level
_settmsl	- Change scheduling time slice
_setuic	- Set process uic

Assembler Calling Sequence:

```
push    mode                      ;value - spawn or fork
push    siteid                    ;value - system id
push    fname                     ;address - name of image file
push    pname                     ;address - process name
push    priv                      ;value - process privilege
push    priort                    ;value - process priority
push    tslice                    ;value - process time slice
push    uic                       ;value - user identification code
push    sysin                     ;address - standard input file
push    sysout                    ;address - standard output file
push    syserr                    ;address - standard error file
push    cmd                       ;address - command line
push    cmdlen                    ;value - length of cmd
push    pid                       ;address - childs pid
push    ccode                     ;address - childs completion code
push    status                    ;address - result of the operation
jsr     _crproc                   ;create a new process
```

C function declaration:

```
                                    /* create a new process */
long                                /* returns result of the operation */
_crproc (mode, siteid, fname, pname, priv, priort, tslice, uic, sysin,
                  sysout, syserr, cmd, cmdlen, pid, ccode)
        long mode;                  /* spawn or fork */
        long siteid;                /* system id */
        char fname[94];             /* name of image file */
        char pname[17];             /* process name */
        long priv;                  /* process privilege */
        long priort;                /* process priority */
        long tslice;                /* process time slice */
        long uic;                   /* user identification code */
        char sysin[94];             /* standard input file */
        char sysout[94];            /* standard output file */
        char syserr[94];            /* standard error file */
        char cmd[3072];             /* command line */
        long cmdlen;                /* length of cmd */
        long *pid;                  /* childs pid */
        long *ccode;                /* childs completion code */
```

Fortran Subroutine Declaration:

```
c                                   ! create a new process
        subroutine crproc(mode, siteid, fname, pname, priv,
      &         priort, tslice, uic, sysin, sysout, syserr, cmd,
      &         cmdlen, pid, ccode, status)
            integer*4 mode          ! spawn or fork
            integer*4 siteid        ! system id
            character*94 fname      ! name of image file
            character*17 pname      ! process name
            integer*4 priv          ! process privilege
            integer*4 priort        ! process priority
            integer*4 tslice        ! process time slice
            integer*4 uic           ! user identification code
            character*94 sysin      ! standard input file
            character*94 sysout     ! standard output file
            character*94 syserr     ! standard error file
            character*(*) cmd       ! command line
            integer*4 cmdlen        ! length of cmd
            integer*4 pid           ! childs pid
            integer*4 ccode         ! childs completion code
            integer*4 status        ! result of the operation
```

Pascal Procedure Declaration:

```
procedure _crproc(                  {** create a new process}
        mode    : longint;          {** spawn or fork}
        siteid  : longint;          {** system id}
        fname   : string[93];       {** name of image file}
        pname    : string[16];      {** process name}
        priv     : longint;         {** process privilege}
        priort   : longint;         {** process priority}
        tslice   : longint;         {** process time slice}
        uic      : longint;         {** user identification code}
        sysin    : string[93];      {** standard input file}
        sysout   : string[93];      {** standard output file}
        syserr   : string[93];      {** standard error file}
        cmd      : ^array_of_char; {** command line}
        cmdlen   : longint;         {** length of cmd}
    var pid      : longint;         {** childs pid}
    var ccode    : longint;         {** childs completion code}
    var status   : longint          {** result of the operation}
); external;
```

crshdp - Enable/disable crash display.

Description:

Enable or disable the crash display report when an error occurs in a process. The crash display report is the report that is generated by the system which shows the value of the processor registers, the system stack, user stack, etc.

When a process is created, crash displays are enabled. That is, if the process performs an invalid operation, e.g. accessing non-existent memory or executing an illegal instruction, the crash display will be written to the standard error file for that process.

Using this system call the programmer can specify that crash displays are to be suppressed. Having the crash display report disabled does not affect the normal cleanup, by WMCS, of a process when it performs an invalid operation.

Related Privileges:

None.

Parameters:

mode	- A flag indicating whether the crash display report is to be enabled or disabled. A value of 0 will disable crash display reports, a non-zero value will enable crash display reports.

Diagnostics:

None.

See Also:

None.

Assembler Calling Sequence:

```
push    mode                       ;value - enable or disable
jsr     _crshdp                    ;enable/disable crash display
```

C Function Declaration:

```
                                        /* enable/disable crash display */
    void                                /* no result */
    _crshdp(mode)
            long mode;                  /* enable or disable
```

Fortran Subroutine Declaration:

```
    c                                    ! enable/disable crash display
            subroutine crshdp(mode)
                    integer*4 mode       ! enable or disable
```

Pascal Procedure Declaration:

```
    procedure  crshdp(                   {** enable/disable crash display}
            mode    : longint;           {** enable or disable}
    ); external;
```

Set/clear control-c protection.

Description:

Enable or disable process termination upon receipt of a CTRL/C character.

Any process which accesses a standard terminal port (using _open, _read, _write, _create, _exproc or _crproc) will be asynchronously exited if a CTRL/C character is received from the terminal. This system call enables or disables this feature.

By default when a process is created the control c protection is disabled, i.e. the process will be deleted if control c is pressed.

Note that terminals also have a control C feature that determines whether control C characters should be passed on to the application program. In order for a process to terminate when control C is pressed, The process must have been the last process to have accessed the terminal, the terminal must be set to "CONTROL C" status and the process must not be control C protected.

Related Privileges:

None.

Parameters:

mode	- A flag indicating whether the process is to be control C protected. A 0 indicates that the process is not protected, i.e. it will be deleted when control C is pressed.

Diagnostics:

none.

See Also:

_getdst - Get device status
_setdst - Set device status

Assembler Calling Sequence:

```
push    mode                        ;value - protect or unpr[illegible]ct
```

```
    jsr     _ctrlc                      ;set/clear control c protection
```

C function declaration:

```
                                        /* set/clear control c protection */
    void                                /* no result */
    _ctrlc(mode)
            long mode                   /* protect or unprotect */
```

Fortran Subroutine Declaration:

```
    c                                   ! set/clear control c protection
            subroutine ctrlc(mode)
                integer*4 mode          ! protect or unprotect
```

Pascal Procedure Declaration:

```
    procedure _ctrlc(                   {** set/clear control c protection}
            mode    : longint           {** protect or unprotect}
    ); external;
```

Disconnect all remote connections this process has.

Description:

This system call is used to break all logical connections with remote machines. It does this by deallocating the network links (virtual circuits) to the process created by the _connect system call.

Related Privileges:

None.

Parameters:

status	- Address of a long word to receive the result of the operation.

Diagnostics:

None.

See Also:

_connect - Make a remote connection
_disconn - Break a remote connection
_dconidl - Break all idle remote connections

Assembler Calling Sequence:

```
push    status                      ;address - result of the operation
jsr     _dconall                    ;break all remote connections
```

C Function Declaration:

```
                                    /* break all remote connections */
long                                /* returns result of the operation */
_dconall();
```

FORTRAN Subroutine Declaration:

```
     c                                ! break all remote connections
              subroutine _dconal(status)
                  integer*4 status    ! result of the operation
```

Pascal Procedure Declaration:

```
     procedure _dconall(              {** break all remote connections }
         var status  : longint        {** result of the operation }
     ); external;
```

Disconnect the idle remote connections this process has.

Description:

This system call is used to break all logical connections that are currently idle. It does this by deallocating the network links (virtual circuits) to the process created by the _connect system call. A connection is considered idle if no files are open on the remote system and if your default directory is not on the remote system.

Related Privileges:

None.

Parameters:

status	- Address of a long word to receive the result of the operation.

Diagnostics:

None.

See Also:

_connect - Make a remote connection
_disconn - Break a remote connection
_dconall - Break all remote connections

Assembler Calling Sequence:

```
     push    status                     ;address - result of the operation
     jsr     _dconidle                   ;break all idle remote connections
```

C Function Declaration:

```
                                        /* break all idle remote connections */
     long                                 /* returns result of the operation */
   _dconidle();
```

FORTRAN Subroutine Declaration:

```
     c                                  ! break all idle remote connections
               subroutine _dconid(status)
                   integer*4 status     ! result of the operation
```

Pascal Procedure Declaration:

```
     procedure _dconidle(               {** break all idle remote connections }
         var status  : longint          {** result of the operation }
     ); external;
```

Deallocate an allocated device.

Description:

This SVC is used to deallocate a device which was previously allocated using the _alloc SVC.

Related Privileges:

none	- Allows deallocation of a device which is currently allocated to a process with the same owner id and group id (uic) as the calling process.
group	- Allows deallocation of a device which is allocated to a process with the same group id but a different owner id than the calling process.
world	- Allows deallocation of a device allocated to any process whatsoever.

Parameters:

dname	- Address of a null terminated string identifying the specific device which is to be deallocated. This string will be translated automatically by WMCS into its logical equivalent. The string may contain up to 93 significant characters followed by a null, but must translate to a valid device name of not more than 27 characters (16-character nodename with two underscores and an 8-character devicename with one underscore and a null).
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
errnotalloc	(16)	The specified device is not allocated.
errnamenull	(80)	The specified name must not be null.

See Also:

_alloc	- Allocate a device
_getalc	- Get names of allocated devices
_getrel	- Get names of rotor list elements
_getrtr	- Get rotor list names
_setrtr	- Assign device names to a rotor list

Assembler Calling Sequence:

```
push    dname                       ; address - device name
push    status                      ; address - result of the operation
jsr     _dealloc                    ; deallocate an allocated device
```

C Function Declaration:

```
                                    /* deallocate an allocated device */
long                                /* returns result of the operation */
_dealloc(dname)
        char    dname[94];          /* device name */
```

FORTRAN Subroutine Declaration:

```
c                                   ! deallocate an allocated device
        subroutine _deallo(dname,status)
            character*94 dname ! device name
            integer*4 status   ! result of the operation
```

Pascal Procedure Declaration:

```
procedure _dealloc(                 {** deallocate an allocated device}
        dname   : string[93]; {** device name}
    var status  : longint     {** result of the operation}
); external;
```

defdprt - Set default device protection.

Description:

Establishes the default protection to be applied to a device. The default protection is the protection that is assigned to a device when the device is not being referenced by any process.

Device protection can be assigned with the setdprt system call. But, as soon as the device is not being referenced (no process has the device, or any file on the device open) the protection reverts back to the most recently defined default protection.

If no default protection has been assigned, the protection of the device does not change when the device is not referenced.

This operation is valid for any mounted device.

To successfully change protection on a device the process must have operator privilege or bypass privilege.

Related Privileges:

None	- The process can not change the default protection of a device.
bypass	- Allows the process to change the default protection on any device.
operator	- Allows the process to change the default protection on any device.

Parameters:

dname	- Address of a null terminated string containing the the name of the device whose protection is to be set. This string may contain up to 93 significant characters followed by a null. This string will be translated automatically by the MCS to its logical equivalent. If this string contains a file designation, the devicename portion of the file designation is used for this parameter.
prot	- File protection mask. The least significant 16 bit word of this parameter is divided into 4 nibbles. Each nibble corresponds to a class of users. The bits within each nibble represent the type of access that class of user is granted for this device. If the bit is set (1) the access

is granted.

From the least to the most significant nibble the user classes are:

Ownr	- device owner
Grp	- processes with the same group id as the owner
Pub	- all other processes in the system
Sys	- processes with SYSTEM privilege

```
  Sys  Pub  Grp  Ownr
 |----|----|----|----|
 |DWRE|DWRE|DWRE|DWRE|
 |-------------------|
MSB                 LSB
```

From the least to the most significant bits within the nibbles, the access privileges are:

E	- Execute access
R	- Read access
W	- Write access
D	- Delete access

A long word -1 ($FFFFFFFF) is a reserved value that means that the user's default protection mask is to be used.

status - Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
errinvdevnam	(130)	The specified devicename is syntactically incorrect.
errundevnam	(131)	The MCS does not recognize the devicename. Is the device mounted?

Device integrity errors

See Also:

_defprot - Set default protection mask
_getdprt - Get device protection
_getfprt - Get file protection
_setdprt - Set device protection
_setfprt - Set file protection

Assembler Calling Sequence:

```
push    dname                       ;address - device name
push    prot                        ;value - protection mask
push    status                      ;address - result of the operation
jsr     _defdprt                    ;set default device protection
```

C function declaration:

```
                                    /* set default device protection */
long                                /* returns result of the operation */
_defdprt(dname, prot)
        char dname[94];             /* device name */
        long prot;                  /* protection mask */
```

Fortran Subroutine Declaration:

```
c                                    ! set default device protection
        subroutine defdpr(dname, prot, status)
            character*94 dname       ! device name
            integer*4 prot           ! protection mask
            integer*4 status         ! result of the operation
```

Pascal Procedure Declaration:

```
procedure  defdprt(                  {** set default device protection}
        dname   : string[93];        {** device name}
        prot    : longint;           {** protection mask}
    var status  : longint            {** result of the operation}
); external;
```

defduic - Set default device UIC.

Description:

This system call allows a process to change the default user identification code (uic) of a device. Given the correct privileges a process can change the uic of a device with the _setduic svc. As soon as no processes have a device open, it's uic will revert back to this default value. When a device is first mounted the default device uic value is the same as the device uic. By changing the uic the ownership of the device is changed.

To successfully change the uic of a device, either the device must have the UNOWNED uic ([0000,0001]) or the calling process must have operator privilege, and either group privilege or world privilege.

If the calling process has group privilege, and the group id of the device is the same as the group id of the calling process, the process can modify the owner id of the device.

If the calling process has world privilege and operator privilege it can change the uic of any device to be any other uic except zero.

This system call is valid for any class of device.

Related Privileges:

none	- If the device has the UNOWNED uic ([0000,0001]) the process can change the uic of the device to the same uic as the calling process.
group	- If the process also has operator privilege, it can modify the owner id of any mounted device which has the same group id as the calling process. If the process does not have operator privilege but the device has the UNOWNED uic ([0000,0001]) the process can set the group id to it's own group id, and it can set the owner id to any value.
operator	- Allows setting the uic if the process also has either group or world privilege.
world	- If the process also has operator privilege, it can modify the uic of any mounted device to any other uic except zero. If the process does not have operator privilege but the device has the UNOWNED

uic ([0000,0001]) the process can set the uic of the device to any other uic except zero.

Parameters:

dname	- Address of a null terminated string containing the name of the device whose uic is to be changed. This string will be translated automatically by the MCS to its logical equivalent. This string may contain up to 93 valid characters followed by a null byte. If this string contains a file designation, the devicename portion of the file designation is used for this parameter.
uic	- A long word containing the user identification code. This long word is divided into two fields. The most significant 16 bits constitute the owner id number. The least significant 16 bits constitute the group id number (identifying the group to which the user belongs). The value $FFFFFFFF (-1) is a reserved value that means to use the default uic, i.e. the uic of the calling process. A value of zero is invalid.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
errinvdevnam	(130)	The specified devicename is syntactically incorrect.
errundevnam	(131)	The MCS does not recognize the devicename. Is the device mounted?

See Also:

_getduic - Get device uic
_getfuic - Get file uic
_getuic - Get process uic
_setduic - Set device uic.
_setfuic - Set file uic
_setuic - Set process uic

Assembler Calling Sequence:

```
push    dname                       ;address - device name
push    uic                         ;value - owner id code
push    status                      ;address - result of the operation
```

```
    jsr     _defduic                    ;set default device uic
```

C Function Declaration:

```
                                        /* set default device uic */
    long                                /* returns result of the operation */
    _defduic(dname, uic)
            char dname[94];             /* device name */
            long uic;                   /* owner id code */
```

Fortran Subroutine Declaration:

```
    c                                   ! set default device uic
            subroutine defdui(dname, uic, status)
                character*94 dname      ! device name
                integer*4 uic           ! owner id code
                integer*4 status        ! result of the operation
```

Pascal Procedure Declaration:

```
    procedure  defduic(                 {** set default device uic}
            dname   : string[93];       {** device name}
            uic     : longint;          {** owner id code}
        var status  : longint           {** result of the operation}
    ); external;
```

defmem - Define named shared memory area.

Description:

Named sharable memory areas are created with defmem. Named sharable memory areas are sections of system memory which have an associated name. Using this name, a process may request that this section of memory be mapped into its logical memory space which extends from address $00001000 through address $001fefff. The size of these memory areas will be some multiple of the hardware page size which is 4K bytes.

A process which wants to create a named sharable memory area must first have allocated the memory to itself. This may have happened at initial program load time or the process may use the normal memory allocation routines to cause additional system memory to be mapped into empty portions of his logical address space. After having initialized this memory space, the process calls _defmem to make this memory space to available to other processes.

After having called _defmem. the named sharable memory area is defined and has one process, that of the definer, which references it. At the time that no more processes reference the named sharable memory area, the system will deallocate the memory and return it to the free memory list. If desirable, the linger bit may be set which will cause the named sharable memory area to remain defined even though no process references it. In this case, an explicit call to _udefmem is needed to deallocate the memory area.

Related Privileges:

None	- The defined memory area may not have a uic other than that of the calling process.
group	- Allows the process to define a memory area with the same group id but a different owner id than the calling process.
world	- Allows the process to define a memory area with any uic.

Parameters:

mname	- Address of a null terminated string containing the name to be assigned to the memory area. This string will be translated automatically by WMCS into its logical equivalent. This string may

contain up to 93 significant characters followed by a null.

adr	- A long word containing the location in local user logical memory where the shared memory area will start.
size	- A long word containing the length in bytes of the new memory area. The value saved in the control structure will be rounded up to the hardware page size.
uic	- A long word containing the user identification code (uic) specifying the owner of the memory area. The most significant 16 bits of this parameter contain the owner id while the least significant 16 bits contain the group id. A value of $FFFFFFFF (-1) is a reserved value which means to give the memory are the same uis as the calling process.
prot	- File protection mask. The least significant 16 bit word of this parameter is divided into 4 nibbles. Each nibble corresponds to a class of users. The bits within each nibble represent the type of access that class of user is granted for this memory area. If the bit is set (1) the access is granted.

From the least to the most significant nibble the user classes are:

Ownr	- memory area owner
Grp	- processes with the same group id as the owner
Pub	- all other processes in the system
Sys	- processes with SYSTEM privilege

```
Sys  Pub  Grp  Ownr
|----|----|----|----|
|DWRE|DWRE|DWRE|DWRE|
|-------------------|
MSB                LSB
```

From the least to the most significant bits within the nibbles, the access privileges are:

E	- Execute access
R	- Read access
W	- Write access
D	- Delete access

The value $FFFFFFFF (-1) is a reserved value that means that the users default protection mask is to be used.

```
mode            - A long word which contains the linger bit which
                  allows the memory area remain even though no one
                  is currently referencing it.
                  BIT #  NAME          DESCRIPTION
                    0    linger        NSM remains defined after
                                       process dies.
status          - Address of a long word to receive the result of
                  the operation.
```

Diagnostics:

```
errinsufpriv   (  1) The process lacks the privileges required
                      to perform the operation.
errnonowned    (  6) Attempt to affect non-owned memory.
errsizovfl     ( 60) The size passed to the MCS is out of range.
errnamenull    ( 80) The name specified must not be null.
errnameexists  ( 81) The name specified already exists.
```

See Also:

```
_udefmem  -  Undefine a named sharable memory area.
_shrmem   -  Share a named sharable memory area.
_ushrmem  -  Unshare a named sharable memory area.
_getmlst  -  Get a list of named sharable memory areas.
_setmuic  -  Change owner of a named sharable memory area.
_setmprt  -  Change protection of a named sharable memory area.
```

Assembler Calling Sequence:

```
push    mname                   ; address - memory area name
push    adr                     ; value   - address of memory area
push    size                    ; value   - size of memory area
push    uic                     ; value   - user identification code
push    prot                    ; value   - memory area protection
push    mode                    ; value   - mode flags
push    status                  ; address - result of the operation
jsr     _defmem                 ; define named shared memory area
```

C Function Declaration:

```
                                /* define named shared memory area */
long                            /* returns result of the operation */
_defmem(mname,adr,size,uic,prot,mode)
    char        mname[94];      /* memory area name */
    long        adr;            /* address of memory area */
    long        size;           /* size of memory area */
    long        uic;            /* user identification code */
    long        prot;           /* memory area protection */
```

```
        long        mode;           /* mode flags */
```

FORTRAN Subroutine Declaration:

```
    c                               ! Define named shared memory area
            subroutine defmem(mname, adr, size, uic, prot, mode, status)
                character*94 mname  ! memory area name
                integer*4    adr    ! address of memory area
                integer*4    size   ! size of memory area
                integer*4    uic    ! user identification code
                integer*4    prot   ! memory area protection
                integer*4   mode    ! mode flags
                integer*4   status  ! result of the operation
```

PASCAL Procedure Declaration:

```
    procedure _defmem(              {** define named shared memory area}
            mname   : string[93];   {** memory area name}
            adr     : longint;      {** address of memory area }
            size    : longint;      {** size of memory area }
            uic     : longint;      {** user identification code}
            prot    : longint;      {** memory area protection}
            mode    : longint;      {** mode flags }
        var status  : longint       {** result of the operation}
    ); external;
```

Set default protection mask.

Description:

Specifies to the system the protection to be applied to newly created files when the _create 'prot' parameter is (-1). This mask will be used for any files created by the current process and any child processes of the current process.

Related Privileges:

None.

Parameters:

prot - File protection mask. The least significant 16 bit word of this parameter is divided into 4 nibbles. Each nibble corresponds to a class of users. The bits within each nibble represent the type of access that class of user is granted for this file. If the bit is set (1) the access is granted.

From the least to the most significant nibble the user classes are:

Ownr	- file owner
Grp	- processes with the same group id as the owner
Pub	- all other processes in the system
Sys	- processes with SYSTEM privilege

```
  Sys  Pub  Grp  Ownr
 |----|----|----|----|
 |DWRE|DWRE|DWRE|DWRE|
 |-------------------|
MSB                 LSB
```

From the least to the most significant bits within the nibbles, the access privileges are:

E	- Execute access
R	- Read access
W	- Write access
D	- Delete access

Diagnostics:

None.

See Also:

_create - Create a file
_creats - Simplified file creation
_getprot - Get default protection mask
_setfprt- Set file protection

Assembler Calling Sequence:

```
push    prot                    ;value - protection mask
jsr     _defprot                ;set default protection mask
```

C Function Declaration:

```
                                /* set default protection mask */
void                            /* no result */
_defprot ( prot )
        long prot;              /* protection mask */
```

Fortran Subroutine Declaration:

```
c                               ! set default protection mask
        subroutine defpro(prot)
            integer*4 prot      ! protection mask
```

Pascal Procedure Declaration:

```
procedure _defprot(             {** set default protection mask}
        prot    : longint       {** protection mask}
); external;
```

Deinstall privileged file.

Description:

This call is used to remove entries from the system table of installed files. Once a file is deinstalled, it will execute with only those privileges owned by the user. That is, it will not have any special privileges.

Related Privileges:

none	- The process is not allowed to deinstall privileged files.
operator	- The process can deinstall any installed file.

Parameters:

siteid	- The site id of the system on which the file is currently installed. If the value of this parameter is zero, the system on which the calling process is running is assumed.
fname	- The name of the file that you wish to deinstall.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
errinvsiteid	(8)	The specified site id does not exist.
erridxrange	(56)	The table ends before the specified occurrence.
errinvvernum	(129)	A file's version number cannot be greater than 65535.
errinvdevnam	(130)	The specified devicename is syntactically incorrect.
errundevnam	(131)	The WMCS does not recognize the devicename. Is the device mounted?
errinvfnstr	(147)	The specified filename is syntactically incorrect.
errinvdirfle	(148)	The specified directory is not a directory-type file.
errinvdirstr	(149)	The specified directory name is syntactically incorrect.
errdirnotfnd	(177)	The specified directory does not exist.

See Also:

_getinst - Get installed privileged file
_install - Install privileged file

Assembler Calling Sequence:

```
push    siteid                  ;value - the system id
push    fname                   ;value - file to deinstall
push    status                  ;address - result of the operation
jsr     _deinst                 ;deinstall privileged file
```

C Function Declaration:

```
                                /* deinstall privileged file */
long                            /* returns result of the operation */
_deinst (siteid, fname)
        long siteid;            /* the system id */
        char fname[94];         /* file to deinstall */
```

FORTRAN Subroutine Declaration:

```
c                               ! deinstall privileged file
        subroutine _deinst(siteid, lun, status)
            integer*4 siteid    ! the system id
            character*94 fname  ! file to deinstall
            integer*4 status    ! result of the operation
```

Pascal Procedure Declaration:

```
procedure _deinst(              {** deinstall privileged file}
        siteid  : longint;      {** the system id}
        fname   : string[93];   {** file to deinstall}
    var status  : longint       {** result of the operation}
); external;
```

Delete a file.

Description:

The named file is removed from the file structure, freeing the space it had consumed. In the absence of an explicit version number, the file with the highest version number is deleted.

This call will result in the file being marked for deletion, but the file will not actually be deleted until it is closed by all processes.

Tape files cannot be deleted.

Unless the process has bypass privilege, it must have read and write privilege to the device containing the file, it must have execute privilege of all directories in the path leading to the file, it must have read and write privilege to the directory containing the file, and delete privilege to the file itself in order for the file to be successfully deleted.

If the fname is specified in fcb.seq number format, the process must have read and write privilege to the device, read and write privilege to the directory containing the file and delete privilege to the file itself.

Related Privileges:

None	- Allows deletion only if process has access to the file as described above.
altuic	- Allows deletion if the owner of image file for the current process has access to the file as described above.
bypass	- Allows the process to delete the file independent of the file protection.
system	- Allows deletion if the system has access to the file as described above.

Parameters:

fname	- Address of a null terminated string containing the name of the file to be deleted. This string will be translated automatically by the MCS into its logical equivalent. This string may contain up to 93 significant characters followed by a null.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinvdevnam	(130)	The specified devicename is syntactically incorrect.
errundevnam	(131)	The MCS does not recognize the devicename. Is the device mounted?
errfilnotfnd	(133)	The specified file could not be found.
errnoexecpriv	(143)	The process does not have Execute Privilege for the file.
errnoreadpriv	(144)	The process does not have Read Privilege for the file.
errnowritepriv	(145)	The process does not have Write Privilege for the file.
errnodelpriv	(146)	The process does not have Delete Privilege for the file.
errinvfnstr	(147)	The specified filename is syntactically incorrect.
errinvdirfle	(148)	The specified directory is not a directory.
errinvdirstr	(149)	The specified directory name is syntactically incorrect.
erropendel	(153)	The specified file is open, has been marked for deletion.
errdelfile	(158)	System files cannot be deleted.
errinvcloper	(173)	The device class is inappropriate for the operation.
errdirnotfnd	(177)	The specified directory does not exist.
errinvseqnum	(178)	The file's FCB.SEQ number in the directory file is incorrect.
		Device integrity errors

See Also:

_close - Close a file
_create - Create a file
_open - Open a file

Assembler Calling Sequence:

```
push    fname                     ;address - file name
push    status                    ;address - result of the operation
jsr     _delete                   ;delete a file
```

C function declaration:

```
                                  /* delete a file */
long                              /* returns result of the operation */
_delete(fname)
        char fname[94];           /* file name */
```

Fortran Subroutine Declaration:

```
c                                 ! delete a file
```

```
          subroutine delete(fname, status)
              character*94 fname    ! file name
              integer*4 status      ! result of the operation
```

Pascal Procedure Declaration:

```
    procedure _delete(                  {** delete a file}
            fname   : string[93];       {** file name}
        var status  : longint           {** result of the operation}
    ); external;
```

Break a connection to a remote machine.

Description:

This system call is used to break a logical connection with a remote machine. It does this by deallocating the network link (virtual circuit) to the process created by the _connect system call.

Related Privileges:

None.

Parameters:

siteid	- Site ID of the system with which a connection is being broken.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinvsiteid	(8)	The specified site ID does not exist.
errremotelogon	(47)	The process was not allowed to log on to the remote system

See Also:

_connect - Make a remote connection
_dconall - Break all remote connections
_dconidl - Break all idle remote connections

Assembler Calling Sequence:

```
push    siteid                  ;value - site being disconnected
push    status                  ;address - result of the operation
jsr     _disconn                ;break a remote connection
```

C Function Declaration:

```
                                        /* break a remote connection */
    long                                /* returns result of the operation */
    _disconn(siteid);
            long siteid;                /* site being disconnected */
```

FORTRAN Subroutine Declaration:

```
    c                                   ! break a remote connection
            subroutine _discon(siteid, status)
                integer*4 siteid        ! site being connected to
                integer*4 status        ! result of the operation
```

Pascal Procedure Declaration:

```
    procedure _disconn(                 {** break a remote connection }
            siteid   : longint;         {** site being disconnected }
        var status   : longint          {** result of the operation }
    ); external;
```

Dismount a logical device.

Description:

Removes a device from further consideration by the O.S. A device cannot be dismounted if it contains open files.

After the device is dismounted, if the device driver is no longer needed (no other similar devices are mounted), the device driver is discarded and the space it occupied is returned to the system dynamic memory pool.

The process dismounting a user device must have either delete privilege to the device, or bypass privilege.

Related Privileges:

None	- Allows dismounting of devices for which the process has delete privilege
bypass	- Allows dismounting of any device

Parameters:

dname	- Address of null terminated string containing the name of the device to be dismounted. This string will be translated automatically by the MCS into its logical equivalent. This string may contain up to 93 significant characters followed by a null. If this string contains a file designation, the devicename portion of the file designation is used for this parameter.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
errinvdevnam	(130)	The specified devicename is syntactically incorrect.
errundevnam	(131)	The MCS does not recognize the devicename. Is the device mounted?
errnodelpriv	(146)	The process does not have Delete Privilege for the file.
errfilesopen	(160)	The device cannot be dismounted because files are still open on it.
errdiffbtblk	(168)	The boot block has changed since the device was mounted.

See Also:

```
_flush  - Flush I/O buffers to the device
_getdnam- Get device name
_mount  - Mount a logical device
```

Assembler Calling Sequence:

```
push    dname                     ;address - device name
push    status                    ;address - result of the operation
jsr     _dismnt                   ;dismount a logical device
```

C function declaration:

```
                                  /* dismount a logical device */
long                              /* returns result of the operation */
_dismnt (dname)
        char dname[94];           /* device name */
```

Fortran Subroutine Declaration:

```
c                                 ! dismount a logical device
        subroutine dismnt(dname, status)
            character*94 dname    ! device name
            integer*4 status      ! result of the operation
```

Pascal Procedure Declaration:

```
procedure _dismnt(                {** dismount a logical device}
        dname   : string[93];     {** device name}
    var status  : longint         {** result of the operation}
); external;
```

Duplicate a logical unit number of a file.

Description:

Given a valid logical unit number (lun), duplicate it. That is, make the file accessible via a new lun. Both the original and the new lun share the same characteristics and position in the file.

Related Privileges:

None.

Parameters:

lun	- Logical unit number to duplicate.
newlun	- The new duplicate logical unit number.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errnomemavail	(7)	All available memory has been allocated.
errinvlfn	(132)	The logical unit number does not correspond to an open file.

See Also:

_create	- Create a file
_open	- Open a file

Assembler Calling Sequence:

```
push    lun                     ;value - logical unit number
push    newlun                  ;address - new logical unit number
push    status                  ;address - result of the operation
jsr     _duplun                 ;duplicate an existing lun
```

C Function Declaration:

```
                                /* duplicate an existing lun */
long                            /* returns result of the operation */
_duplun (lun, newlun)
        long lun;               /* logical unit number */
        long *newlun;           /* new logical unit number */
```

FORTRAN Subroutine Declaration:

```
c                                ! duplicate an existing lun
          subroutine _duplun(lun, newlun, status)
              integer*4 lun      ! logical unit number
              integer*4 newlun   ! new logical unit number
              integer*4 status   ! result of the operation
```

Pascal Procedure Declaration:

```
procedure _duplun(               {** duplicate an existing lun}
        lun     : longint;       {** logical unit number}
    var newlun  : longint;       {** new logical unit number}
    var status  : longint        {** result of the operation}
); external;
```

errno - Receive process abort reason.

Description:

Obtain the process abort reason from the process control block (pcb) for any process in the system.

This call is most useful if called from an exit handler. With this svc a process can obtain the reason it entered its exit handler, i.e. the reason it is being terminated.

The value will be zero if the process has not terminated yet.

Related Privileges:

none	- Allows process to obtain the abort reason for any process with the same owner id and group id (uic) as the calling process.
group	- Allows process to obtain the abort reason for any process with the same group id as the calling process.
world	- Allows process to obtain the abort reason for any process in the system.

Parameters:

pid	- Process ID of the process whose abort reason is to be obtained. 0 refers to the calling process, -1 refers to the parent of the calling process.
reason	- Address of a long word to receive the reason the given process terminated. This value will be zero if the process has not terminated yet.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
errprcsnotfnd	(2)	The specified process is not in the system process table.

See Also:

_setexit - Define exit handler.

Assembler Calling Sequence:

```
    push    pid                         ;value - process id
    push    reason                      ;address - receives abort reason
    push    status                      ;address - result of the operation
    jsr     _errno                      ;receive process abort reason
```

C function declaration:

```
                                        /* receive process abort reason
    long                                /* returns result of the operation */
    _errno(pid, reason)
            long pid;                   /* process id */
            long *reason;               /* receives abort reason */
```

Fortran Subroutine Declaration:

```
    c                                   ! receive process abort reason
            subroutine errno(pid, reason, status)
                integer*4 pid           ! process id
                integer*4 reason        ! receives abort reason
                integer*4 status        ! result of the operation
```

Pascal Procedure Declaration:

```
    procedure  errno(                   {** receive process abort reason}
            pid     : longint;          {** process id}
        var reason  : longint;          {** receives abort reason}
        var status  : longint           {** result of the operation}
    ); external;
```

Define a returnable exit handler.

Description:

The user may define an exit handler to be executed when the process is deleted. An exit handler can be used as a cleanup and restore routine or as a mechanism for "catching" otherwise fatal errors. Use of this SVC also allows a process to return to the point from which the process was exited instead of merely altering the path to final exit. The return feature allows processes to use the exit handler as a software interrupt routine. Other processes send the interrupt using the _exproc system call and mutually recognized abort codes.

Return code values from -65535 to -4096 are for users to define as they wish. Values from -4095 to +4095 are reserved for WMCS. Values from +4096 to +65535 are also for users to define. The abort code can be determined using the _errno system call. Exit routines cannot have any call arguments.

The exit handler for a process is executed when a process exits regardless of the cause or circumstances of the exit. The exit handler is executed in the same processor mode (user or supervisor mode) as the mode from which the exit handler was defined.

When control is passed to the exit handler the OS notes that the process is executing its exit handler. If a fatal process error occurs while the process is executing its exit handler, the process will be deleted without passing through the exit handler again. If the process wants an exit handler to be called again as the process exits, it must define a new exit handler while it is executing its exit handler. Since no further abort conditions will be honored until the next time the process is scheduled, a carefully written exit handler can determine the reason for being transferred to the exit handler and be able to define a new one if necessary.

To terminate the process normally once the exit handler has been called, issue a call to _exproc from within the exit handler.

When a returnable exit handler is called, the registers contain the context of the process at the point it was interrupted. The top of the stack contains a return address to a piece of runtime code which will execute an RTR or RTE instruction upon return from the exit handler. The actual return address and status register of the interrupted process are stored at 6 and 4 bytes respectively from the top of the stack. Because an exithandler is capable of being called asynchronously in relation to the main process, changing global variables from within an exit handler may cause seemingly mysterious results when control is returned to the main body of a process which uses those same variables.

Related Privileges:

None.

Parameters:

adr	- Address of the first executable instruction of the exit handler to be called upon process exit.

Diagnostics:

None.

See Also:

_errno	- Receive process abort reason
_exproc	- Terminate the specified process
_setexit	- Define an exit handler

Assembler Calling Sequence:

```
push    adr                     ;exit handler address
jsr     _exitrtn                ;define a returnable exit handler
```

C Function Declaration:

```
                                /* define a returnable exit handler */
void                            /* no status is returned */
_exitrtn (adr)
         long adr;              /* exit handler address */
```

FORTRAN Subroutine Declaration:

```
c                               ! define a returnable exit handler
         subroutine _exitrt(adr)
             external adr       ! name of exit hander process
```

Pascal Procedure Declaration:

```
procedure _exitrtn(             {** define a returnable exit handler}
        adr     : longint       {** exit handler address}
); external;
```

Terminate the specified process.

Description:

The specified process is terminated, returning a 32-bit return code to the parent of the terminated process. The return code is received in the ccode parameter of the _crproc system call.

Return code values from -65535 to -4096 are for users to define as they wish. Values from -4095 to +4095 are reserved for WMCS. Values from +4096 to +65535 are also for users to define.

If the terminated process has an exit handler defined, it can request the "result" parameter using the _errno system call.

Related Privileges:

none	- Allows termination of any process with the same owner id and group id (uic) as the calling process
group	- Allows termination of any process with the same group id as the calling process
world	- Allows termination of any process in the system

Parameters:

pid	- The process id (pid) of the process to be terminated A process id of 0 represents the current process. A process id of -1 represents the parent of the current process.
result	- 32 bit result returned to the parent of the terminated process.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
errprcsnotfnd	(2)	The specified process is not in the system process table.

See Also:

```
_crprcs  - Simplified create process
_crproc  - Create a new process
_exitrtn - Define a returnable exit handler
_setexit - Define exit handler
```

Assembler Calling Sequence:

```
push    pid                     ;value - process id
push    result                  ;value - return code
push    status                  ;address - result of the operation
jsr     _exproc                 ;terminate the specified process
```

C Function Declaration:

```
                                /* terminate the specified process */
long                            /* returns result of the operation */
_exproc (pid, result)
        long pid;               /* process id */
        long result;            /* return code */
```

FORTRAN Subroutine Declaration:

```
c                                ! terminate the specified process
        subroutine _exproc(pid, result, status)
            integer*4 pid        ! process id
            integer*4 result     ! return code
            integer*4 status     ! result of the operation
```

Pascal Procedure Declaration:

```
procedure _exproc(              {** terminate the specified process}
        pid     : longint;      {** process id}
        result  : longint;      {** return code}
    var status  : longint       {** result of the operation}
); external;
```

Flush I/O buffers to the device.

Description:

Write all of the modified device cache buffers and modified file control blocks (fcb's) to the device, making the file system on the device current.

Requires that the process have write privilege to the device being flushed.

Related Privileges:

None	- Allows a process with write privilege to the device to flush the buffers.
bypass	- Allows a process to flush the buffers independent of the file protection.
operator	- Allows a process to flush the buffers independent of the file protection.

Parameters:

dname	- Address of a null terminated string containing the name of the device to be flushed. This string is translated automatically by the MCS into its logical equivalent. This string may contain up to 93 significant characters followed by a null. If this string contains a file designation, the devicename portion of the file designation is used for this parameter.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinvdevnam	(130)	The specified devicename is syntactically incorrect.
errundevnam	(131)	The MCS does not recognize the devicename. Is the device mounted?
errnowritepriv	(145)	The process does not have Write Privilege for the file.
errinvcloper	(173)	The operation is inappropriate for the device class.

See Also:

_close	- Close a file
_dismnt	- Dismount a logical device
_getdnam	- Get device name
_write	- Write to an open file

Assembler Calling Sequence:

```
    push    dname                      ;address - device name
    push    status                     ;address - result of the operation
    jsr     _flush                     ;flush I/O buffers to the device
```

C function declaration:

```
                                       /* flush I/O buffers to the device */
    long                               /* returns result of the operation */
    _flush (dname)
            char dname[94];            /* device name */
```

Fortran Subroutine Declaration:

```
    c                                  ! flush I/O buffers to the device
            subroutine flush(dname, status)
                character*94 dname     ! device name
                integer*4 status       ! result of the operation
```

Pascal Procedure Declaration:

```
    procedure _flush(                  {** flush I/O buffers to the device}
            dname   : string[93];      {** device name}
        var status  : longint          {** result of the operation}
    ); external;
```

Wait for fast read to complete.

Description:

Given a valid logical unit number, wait for any asynchronous read operations to complete. Any errors pending from previous asynchronous read operations are reported in the status of this system call.

If there was not a previous asynchronous read, this system call returns successfully.

This call is only implemented on disk class devices.

Related Privileges:

None.

Parameters:

lun	- The logical unit number of the open file on which the fast read was initiated.
status	- The address of a long word to receive the result of the operation.

Diagnostics:

errinvlfn	(132)	The logical unit number does not correspond to an open file.
errinvcloper	(173)	The device class is inappropriate for the operation.
		Device integrity errors.

See Also:

_close	- Close a file
_create	- Create a file
_open	- Open a file
_read	- Read from an open file

Assembler Calling Sequence:

```
push    lun                     ;value - logical unit number
push    status                  ;address - result of the operation
jsr     _frdwait                ;wait for fast read to complete
```

C Function Declaration:

```
                                /* wait for fast read to complete */
```

```
long                                 /* returns result of the operation */
_frdwait (lun)
        long lun;                    /* logical unit number */
```

Fortran Subroutine Declaration:

```
c                                    ! wait for fast read to complete
        subroutine frdwai(lun, status)
            integer*4 lun            ! logical unit number
            integer*4 status         ! result of the operation
```

Pascal Procedure Declaration:

```
procedure _frdwait(                  {** wait for fast read to complete}
        lun     : longint;           {** logical unit number}
    var status  : longint            {** result of the operation}
); external;
```

Deallocate a page of memory.

Description:

This supervisor call allows a process to remove a four kilobyte page of logical memory from its pcb. Unless the page is shared by another process, it is returned to the system memory pool.

A process can deallocate any page which has been allocated to it and which is owned by the calling process.

If the process has writephys privilege, it can deallocate any page of memory which has been allocated to it, independent of whether the page is owned by the calling process.

Related Privileges:

none	- Allows the process to deallocate any page which is allocated to it and which it owns.
writephys	- Allows the process to deallocate any page which is allocated to it.

Parameters:

adr	- Logical address in the 2 megabyte logical address space of the page to be deallocated. This address must be on a 4K byte boundary.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinvadr	(4)	The logical address, for the memory requested, is invalid.
errnonowned	(6)	The process tried to affect a page in memory it did not own.
errmemdeall	(9)	The process attempted to release memory that does not exist.

See Also:

_allmem	- Allocate dynamic memory
_protmem	- Change memory page protection

Assembler Calling Sequence:

```
push    adr                          ;value - address of page
push    status                       ;address - result of the operation
jsr     _fremem                      ;deallocate a page of memory
```

C function declaration:

```
                                        /* deallocate a page of memory */
    long                                /* returns result of the operation */
    _fremem(adr)
            long adr;                   /* address of page */
```

Fortran Subroutine Declaration:

```
    c                                   ! deallocate a page of memory
            subroutine fremem(adr, status)
                integer*4 adr           ! address of page
                integer*4 status        ! result of the operation
```

Pascal Procedure Declaration:

```
    procedure _fremem(                  {** deallocate a page of memory}
            adr     : longint;          {** address of page}
        var status  : longint           {** result of the operation}
    ); external;
```

Assign a global logical name.

Description:

Creates, deletes or replaces a logical name in the global logical name translation table of the current system or another system.

A system's global logical name table contains logical name equivalences that apply to every process in the system. A logical name in the global logical name table does not have to be duplicated in the logical name table of each forked process. Global logical names remain until they are explicitly removed, independent of any process on the system.

Abbreviations are allowed in logical names. An asterisk (*) in the logical name is a marker that indicates the minimum string that is a recognized abbreviation of the logical name. Abbreviations are recognized only during logical name translation (see _trans). For example, if the logical name is "PR*INT", a translation of any of the strings "PR", "PRI", "PRIN", or "PRINT" will return the equivalence.

The values of the parameters lname and equiv determine whether an entry in the logical name table of the specified process is created, removed, or replaced.

To create a new logical name, the lname parameter must contain a logical name which does not match any existing logical names in the global logical name table of the specified system and the equiv parameter must not be null.

To remove a logical name assignment, the lname parameter must contain a logical name which matches a logical name found in the global logical name table of the specified system and the equiv parameter must be null.

To replace the equivalent string associated with a logical name the lname parameter must contain a logical name which matches an existing logical name found in the global logical name table of the specified system and the equiv parameter must not be null.

If the lname parameter contains a logical name which does not match any existing name found in the global logical name table and the equiv parameter is null, or if the lname parameter is null, this system call has no effect.

Related Privileges:

none	- Does not allow the process to affect any names in the global logical name table.
operator	- Allows creation, replacement or deletion of any logical name in the global logical name table.

Parameters:

lname	- Address of null terminated string containing the logical name to be added, replaced or deleted from the logical name table of the specified system. This string may contain up to 93 characters plus a null.
equiv	- Address of null terminated string containing the equivalent to which the logical name translates. It this parameter contains a null string, the logical name represented in parameter lname is removed from the logical name table. This string may contain up to 93 characters plus a null.
siteid	- A long word containing the site id of the system for which this logical name will be in effect. 0=the system on which the calling process is executing.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
errprcsnotfnd	(2)	The specified process is not in the system process table.
errnomemavail	(7)	All available memory has been allocated.
errinvsiteid	(8)	The specified site id does not exist.

See Also:

_assign	- Assign a logical name
_getglb	- Retreive a global logical name
_getlog	- Retrieve a logical name
_trans	- Translate a logical name

Assembler Calling Sequence:

```
push    lname                       ;address - logical name
push    equiv                       ;address - translation string
push    siteid                      ;value - system id
push    status                      ;address - result of the operation
jsr     _gassign                    ;assign a global logical name
```

C function declaration:

```
                                    /* assign a global logical name */
```

```
long                              /* returns result of the operation */
_gassign (lname, equiv, siteid)
        char lname[94];           /* logical name */
        char equiv[94];           /* translation string */
        long siteid;              /* system id */
```

Fortran Subroutine Declaration:

```
c                                 ! assign a global logical name
        subroutine gassig(lname, equiv, siteid, status)
            character*94 lname    ! logical name
            character*94 equiv    ! translation string
            integer*4 siteid      ! system id
            integer*4 status      ! result of the operation
```

Pascal Procedure Declaration:

```
procedure _gassign(                {** assign a global logical name}
        lname   : string[93];      {** logical name}
        equiv   : string[93];      {** translation string}
        siteid  : longint;         {** system id}
    var status  : longint          {** result of the operation}
); external;
```

1

Get PID of ancestor process.

Description:

Return the process id (pid) of a specified ancestor process of the given process.

Related Privileges:

None.

Parameters:

refpid	- The process id (pid) of the process which will serve as the reference point from which ancestors or children PID's will be received. If the refpid is zero (0), it corresponds to the current process. A refpid of $FFFFFFFF (-1) corresponds to the parent of the current process.
rel	- Relative relationship with specified process. ..., -2=grandparent, -1=parent, 0=current process, If the requested relationship goes beyond the actual number of ancestors an error is returned. Specify a relationship of one (1) to get the pid of the oldest ancestor.
pid	- Address of a long word to receive the process id of the relative.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errprcsnotfnd	(2)	The specified process is not in the system process table.

See Also:

_getpcb - Get process control block
_getpid - Get process id (pid) from name
_getpnam- Get process name from pid

Assembler Calling Sequence:

```
push    refpid                      ;value - reference point pid
push    rel                         ;value - relative relationship
push    pid                         ;address - process id
push    status                      ;address - result of the operation
jsr     _gengy                      ;get pid of ancestor process
```

C function declaration:

```
                                        /* get pid of ancestor process */
long                                    /* returns result of the operation */
_gengy(refpid, rel, pid)
        long refpid;                    /* reference point pid */
        long rel;                       /* relative relationship */
        long *pid;                      /* process id */
```

Fortran Subroutine Declaration:

```
c                                       ! get pid of ancestor process
        subroutine gengy(refpid, rel, pid, status)
            integer*4 refpid            ! reference point pid
            integer*4 rel               ! relative relationship
            integer*4 pid               ! process id
            integer*4 status            ! result of the operation
```

Pascal Procedure Declaration:

```
procedure _gengy(                       {** get pid of ancestor process}
        refpid  : longint;              {** reference point pid}
        rel     : longint;              {** relative relationship}
    var pid     : longint;              {** process id}
    var status  : longint               {** result of the operation}
); external;
```

getalc - Get names of allocated devices

Description:

Given a PID, return the names of all devices allocated to that process.

Related Privileges:

none	- Allows the caller to determine which if any devices are allocated to processes with the same uic as the itself.
group	- Allows the caller to determine which if any devices are allocated to processes in the same group as the itself.
world	- Allows the caller to determine which if any devices are allocated to any process.

Parameters:

pid	- Process IDentification number of the process which is to be examined for allocated devices.
devlst	-This parameter is the address of a string buffer in which will be placed the names of the devices allocated to the specified PID. All names are separated by commas. The string is null terminated.
maxlen	- This parameter contains the maximum length of the devlst string.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
errprcsnotfnd	(2)	The specified process is not in the system process table.

See Also:

_alloc	- Allocate an availble device.
_dealloc	- Deallocate an allocated device.
_getrel	- Get names of rotor list elements.
_getrtr	- Get rotor list names.
_setrtr	- Assign device names to a rotor list.

Assembler Calling Sequence:

```
    push    pid                     ; value - process id
    push    devlst                  ; address - string where devices return
    push    maxlen                  ; value - max length of devlst
    push    status                  ; address - status
    jsr     _getalc                 ; get names of allocated devices
```

C Function Declaration:

```
                                    /* get names of allocated devices */
    long                            /* returns result of the operation */
    _getalc(pid,devlst,maxlen);
            long    pid;            /* process id */
            char    devlst[1024];   /* string where devices return */
            long    maxlen;         /* max length of devlst */
```

FORTRAN Subroutine Declaration:

```
    c                                ! get names of allocated devices
        subroutine getalc(pid,devlst,maxlen,status)
            integer*4 pid            ! process id
            character*1024 devlst    ! string where devices return
            integer*4 maxlen         ! max length of alcdev
            integer*4 status         ! result of the operation
```

PASCAL Procedure Declaration:

```
    procedure  getalc(              {** get names of allocated devices}
            pid     : longint;      {** process id}
        var devlst  : string[1024]; {** string where devices return }
            maxlen  : longint;      {** max length of devlst }
        var status  : longint       {** result of the operation}
    ); external;
```

Get PCB attribute bits.

Description:

Call this routine to get the process attribute bits in the PCB for a particular process. To modify the process attributes of a process, use this routine first to get the current ones and set or reset the appropriate bits, then call _SETATTR with the modified value. The pcbattrforceset bit is always returned set.

Related Privileges:

None.

Parameters:

pid	- A long word containing the process ID of the process whose attributes are to be changed. 0 represents the current process; -1 ($FFFFFFFF) represents the parent of the current process.
attr	- Address of a long word to receive the attributes.

Process attribute bit definitions. Note that these offsets are defined for being in the high word of a longword. Because it is only a word in the PCB, if you access the PCB directly you will have to shift these numbers right by 16.

Bit Name	Bit Number	Description
pcbattrdesencrypt	16	If set, do network encryption with DES algorithm.
pcbattrfastencrypt	17	If set, do network encryption with fast algorithm.
pcbattruser1	23	If set, user attribute bit 1.
pcbattruser2	24	If set, user attribute bit 2.
pcbattruser3	25	If set, user attribute bit 3.

pcbattruser4	26	If set, user attribute bit 4.
pcbattrnowatchdog	27	If set, cannot be killed by WATCHDOG utility.
pcbattrswappable	28	If set, the OS will not swap this process.
pcbattrprezeromem	29	If set, pages of memory are zeroed as they are allocated.
pcbattrpostzeromem	30	If set, pages of memory are zeroed as they are released.
pcbattrforceset	31	If set, then modify the bits. Must be set to cause other bits to take effect.

status - Address of a long word to receive the result of the operation.

Diagnostics:

errprcsnotfnd	(2)	The specified process is not in the system process table.

See Also:

_setattr - Set PCB attribute bits

Assembler Calling Sequence:

```
push    pid                     ;value - process id
push    attr                    ;address - to store attribute bits
push    status                  ;address - result of the operation
jsr     _getattr                ;get the attributes
```

C Function Declaration:

```
                                /* get process attributes */
long                            /* returns result of the operation */
_getattr(pid, attr)
        long pid;               /* process id */
        long *attr;             /* returned attributes */
```

FORTRAN Subroutine Declaration:

```
c                                    ! get process attributes
         subroutine _getatt(pid, attr, status)
             integer*4 pid           ! process id
             integer*4 attr          ! returned attributes
             integer*4 status        ! result of the operation
```

Pascal Procedure Declaration:

```
procedure _getattr(                {** get process attributes}
        pid      : longint;        {** process id}
    var attr     : longint;        {** returned attributes}
    var status   : longint         {** result of the operation}
); external;
```

_GETDIR

Get default device and directory.

Description:

Obtain from the OS the current default device and directory specification.

Related Privileges:

None.

Parameters:

devdir	- Address of a 94 byte buffer to receive the default string. The string returned may be up to 93 significant characters followed by a null character.

Diagnostics:

None.

See Also:

_chdir	- Set default device and directory

Assembler Calling Sequence:

```
push    devdir                      ;address - default string
jsr     _getdir                     ;get default device and directory
```

C function declaration:

```
                                    /* get default device and directory */
void                                /* no result */
_getdir (devdir)
        char devdir[94];            /* default string */
```

Fortran Subroutine Declaration:

```
c                                   ! get default device and directory
        subroutine getdir(devdir)
            character*94 devdir     ! default string
```

Pascal Procedure Declaration:

```
procedure _getdir(                  {** get default device and directory}
    var devdir  : string[93]        {** default string}
); external;
```

Get devicename.

Description:

The operating system maintains a device table for each mounted device. Given an index into the array of device tables, this SVC returns the corresponding devicename and device class.

Use this call to obtain the devicenames of mounted devices.

Related Privileges:

None.

Parameters:

siteid	- The site ID of the system whose device table is being queried. A site ID of zero corresponds to the system on which the calling process is running.
index	- The index of which device is desired. An index of 0 returns the name of the first device.
dname	- Address of where to store the devicename. The devicename string will be null terminated. The string must be at least 32 characters long, allowing for up to 31 significant characters plus a null.
class	- Address of a long word to receive the device class.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinvsiteid	(8)	The specified site ID does not exist.
erridxrange	(56)	The table ends before the specified occurrence.

See Also:

_dismnt	- Dismount a logical device
_flush	- Flush I/O buffers to the device
_getdst	- Get device status
_mount	- Mount a logical device
_setdst	- Set device status

Assembler Calling Sequence:

```
push    siteid                  ; value - the system ID
push    index                   ; value - sequence number
push    dname                   ; address - receives devicename
push    class                   ; address - receives device class
push    status                  ; address - result of the operation
jsr     _getdnam                ; get devicename
```

C Function Declaration:

```
                                /* get devicename */
long                            /* returns result of the operation */
_getdnam(siteid, index, dname, class)
         long    siteid;        /* the system ID */
         long    index;         /* sequence number */
         char    dname[94];     /* receives devicename */
         long    *class;        /* receives device class */
```

FORTRAN Subroutine Declaration:

```
c                               ! get devicename
         subroutine _getdna(siteid, index, dname, class, status)
             integer*4 siteid   ! the system ID
             integer*4 index    ! sequence number
             character*94 dname ! receives devicename
             integer*4 class    ! receives device class
             integer*4 status   ! result of the operation
```

Pascal Procedure Declaration:

```
                                {** get devicename}
procedure _getdnam(
        siteid  : longint;      {** the system ID}
        index   : longint;      {** sequence number}
    var dname   : string[93];   {** receives devicename}
    var class   : longint;      {** receives device class}
    var status  : longint       {** result of operation}
);external;
```

Get device protection.

Description:

Retrieves the protection mask on a specified device. The protection mask determines the type of access granted to classes of users on the device.

Protection can be retrieved on any class of device, independent of the privileges posessed by the calling process.

Related Privileges:

None.

Parameters:

dname - Address of a null terminated string containing the name of the device whose protection is sought. This string is translated automatically by the MCS to its logical equivalent. This string may contain up to 93 significant characters followed by a null. If this string contains a file designation, the devicename portion of the file designation is used for this parameter.

prot - Address of a long word to receive the protection mask. The least significant 16 bit word of this long word is divided into 4 nibbles. Each nibble corresponds to a class of users. The bits within each nibble represent the type of access that class of user is granted for the device. If the bit is set (1) the access is granted.

From the least to the most significant nibble the user classes are:

Ownr	- The device owner
Grp	- Processes with the same group id as the owner
Pub	- All other processes in the system
Sys	- Processes with system privilege

```
  Sys  Pub  Grp  Ownr
 |----|----|----|----|
 |DWRE|DWRE|DWRE|DWRE|
 |-------------------|
MSB                 LSB
```

From the least to the most significant bit within the nibbles, the access privileges are:

E - Execute access

```
                     R  - Read access
                     W  - Write access
                     D  - Delete access

    status  - Address of a long word to receive the result of
              the operation.
```

Diagnostics:

errinvdevnam	(130)	The specified devicename is syntactically incorrect.
errundevnam	(131)	The MCS does not recognize the devicename. Is the device mounted?
errnoreadpriv	(144)	The process does not have Read Privilege for the file.

See Also:

_getfprt - Get file protection
_setdprt - Set device protection
_setfprt - Set file protection

Assembler Calling Sequence:

```
    push    dname                       ;address - device name
    push    prot                        ;address - protection mask
    push    status                      ;address - result of the operation
    jsr     _getdprt                    ;get device protection
```

C Function Declaration:

```
                                        /* get device protection */
    long                                /* returns result of the operation */
    _getdprt(dname, prot)
            char dname[94];             /* device name */
            long *prot;                 /* protection mask */
```

Fortran Subroutine Declaration:

```
    c                                   ! get device protection
            subroutine getdpr(dname, prot, status)
                character*94 dname      ! device name
                integer*4 prot          ! protection mask
                integer*4 status        ! result of the operation
```

Pascal Procedure Declaration:

```
    procedure _getdprt(                 {** get device protection}
            dname   : string[93];       {** device name}
        var prot    : longint;          {** protection mask}
        var status  : longint           {** result of the operation}
```

```
); external;
```

Get device status.

Description:

Given the device name of a currently mounted device, copies the device table and device status into user specified buffers.

CAUTION: The format of the device table may change with each release. The current definition is included in each release in the file /SYSINCL.SYS/DEVTDISP.*. The record definition is named "devicetable", i.e. in your program you can declare a variable of type "devicetable."

The device table for a device contains the information maintained about the device by the class handler. The device table is divided into two parts. The first part is device independent, and the second part is device class dependent. The device independent part is as follows:

Name	Length (bytes)	Description
dtnextlink	4	Pointer to the next device table
dtbacklink	4	Pointer to the previous device table
dtdevname	8	The user supplied device name
dtclass	2	Contains the device class. Valid options are:

Class Name	Value	Description
dtclassttyspc	0	Character device (ttyspc)
dtclasstty	1	Character device (tty)
dtclasstapespc	2	Tape device (tapespc)
dtclasstape	3	Tape device (tape)
dtclassdiskspc	4	Disk device (diskspc)
dtclassdisk	5	Disk device (disk)
dtclassnetspc	6	Network dev. (networkspc)
dtclassnet	7	Network device (network)
dtclasspipespc	8	Pipe device (pipespc)
dtclasspipe	9	Pipe device (pipe)
dtclasssyncspc	10	BSC device (syncspc)
dtclasssync	11	BCS device (sync)
dtclassquespc	12	Queue device (quespc)
dtclassque	13	Queue device (que)
dtclassnondevspc	14	Non-dev device(nondevspc)
dtclassnondev	15	Non-dev device (nondev)

dtrefcount	2	The number of files currently open on the device
dtdriveid	4	Internal drive ID
dtallocpid	4	The PID of the process that has this device allocated
dtsiteid	2	The site ID of this device
dtseqnum	2	The mount sequence number of this device. This will be unique for each device on the machine.
dtdefuserid	2	The default userid for this device. This will be loaded into the DTUSERID variable everytime the DTREFCOUNT variable goes to zero.
dtdefgroupid	2	The default group ID for this device. This will be loaded into the DTGROUPID variable everytime the DTREFCOUNT variable goes to zero.
dtdefprotect	2	The default protection mask for this device. This will be loaded into the DTPROTECT variable everytime the DTREFCOUNT variable goes to zero.
dtclassptr	4	Address of the class handler for this device
dtdriverptr	4	Address of the device driver for this device
dtflags	2	Device flags. This is a bit encoded word.

Bit Name	Bit #	Description
dtflfcbflushmode	4	Current flush mode for disk fcbs
dtflchflushmode	5	Current flush mode for disk cache
dtflflushing	6	Device is currently being flushed
dtflwriteprot	7	Device is write protected
dtflcreatmode	10	Tape file is being created
dtflfileopen	11	Tape file is open
dtfleot	12	Tape is at physical end of tape
dtfleof	13	Tape is at logical end of file
dtflsessionestb	15	A session is currently established

dtfcbptr	4	Address of the file control block of the first open file on the device. A list head pointer. (Used for disks only)
dtblksz	2	Block size for the device
dtuserid	2	Owner ID portion of the UIC. Corresponds to the owner of the device.
dtgroupid	2	Group ID portion of the UIC. Corresponds to the owner of the device.

dtprotect	2	The device protection flags. Uses the same format at the file protection flags.
dtmntmstime	4	The most significant 32 bits (year and day) of the date and time the device was mounted
dtmntlstime	4	The least significant 32 bits (hour, minute, second and tick) of the date and time the device was mounted
dtidfield	2	Table identifier flag
dtidtag	$5555	Table ID value for this table

For TTY, PIPE, SYNC, and NONDEV class devices, the second part of the table is defined as follows:

Name	Length (bytes)	Description
dttyreadacc	1	The read access count (the number of times this device has been opened for read access)
dttyreadlock	1	The read lock count (the number of times this device has been opened with read lock)
dttywriteacc	1	The write access count (the number of times this device has been opened for write access)
dttywritelock	1	The write lock count (the number of times this device has been opened with write lock)
dttywriteqh	4	The write queue header
dttyreadqh	4	The read queue header
dttydriveid	2	Contains drive table index
dttyboardid	2	Contains board table index
dttytypeid	2	Contains type ID of board

For TAPE class devices, the second part of the table is defined as follows:

Name	Length (bytes)	Description
dttpreadahead	2	Read ahead flag
dttpfilseqno	4	Sequence number of currently open file or next file to be opened.
dttpcachesz	2	Number of elements in tape cache

dttpcacheadr	4	Address of cache header
dttpskpcache	4	Address of special cache header for non-buffered commands, i.e., skip, get or set status, write file mark
dttpnextblk	4	Next logical block number in the currently open file
dttpreadpos	2	Actual block number to be read next physically

For DISK class devices, the second part of the table is defined as follows:

Name	Length (bytes)	Description
dtdkflags	2	Disk class flags. This is a bit encoded word. Bit Name / Bit # / Description dtdkflautoflush 0 If set do auto flushing dtdkflreadahead 1 If set do readahead dtdkflforcedwrite 2 If set do forced writes on all writes
dtdksecshfcnt	2	The sector shift count
dtdkdefalloc	2	The initial file allocation
dtdksecalloc	2	The secondary file allocation
dtdkchreadmin	2	Non-modified cache minimum size
dtdkmaxuserch	2	Number of cache elements (minus 1) that can be consumed in a single request to the OS
dtdkszmaxch	2	Size of stack area in bytes used to hold the addresses of used cache elements ((devcldsmaxcache+2)*4)
dtdkcachecolsz	2	The number of columns in the cache
dtdkcachesze	2	The number of cache sectors
dtdkchaddr	4	Address of disk cache column table
dtdkbmpos	4	Bitmap file's next allocation location
dtdkfcbbmpos	4	Fcbbitmap file's next allocation location
dtdkfcbptr	4	Address of fcb for FCB.SYS
dtdkdirptr	4	Address of fcb for ROOTDIR.DIR
dtdkfcbbitptr	4	Address of fcb for FCBBITMAP.SYS
dtdkbitptr	4	Address of fcb for BITMAP.SYS
dtdkalocsecqh	4	Allocate disk queue head
dtdkalocfcbqh	4	Allocate fcb queue head

For NETWORK class devices, the second part of the table is defined as follows:

Name	Length (bytes)	Description
dtnkreadacc	1	The read access count (the number of times this device has been opened for read access)
dtnkreadlock	1	The read lock count (the number of times this device has been opened with read lock)
dtnkwriteacc	1	The write access count (the number of times this device has been opened for write access)
dtnkwritelock	1	The write lock count (the number of times this device has been opened with write lock)
dtnkflags	2	Network class flags. This is a bit encoded word. Bit Name / Bit # / Description dtnkflvcdriver / 0 / If set, this is a virtual circuit driver
dtnkwriteqh	4	The write access queue header
dtnkreadqh	4	The read access queue header
dtnkhwrite	4	Pointer to network layer write routine
dtnkhuninit	4	Pointer to network layer uninit routine

For QUEUE class devices, the second part of the table is defined as follows:

Name	Length (bytes)	Description
dtqucbptr	4	Contains the address of control block page which is the communication block between the QUEUE class handler and the queue manager process
dtqufhptr	4	Contains the address of the queue control file header page
dtquwriteoper	4	Contains how many write operations have been performed on the QUEUE

dtquflags	2	QUEUE class flags. Bit encoded word.		
		Bit Name	Bit #	Description
		dtqufldefcrp	0	A default create process record is defined. This means a user can redirect I/O directly to the QUEUE.
		dtquflqmres	1	The queue manager process is to remain resident at all times
		dtquflqmnodie	2	In critical code and the queue manager process cannot die
		dtquflclosed	3	The queue is marked as closed. No new entries may be queued.
		dtquflhalted	4	The queue is marked as halted. No pending entries will be executed.
		dtqufl clean	5	There are no entries in the queue control files

The device status is a device class dependent 128 byte table. It is maintained by the device driver for each device.

NOTE: The device status table may change with each release of the operating system. The current definition is included in each release in the file named: /SYSINCL.SYS/ DSTATDISP.*. The name of the record included in that file is "devicestatus," i.e. in your program you can declare a variable whose type is "devicestatus."

The device status table is divided into two parts. The first half is device independent and is composed of the following fields:

Name	Length (bytes)	Description
dsclassid	2	The device class. Valid classes are: (Note that these names are defined in the devtdisp.* files)

Class Name	Value	Description
dtclassttyspc	0	Character device (ttyspc)
dtclasstty	1	Character device (tty)
dtclasstapespc	2	Tape device (tapespc)
dtclasstape	3	Tape device (tape)
dtclassdiskspc	4	Disk device (diskspc)
dtclassdisk	5	Disk device (disk)
dtclassnetspc	6	Network dev. (networkspc)
dtclassnet	7	Network device (network)
dtclasspipespc	8	Pipe device (pipespc)
dtclasspipe	9	Pipe device (pipe)
dtclasssyncspc	10	BSC device (syncspc)
dtclasssync	11	BCS device (sync)
dtclassquespc	12	Queue device (quespc)
dtclassque	13	Queue device (que)
dtclassnondevspc	14	Non-dev device(nondevspc)
dtclassnondev	15	Non-dev device (nondev)

Name	Length (bytes)	Description
dsdriverid	2	The unique id number for this device driver
dsblksz	2	The block size of the device (e.g. sector size)
dsharderr	2	The hard error count for the device
dssofterr	2	The soft error count for the device
dsreadoper	4	The number of read operations on this device
dswriteoper	4	The number of write operations on this device
dsmaxnumdev	2	Maximum # of devices this driver can handle
dscurnumdev	2	Number of devices currently mounted using this device driver
dsnumtoretry	2	Number of times to retry before reporting a hard error
dserrorreason	4	This contains the hardware error code for the last error received on this device
dsreserved	32	Reserved
dsnexttableptr	4	Address of next device status table

The second half of the device status table is device class dependent For TAPE class devices the second part is defined as follows:

Name	Length (bytes)	Description		
dstpstatus	2	Tape device status. A bit encoded word.		
		Bit name	bit #	Description
		dstpready	0	Set if device ready
		dstpintpend	1	Set if interrupt pending
		dstprewinding	2	Set if tape rewinding
		dstpbotdetect	3	Set if device is at physical BOT
		dstpeotdetect	4	Set if device is at physical EOT
		dstpwriteprot	5	Set if tape is write protected
dstpflagsl	2	Tape status information. A bit encoded word.		
		Bit name	bit #	Description
		dstpdoraw	0	0=Read after write disabled 1=Read after write enabled
		dstperrintenb	1	0=Error interrupts are enabled 1=Error interrupts are disabled
dstpspeed	1	Tape speed. Values are:		
			0 - Reserved	
		dstpspeed12ips	1 - 12 ips	
		dstpspeed25ips	2 - 25 ips	
		dstpspeed30ips	3 - 30 ips	
		dstpspeed50ips	4 - 50 ips	
		dstpspeed90ips	5 - 90 ips	
		dstpspeed100ips	6 - 100 ips	
		dstpspeed125ips	7 - 125 ips	
dstpdensity	1	Tape density. Values are:		
			0 - Reserved	
		dstpdens800bpi	1 - 800 bpi	
		dstpdens1600bpi	2 - 1600 bpi	
		dstpdens3200bpi	3 - 3200 bpi	
		dstpdens6250bpi	4 - 6250 bpi	
		dstpdens6400bpi	5 - 6400 bpi	

dstpiopbcnt	2	Number of IOPBs allocated to device
dstpcachesz	2	Number of cache elements allocated to device
dstpreserved	46	Reserved
dstpuserfield	8	User defined status

For DISK class devices the second half of the device status table is defined as follows:

Name	Length (bytes)	Description
dsdkintfac	2	Disk interleave factor
dsdkiopbcnt	2	Number of IOPB's allocated to the drive
dsdknumbsect	4	The number of sectors on the volume
dsdksectrack	2	The number of sectors on a track
dsdkheads	2	The number of heads on the device
dsdkcylinders	2	The number of cylinders on the volume
dsdkflagsl	2	Disk status information. A bit encoded word.

Bit Name	Bit #	Description
dsdkdensity1	0	Device density
dsdkdensity2	1	
dsdkdenssignle		00 - Single density
dsdkdensdouble		01 - Double density
dsdkdensquad		10 - Quad density
dsdkdensreserve		11 - Reserved
dsdkdoraw	3	If set, do Read after write verify
dsdkwriteprot	4	If set, Device write protected
dsdkseekdir	15	Current seek direction
dsdkseekincr		0 - Increasing cylinder numbers
dsdkseekdecr		$8000 - Decreasing cylinder numbers

dsdkcurcyl	2	Current cylinder position
dsdkcachesz	2	Number of sectors in the disk cache
dsdkentryname	16	A null terminated string containg the name of this type of drive
dsdkreserved	20	Reserved
dsdkuserfield	8	User Defined status

For TTY class devices the second half of the device status table is defined as follows:

Name	Length (bytes)	Description
dstymoderegl	1	Uart mode register 1. This byte is bit encoded as follows:

Bit Name	Bit #	Description
dstymrlbaudfacl	0	Baud factor
dstymrlbaudfac2	1	
dstymrlsyncl		00 - sync 1 x clock rate
dstymrlasyncl		01 - async 1 x clock rate
dstymrlasync16		10 - async 16 x clock rate
dstymrlasync64		11 - async 64 x clock rate
dstymrlcharlenl	2	Character length
dstymrlcharlen2	3	definition
dstymrldw5bit		00 - 5 data bits
dstymrldw6bit		01 - 6 data bits
dstymrldw7bit		10 - 7 data bits
dstymrldw8bit		11 - 8 data bits
dstymrlparityctrl	4	Parity control
dstymrlpardis		0 - disable parity
dstymrlparenb		1 - enable parity
dstymrlparitytype	5	Parity type
dstymrlparodd		0 - odd parity
dstymrlparevn		1 - even parity
dstymrlstopbitsl	6	Async mode # of stop bits
dstymrlstopbits2	7	Async mode # of stop bits
dstymrlbinv		00 - invalid
dstymrlsbl		01 - 1 stop bit
dstymrlsbl5		10 - 1.5 stop bits
dstymrlsb2		11 - 2 stop bits
dstymrltransctrl	6	Sync mode transparent
dstymrlnormal		0 - normal
dstymrltrans		1 - transparent
dstymrlnumsync	7	Sync mode # of syncs
dstymrlsyncdouble		0 - double sync
dstymrlsyncsingle		1 - single sync

dstymodereg2	1	Uart mode register 2. This byte is bit encoded as follows:		
		Bit Name	Bit #	Description
		dstymr2baudrt1	0	The baud rate
		dstymr2baudrt2	1	Baud rate continued
		dstymr2baudrt3	2	Baud rate continued
		dstymr2baudrt4	3	Baud rate continued
		dstymr2baud50		0000 - 50 baud
		dstymr2baud75		0001 - 75 baud
		dstymr2baud110		0010 - 110 baud
		dstymr2baud1345		0011 - 134.5 baud
		dstymr2baud150		0100 - 150 baud
		dstymr2baud300		0101 - 300 baud
		dstymr2baud600		0110 - 600 baud
		dstymr2baud1200		0111 - 1200 baud
		dstymr2baud1800		1000 - 1800 baud
		dstymr2baud2000		1001 - 2000 baud
		dstymr2baud2400		1010 - 2400 baud
		dstymr2baud3600		1011 - 3600 baud
		dstymr2baud4800		1100 - 4800 baud
		dstymr2baud7200		1101 - 7200 baud
		dstymr2baud9600		1110 - 9600 baud
		dstymr2baud19200		1111 - 19200 baud
		dstymr2recvclock	4	Receiver clock
		dstymr2recextclk		0 - External clock
		dstymr2recintclk		1 - Internal clock
		dstymr2transclock	5	Transmitter clock
		dstymr2trnextclk		0 - External clock
		dstymr2trnintclk		1 - Internal clock
			6-7	Reserved
dstycmdreg	1	Uart command register. Bit encoded.		
		Bit Name	Bit #	Description
		dstycrtransctrl	0	Transmitter control
		dstycrtcdis		0 - Disable transmitter
		dstycrtcenb		1 - Enable transmitter
		dstycrdtr	1	Data terminal ready
		dstycrdtrhigh		0 - DTR high
		dstycrdtrlow		1 - DTR low
		dstycrrecvcrtl	2	Receiver control
		dstycrrcdis		0 - Disable receiver
		dstycrrcenb		1 - Enable receiver
		dstycrforcebrk	3	Async force break
		dstycrbrknorm		0 - normal
		dstycrbrkforce		1 - force break

		dstycrsenddle	3	Sync send DLE
		dstycrdlenorm		0 - normal
		dstycrdlesend		1 - send DLE
		dstycrreseterror	4	Reset error
		dstycrnoreset		0 - normal
		dstycrreseterr		1 - reset error
		dstycrrts	5	Request to send
		dstycrrtshigh		0 - RTS high
		dstycrrtslow		1 - RTS low
		dstycropermode1	6	Operating mode
		dstycropermode2	7	Operating mode continued
		dstycromnormal		00 - Normal operation
		dstycromautoecho		01 - Async autoecho
		dstycromstripdle		01 - Sync strip DLE
		dstycromlocallp		10 - Local loop back
		dstycromremotelp		11 - Remote loop back
dstytermtype	1	Terminal type definition. This byte contains values for each type of terminal.		
		Value Name	Value	Description
			0-15	User defined types
			16-246	Reserved
		dstywit	247	WIT terminal
		dstyhydra	248	Hydra terminal
		dstyvt100	250	VT-100 terminal
		dstyvt52	251	VT-52 terminal
		dstyt7000	252	T-7000 terminal
		dstymg8000	253	MG-8000 terminal
		dstytvi912c	254	TVI 912 C terminal
		dstyvisual200	255	Visual 200 terminal
dstystatreg	1	Uart status register. Bit encoded.		
		Bit Name	Bit #	Description
		dstysrtransrdy	0	Transmitter buffer ready
		dstysrtranfull		0 - Transmitter full
		dstysrtranempty		1 - Transmitter empty
		dstysrrecvrdy	1	Receiver buffer ready
		dstysrrecvempty		0 - Receiver empty
		dstysrrecvfull		1 - Receiver full
		dstysrdschg	2	DSR or DCD change
		dstysrdsrnormal		0 - Normal
		dstysrdsrchange		1 - DSR or DCD change

		dstysrparityerr	3	Parity error
		dstysrparnormal		0 - Normal
		dstysrparerror		1 - Async parity error. Sync parity error or DLE received
		dstysroverrunerr	4	Overrun error
		dstysrovernormal		0 - Normal
		dstysrovererror		1 - Overrun error
		dstysrframingerr	5	Framing error
		dstysrframnormal		0 - Normal
		dstysrframerror		1 - Async framing error. Sync SYN char
		dstysrdcddetect	6	DCD Detect
		dstysrdcdhigh		0 - DCD high
		dstysrdcdlow		1 - DCD low
		dstysrdsrdetect	7	DSR Detect
		dstysrdsrhigh		0 - DSR high
		dstysrdsrlow		1 - DSR low
dstypacketterm	1	Holds code for packet termination characters		
		Value Name	Value	Description
		dstyptnoterm	0	Do not terminate packet on any control characters
		dstyptallterm	1	Terminate packets on all control characters
		dstyptcrterm	2	Terminate packet on carriage return <CR> character
dstyflagsl	2	Terminal status information. Bit encoded.		
		Bit Name	bit #	Description
		dstycontrolc	0	Control C enable (0 = enabled)
		dstyxonxoff	1	xon xoff enable (0 = enabled)
		dstycontrolx	2	Control X enable (0 = enabled)
		dstycontrolz	3	Control Z enable (0 = enabled)
		dstycontrolo	4	Control O enable (0 = enabled)
		dstytabmap	5	Tab map enable (1 = enabled)
		dstymask8bit	6	Mask 8th bit enable (0 = enabled)

dstycontrolu	7	Control U enable (0 = enabled)
dstybroadcast	8	Broadcast enable (0 = enabled)
dstyhandshake1	9	Handshaking type
dstyhandshake2	10	
dstyhsbell		00 - No handshake, send bell
dstyhssoft		01 - Software handshake
dstyhshard		10 - Hardware handshake
dstyhsnone		11 - No handshake, no bell
dstyduplex	11	Full/half duplex (0 = full duplex)
dstymodemctrl	12	Modem control enable (1 = enabled)
dstyautobaud	13	Auto baud enable (1 = enabled)
dstyremote	14	Remote enable (1 = enabled)

dstyinputcnt	2	Count of characters in input interrupt buffer
dstyoutptcnt	2	Count of characters in output interrupt buffer
dstycolumnpos	2	Current column position
dstyinbufsz	2	Input buffer size in bytes
dstyoutbufsz	2	Output buffer size in bytes
dstywidth	2	The width of the given terminal screen
dstylength	2	The length of the given terminal screen
dstysubreadoper	4	Number of sub-read operations
dstysubwriteoper	4	Number of sub-write operations
dstyreserved	26	Reserved
dstyuserfield	8	User defined status

For PIPE class devices the second part of the device status table is defined as follows:

Name	Length (bytes)	Description
dsppreaderpid	4	Process ID of pending reader
dsppwriterpid	4	Process ID of pending writer
dspppipeid	4	The pipe's ID
dsppbuffersz	2	The buffer size in bytes
dsppbuffercnt	2	Number of characters in the pipe buffer

dsppreadque	4	Address of read queue
dsppwriteque	4	Address of write queue
dsppreserved	32	Reserved
dsppuserfield	8	User defined status

For SYNC class devices the second part of the device status table is defined as follows:

Name	Length (bytes)	Description
dssymoderegl	1	Mode register 1 of the uart (See DSTYMODEREG1 for bit definitions)
dssymodereg2	1	Mode register 2 of the uart (See DSTYMODEREG2 for bit definitions)
dssycmdreg	1	Command register of the uart (See DSTYCMDREG for bit definitions)
dssytermtype	1	Terminal type definition. A binary value.
dssystatreg	1	Status register of uart (See DSTYSTATREG for bit definitions)
dssynumbsync	1	Number of sync characters to write
dssyflagsl	2	Device Status flags. Bit encoded.

dssytermtype values:

Value Name	Value	Description
dssyibm3741	249	IBM 3741 terminal
dssyibm2968	250	IBM 2968 terminal
dssyibm2770	251	IBM 2770 terminal
dssyibm3276	252	IBM 3276 terminal
dssyibm3275	253	IBM 3275 terminal
dssyibm2780	254	IBM 2780 RJE
dssyibm3780	255	IBM 3780 RJE

dssyflagsl bits:

Bit Name	Bit #	Description
dssymultipnt	0	0=point to point 1=multipoint
dssyebcdic	1	0=ascii line 1=ebcdic line
dssycrcccitt	2	0=crc-16 1=crc-ccitt
dssylrc	3	0=crc (on above types) 1=lrc
dssyasctoebcw	4	0=no translate on write 1=translate ascii to ebcdic on write

		dssyebctoascr	5	0=no translate on read 1=translate ebcdic to ascii on read
		dssytranstbl2	6	0=use translate table 1 1=use translate table 2
dssyinputcnt	2	Number of characters in input interrupt buffer		
dssyoutputcnt	2	Number of characters in output interrupt buffer		
dssyinbufsz	2	Input buffer size in bytes		
dssyoutbufsz	2	Output buffer size in bytes		
dssyprevrderr	4	Error from previous un-verified read		
dssyprevwrerr	4	Error from previous no-wait write		
dssyprevrdtype	1	Type of previous read dssynontran - 0 Non-transparent read dssytran - 1 Transparent read		
dssynumbtrpad	1	The number of trailing pads to write		
dssyrecsize	2	Used in transparent mode with ITBs		
dssyreserved	28	Reserved		
dssyuserfield	8	User defined status		

For NETWORK class devices the second part of the device status table is defined as follows:

Name	Length (bytes)	Description		
dsnkflags	2	Device status flags. Bit encoded.		
		Bit Name	Bit #	Description
		dsnkbyte	0	0=datagram mode 1=byte mode
		dsnkmodemctrl	1	0=not enabled 1=modem ctrl enabled
dsnkwindowsize	1	Window size the circuit will use		
dsnkreserved	53	Reserved		
dsnkuserfield	8	User defined status		

For NONDEV class devices the second part of the device status table is defined as follows:

Name	Length (bytes)	Description
dsnduserfield	64	Reserved

For QUEUE class devices the second part of the device status table is defined as follows:

Name	Length (bytes)	Description
dsquassocdev	9	A null terminated string containing the name of the physical printer device
dsqusenddev	9	A null terminated string containing the name of the physical device that control messages are to be sent to
dsquformname	10	A null terminated string containing the current form name
dsqunumexec	2	Maximum number of entries that can execute concurrently
dsqucurnumexec	2	The number of entries that are currently active
dsquflags	2	Device Status flags. Bit encoded.
dsqulength	2	This holds the length of the forms of the printer associated with this queue
dsquwidth	2	This holds the width of the forms of the printer associated with this queue
dsqunextentry	4	The entry number of the next entry to be enqued

dsquflags bits:

Bit Name	Bit #	Description
dsquflupdating	0	If set, currently updating queue control file
dsquflqmstay	1	If set, the queue manager process will remain running even when queue is empty
dsquflnorestart	2	If set, when the queue is mounted it does not restart the jobs in the queue

dsqutype	1	The type of queue this is. The values are:

Value Name	Value	Description
dsqutpprint	1	Print type queue
dsqutpjob	2	Job entry type queue

dsqubaseprior	1	The priority that entries will be queued at if they specify the default priority
dsqureserved	20	Reserved
dsquuserfield	8	User defined status

Related Privileges:

None.

Parameters:

dname	- Address of a null terminated string containing the name of the device. This string is translated automatically by the MCS into its logical equivalent. This string may contain up to 93 significant characters followed by a null. If this string contains a file designation, the devicename portion of the file designation is used.
dtable	- Address of a buffer to receive the device table. This table must be word aligned.
ldtab	- Length of the device table. Up to this many bytes of the device table will be transferred to the user buffer.
dstat	- Address of a 128 byte buffer to receive the device status.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinvdevnam	(130)	The specified devicename is syntactically incorrect.
errundevnam	(131)	The MCS does not recognize the devicename. Is the device mounted?
errnoreadpriv	(144)	The process does not have Read Privilege for the file.

See Also:

_dismnt	- Dismount a logical device
_getdnam	- Get device name
_giodst	- Get device status with lun
_mount	- Mount a logical device
_setdst	- Set device status
_siodst	- Set device status with lun

Assembler Calling Sequence:

```
%%sys$disk/sysincl.sys/devtdisp.asm
%%sys$disk/sysincl.sys/dstatdisp.asm
push    dname                   ;address - device name
push    dtable                  ;address - device table
push    ldtab                   ;value - length of device table
push    dstat                   ;address - device status
push    status                  ;address - result of the operation
jsr     _getdst                 ;get device status
```

C Function Declaration:

```
#include "sys$disk/sysincl.sys/devtdisp.h"
#include "sys$disk/sysincl.sys/dstatdisp.h"
                                /* get device status */
long                            /* returns result of the operation */
_getdst(dname, dtable, ldtab, dstat)
         char dname[94];        /* device name */
         devicetable *dtable;   /* device table */
         long ldtab;            /* length of device table */
         devicestatus *dstat;   /* device status */
```

FORTRAN Subroutine Declaration:

```
c                               ! get device status
         subroutine _getdst(dname, dtable, ldtab, dstat, status)
             character*94 dname ! device name
             character*(*) dtable ! device table
             integer*4 ldtab    ! length of device table
             character*(*) dstat ! device status
             integer*4 status   ! result of the operation
```

Pascal Procedure Declaration:

```
%%sys$disk/sysincl.sys/devtdisp.pas
%%sys$disk/sysincl.sys/dstatdisp.pas

procedure _getdst(               {** get device status}
        dname   : string[93];    {** device name}
        dtable  : ^array_of_char;{** device table}
        ldtab   : longint;       {** length of device table}
        dstat   : ^array_of_char;{** device status}
    var status  : longint        {** result of the operation}
); external;
```

Get device UIC.

Description:

Given a device name, returns the user identification code (uic) which is composed of an owner id and a group id.

This system call is valid for any class of device.

Related Privileges:

None.

Parameters:

dname	- Address of a null terminated string containing the name of the device whose uic is requested. This string will be translated automatically by the MCS to its logical equivalent. This string may contain up to 93 valid characters followed by a null byte. If the string contains a file designation, the devicename part of the file designation is used for this parameter.
uic	- Address of a long word to receive the user identification code. This long word is divided into two fields. The most significant 16 bits constitute the owner id number. The least significant 16 bits constitute the group id number (identifying the group to which the user belongs).
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinvdevnam	(130)	The specified devicename is syntactically incorrect.
errundevnam	(131)	The MCS does not recognize the devicename. Is the device mounted?

See Also:

_getfuic	- Get file uic
_getuic	- Get process uic
_setduic	- Set device uic
_setfuic	- Set file uic
_setuic	- Set process uic

Assembler Calling Sequence:

```
    push    dname                       ;address - device name
    push    uic                         ;address - user id code
    push    status                      ;address - result of the operation
    jsr     _getduic                    ;get device uic
```

C Function Declaration:

```
                                        /* get device uic */
    long                                /* returns result of the operation */
    _getduic(dname, uic)
            char dname[94];             /* device name */
            long *uic;                  /* user id code */
```

Fortran Subroutine Declaration:

```
    c                                   ! get device uic
            subroutine getdui(dname, uic, status)
                character*94 dname      ! device name
                integer*4 uic           ! user id code
                integer*4 status        ! result of the operation
```

Pascal Procedure Declaration:

```
    procedure _getduic(                 {** get device uic}
            dname    : string[93];      {** device name}
        var uic      : longint;         {** user id code}
        var status   : longint          {** result of the operation}
    ); external;
```

Read event flags.

Description:

Read the event flags of any desired process. The event flags to be read are specified by a mask.

Related Privileges:

None	- Allows reading event flags of any process with the same owner id and group id (uic) as the calling process.
group	- Allows reading event flags of any process with the same group id as the calling process.
world	- Allows reading event flags of any process.

Parameters:

pid	- process id of the process whose event flags are to be read.
efmask	- Event flag mask specifying which of the event flags are to be read. Those bits that correspond to 1's in the mask will be read. The other bits will be set to zero.
eflags	- Address of a long word to receive the event flags which were read.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
errprcsnotfnd	(2)	The specified process is not in the system process table.

See Also:

_andevnt	- Wait for AND of event flags
_clrevnt	- Clear event flags
_orevnt	- Wait for OR of event flags
_setevnt	- Set event flags

Assembler Calling Sequence:

```
push    pid                     ;value - process id
push    efmask                  ;value - event flag mask
push    eflags                  ;address - resulting event flags
push    status                  ;address - result of the operation
```

```
    jsr     _getevnt                      ;read event flags
```

C function declaration:

```
                                          /* read event flags */
    long                                  /* returns result of the operation */
    _getevnt(pid, efmask, eflags)
            long pid;                     /* process id */
            long efmask;                  /* event flag mask */
            long *eflags;                 /* resulting event flags */
```

Fortran Subroutine Declaration:

```
    c                                     ! read event flags
    c                                     ! returns result of the operation
            integer*4 function getevn(pid, efmask, eflags)
                integer*4 pid             ! process id
                integer*4 efmask          ! event flag mask
                integer*4 eflags          ! resulting event flags
```

Pascal Procedure Declaration:

```
    procedure _getevnt(                   {** read event flags}
            pid     : longint;            {** process id}
            efmask  : longint;            {** event flag mask}
        var eflags  : longint;            {** resulting event flags}
        var status  : longint             {** result of the operation}
    ); external;
```

Get the address of the current exit handler.

Description:

Call this routine to get the address of the currently defined exit handler. (See _SETEXIT for a description of exit handlers.) Returns zero if no exit handler is defined.

Related Privileges:

None.

Parameters:

adr	- Address to store exit handler address.

Diagnostics:

None.

See Also:

_errno	- Receive process abort reason
_exitrtn	- Define a returnable exit handler
_exproc	- Terminate the specified process
_setexit	- Set exit handler

Assembler Calling Sequence:

```
push    adr                     ;address - address of exit handler
jsr     _getexit                ;get the exit handler address
```

C Function Declaration:

```
                                /* get exit handler address */
void                            /* no result */
_getexit(adr)
        long *adr;              /* Returned address of exit handler */
```

FORTRAN Subroutine Declaration:

```
c                               ! get exit handler address
        subroutine _getexi(adr)
            integer*4 adr       ! Returned address of exit handler
```

Pascal Procedure Declaration:

```
procedure _getexit(               {** get exit handler address}
    var adr     : longint         {** Returned address of exit handler}
); external;
```

Get file control block.

Description:

Given the logical unit number (lun) of a file successfully opened for read and/or write access by the calling process, the file control block (fcb) for that file is copied to the process's buffer.

CAUTION: The format of the file control block may change with each release. The current definition is included in each release in the file /SYSINCL.SYS/FCBDISP.*. The name of the fcb record is "fcbtype", i.e. in your program you can declare a variable whose type is "fcbtype".

There are several variations on the format of file control blocks, depending on the class of device which contains the file. Disk files contain "root" fcbs and "continuation" fcbs. Tape files have "tape" fcbs. All other files have "tty" fcbs.

On tapes, the zeroeth fcb is the file header. It does not contain accurate file size information. The first continuation fcb on a tape is the file trailer. It is the same as the file header except that it contains correct file size information. If the first continuation fcb of a tape file is requested, the tape is positioned at the logical end of the file.

The format of the first 14 bytes of the fcb record is the same for all types of fcb's. The format of this common type is:

Name	Length (bytes)	Description
fcbnum	4	fcb number for this fcb. The record number of this record within the fcb file. For tty fcbs, the value of this field is zero.
fcbseqnum	2	fcb sequence number. This number is unique for each usage of this fcb. For tty fcbs, the value of this field is zero.

fcbcntfcbnum	4	fcb number of continuation fcb. The record number of the next fcb for this same file. For tape and tty fcbs, the value of this field is zero.
fcbcntseqnum	2	Sequence number of the continuation fcb. For tape and tty fcbs, the value of this field is zero.
fcbusageid	1	Usage id field. The type of fcb. Values are:

Name	Value	Description
fcbunalloc	0	This fcb is unused. The data in this record is invalid.
fcballocroot	1	This record contains a root fcb.
fcballoccont	2	This record contains a continuation fcb.

fcbextusecnt	1	Number of extent fields in use within this fcb.

The format of the last 242 bytes of the fcb is different for "primary" fcbs as opposed to "continuation" fcbs. For primary fcbs (disk, tape and tty) the format is as follows:

fcbfiletype	2	File type. For tty files, it is always set to zero (a data file). Valid file types are:

Name	Value	Description
fcbftdata	0	Data file
fcbftdir	1	Directory file
fcbftimage	2	Image file
fcbftksamdata	3	KSAM data type file
fcbftksamkey	4	KSAM key type file
fcbftllimage	5	11 type image file
fcbftarchcont	6	Archive continuation file
fcbftencrypt	7	Encrypted file
fcbftsystem	8	System file
fcbftarchive	9	Archive file
	10-255	Reserved
	256-65535	User defined file types

fcbfilename	9	File name. For disk and tape files it contains the filename portion of the file designation. For tty files it contains the devicename.

fcbfileext	3	File extension. For tty fcbs this field is set to zero.
fcbfilevers	2	File version number. For tty fcbs this field is set to zero.
fcbdirfcbnum	4	Directory fcb number. The fcb number of the directory file containing this file. For tape and tty fcbs it contains zero.
fcbdirseqnum	2	Directory sequence number. The sequence number of the directory file containing this file. For tty fcbs this field contains zero.
fcbrecordsz	2	Default record size. For tty fcbs this field is set to 1.
fcbuserid	2	Owner id of the files owner.
fcbgroupid	2	Group id of the files owner.
fcbprotect	2	File protection field. For tty fcbs it contains the device protection.
fcbcreatemstim	4	The most significant 32 bits of the file creation date in system time format (year and day). For tty fcbs, it contains the year and day that the device was mounted.
fcbcreatelstim	4	The least significant 32 bits of the file creation date in system time format (hour, minute, ...). For tty fcbs, it contains the hour, minute, ... that the device was mounted.
fcbmodmstim	4	The most significant 32 bits of the date the file was last modified (year and day). For tty fcbs, it contains the year and day that the device was mounted.
fcbmodlstim	4	The least significant 32 bits of the date the file was last modified (hour, minute, second, tick). For tty fcbs, it contains the hour, minute, ... that the device was mounted.
fcbreserved	4	Reserved space
fcbphysicalsz	4	The physical size of the file in bytes. For tty fcbs, it is set to zero.
fcblogicalsz	4	The logical size of the file in bytes. For tty fcbs, it is set to zero.
fcbfileid	2	File id of the file. For tty fcbs, it is set to zero.

fcbrootextblk	180	file extent fields. There are 30 extent fields in a primary fcb. Each extent field is composed of 6 bytes. The first two bytes represent the number of sectors in that extent. The last four bytes are the logical sector number of the first sector in that extent. For tty fcbs, this field is set to zero.
(fcbtapedirlen)	(2)	For tape fcbs, the first two bytes of this field contain the length of the directory name associated with this file.
(fcbtapedirname)	(178)	The other 178 bytes contain the directory name.
fcbnotcksum	2	The fcb's notted checksum

The format of the last 242 bytes of the fcb for "continuation" fcbs (disk only) is as follows:

fcbcontextblk	240	File extent fields in a continuation fcb. There are 40 extent fields in a continuation fcb. Each extent field is composed of 6 bytes. The first two bytes represent the number of sectors in that extent. The last four bytes are the logical sector number of the first sector in that extent.
fcbnotcksum	2	The fcb's notted checksum

The process can obtain the fcb for any file currently opened with read and/or write access by the process on any device.

Related Privileges:

None.

Parameters:

lun	- Logical unit number of file whose fcb is requested.
cont	- Which part of the fcb for this file is desired. 0=primary fcb, 1=1st continuation fcb...
fcbuff	- Address of 256 byte buffer to receive the fcb. This buffer must be word aligned.

status - Address of a long word to receive the result of the operation.

Diagnostics:

erridxrange	(56)	The table ends before the specified occurrence.
errinvlfn	(132)	The logical unit number does not correspond to an open file.
ernoreadacc	(141)	The process does not have read-access to the file.

See Also:

_create	- Create a file
_open	- Open a file
_setfcb	- Write file control block

Assembler Calling Sequence:

```
%%sys$disk/sysincl.sys/fcbdisp.asm
push    lun                     ;value - logical unit number
push    cont                    ;value - continuation fcb number
push    fcbuff                  ;address - buffer to receive the fcb
push    status                  ;address - result of the operation
jsr     _getfcb                 ;get file control block
```

C Function Declaration:

```
#include "sys$disk/sysincl.sys/fcbdisp.h"
                                /* get file control block */
long                            /* returns result of the operation */
_getfcb (lun, cont, fcbuff)
        long lun;               /* logical unit number */
        long cont;              /* continuation fcb number */
        fcbtype *fcbuff;        /* buffer to receive the fcb */
```

FORTRAN Subroutine Declaration:

```
c                               ! get file control block
        subroutine _getfcb(lun, cont, fcbuff, status)
            integer*4 lun       ! logical unit number
            integer*4 cont      ! continuation fcb number
            character*(*) fcbuff ! buffer to receive the fcb
            integer*4 status    ! result of the operation
```

Pascal Procedure Declaration:

```
%%sys$disk/sysincl.sys/fcbdisp.pas
procedure _getfcb(              {** get file control block}
        lun     : longint;      {** logical unit number}
        cont    : longint;      {** continuation FCB number}
        fcbuff  : ^array_of_char;{** buffer to receive the FCB}
    var status  : longint       {** result of the operation}
); external;
```

Get file ID.

Description:

Retrieves the file id on an open file. The file id is a user specified identifier that can be associated with a file.

The file id can be retrieved on any disk file open for read access.

Related Privileges:

None.

Parameters:

lun	- The logical unit number of the open file whose file id is sought.
fid	- Address of a long word to receive the file id. The file id will be moved to the least significant 16 bit word of this long word.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinvlfn	(132)	The logical unit number does not correspond to an open file.
errnoreadacc	(141)	The process does not have read-access to the file.
errinvcloper	(173)	The device class is inappropriate for the operation.

See Also:

_setfid - Set file id

Assembler Calling Sequence:

```
push    lun                     ;value - logical unit number
push    fid                     ;address - file id
push    status                  ;address - result of the operation
jsr     _getfid                 ;get file id
```

C Function Declaration:

```
                                /* get file id */
long                            /* returns result of the operation */
```

```
    _getfid(lun, fid)
            long lun;                /* logical unit number */
            long *fid;               /* file id */
```

Fortran Subroutine Declaration:

```
    c                                ! get file id
            subroutine getfid(lun, fid, status)
                integer*4 lun        ! logical unit number
                integer*4 fid        ! file id
                integer*4 status     ! result of the operation
```

Pascal Procedure Declaration:

```
    procedure _getfid(               {** get file id}
            lun     : longint;       {** logical unit number}
        var fid     : longint;       {** file id}
        var status  : longint        {** result of the operation}
    ); external;
```

getfnam - Given a lun, return the filename.

Description:

Given the logical unit number (lun) of a file successfully opened for read and/or write access by the calling process, the filename for this file is returned. This will work on all classes of devices.

Related Privileges:

None.

Parameters:

lun	- Logical unit number (lun) of the file whose name you wish to receive.
fname	- Address of a 94 byte buffer to receive the filename. The string returned may be up to 93 significant characters followed by a null character.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinvlfn	(132)	The logical unit number does not correspond to an open file.
ernoreadacc	(141)	The process does not have read-access to the file.

See Also:

_pfdnam - Given a PFD address, return the filename.

Assembler Calling Sequence:

```
push    lun                       ;value - logical unit number
push    fname                     ;address - receives filename string
push    status                    ;address - result of the operation
jsr     _getfnam                  ;given a lun, return the filename
```

C function declaration:

```
                                  /* given a lun, return the filename */
long                              /* returns result of the operation */
_getfnam(lun, fname)
        long lun;                 /* logical unit number */
```

```
            char fname[94];          /* receive filename string */
```

Fortran Subroutine Declaration:

```
    c                                ! given a lun, return the filename
            subroutine getfna(lun, fname, status)
                integer*4 lun        ! logical unit number
                character*(94) fname ! receives filename string
                integer*4 status     ! result of the operation
```

Pascal Procedure Declaration:

```
    procedure  getfnam(              {** given a lun, return the filename}
            lun     : longint;       {** logical unit number}
            fname   : string[93];    {** receives filename string}
        var status  : longint        {** result of the operation}
    ); external;
```

Get file protection.

Description:

Retrieves the protection mask on an open file. The protection mask determines the type of access to the file granted to classes of users. Protection can be retrieved on any file open for read or write access on the system.

Related Privileges:

none	- Allows retrieval of the protection if the calling process has successfully opened the file for read access.

Parameters:

lun	- The logical unit number of the open file whose protection mask is sought.
prot	- Address of a long word to receive the protection mask. The least significant 16 bit word of this long word is divided into 4 nibbles. Each nibble corresponds to a class of users. The bits within each nibble represent the type of access that class of user is granted for the file. If the bit is set (1) the access is granted.

From the least to the most significant nibble the user classes are:

Ownr	- The file owner
Grp	- Processes with the same group id as the owner
Pub	- All other processes in the system
Sys	- Processes with system privilege

```
  Sys  Pub  Grp  Ownr
 |----|----|----|----|
 |DWRE|DWRE|DWRE|DWRE|
 |-------------------|
MSB                 LSB
```

From the least to the most significant bit within the nibbles, the access privileges are:

E	- Execute access
R	- Read access
W	- Write access
D	- Delete access

status - Address of a long word to receive the result of the operation.

Diagnostics:

errinvlfn	(132)	The logical unit number does not correspond to an open file.
errnoreadacc	(141)	The process does not have read-access to the file.

See Also:

_getdprt - Get device protection
_setdprt - Set device protection
_setfprt - Set file protection

Assembler Calling Sequence:

```
push    lun                       ;value - logical unit number
push    prot                      ;address - protection mask
push    status                    ;address - result of the operation
jsr     _getfprt                  ;get file protection
```

C Function Declaration:

```
                                  /* get file protection */
long                              /* returns result of the operation */
_getfprt(lun, prot)
        long lun;                 /* logical unit number */
        long *prot;               /* protection mask */
```

Fortran Subroutine Declaration:

```
c                                 ! get file protection
        subroutine getfpr(lun, prot, status)
            integer*4 lun         ! logical unit number
            integer*4 prot        ! protection mask
            integer*4 status      ! result of the operation
```

Pascal Procedure Declaration:

```
procedure _getfprt(               {** get file protection}
        lun     : longint;        {** logical unit number}
    var prot    : longint;        {** protection mask}
    var status  : longint         {** result of the operation}
); external;
```

Get amount of available memory.

Description:

Returns the amount of available memory in the general memory pool. The value is in units of 1024 bytes. Only represents the amount of memory available in free pages.

Related Privileges:

None.

Parameters:

siteid	- A long word containing the system id of the system whose amount of available memory is sought. A siteid of zero (0) corresponds to the system on which the calling process is executing.
fremem	- Address of a long word to receive the amount of available memory.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinvsiteid	(8)	The specified site id does not exist.

See Also:

Assembler Calling Sequence:

```
push    siteid                      ;value - system id
push    fremem                      ;address - free memory
push    status                      ;address - result of the operation
jsr     _getfre                     ;get amount of available memory
```

C function declaration:

```
                                    /* get amount of available memory */
long                                /* returns result of the operation */
_getfre (siteid, fremem)
        long siteid;                /* system id */
        long *fremem;               /* free memory */
```

Fortran Subroutine Declaration:

```
c                                   ! get amount of available memory
        subroutine getfre(siteid, fremem, status)
```

```
                 integer*4 siteid      ! system id
                 integer*4 fremem      ! free memory
                 integer*4 status      ! result of the operation
```

Pascal Procedure Declaration:

```
    procedure _getfre(                  {** get amount of available memory}
            siteid  : longint;          {** system id}
        var fremem  : longint;          {** free memory}
        var status  : longint           {** result of the operation}
    ); external;
```

Get file record size.

Description:

Retrieves the file record size on an open file. The file record size is the number of bytes returned when one record is requested from the operating system. All files have a default record size that was specified when the file was created. The default record size may be overridden when the file is subsequently opened for further access. This system call returns the current record size that the file system has defined for the open file.

Related Privileges:

None.

Parameters:

lun	- The logical unit number of the open file whose record size is sought.
result	- Address of a long word to receive the record size. The record size will be moved to the least significant 16-bit word of this long word.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinvlfn	(132)	The logical unit number does not correspond to an open file.

See Also:

_setfrsz - Set file record size

Assembler Calling Sequence:

```
push    lun                     ;value - logical unit number
push    result                  ;address - record size
push    status                  ;address - result of the operation
jsr     _getfrsz                ;get file record size
```

C Function Declaration:

```
                                /* get file record size */
long                            /* returns result of the operation */
_getfrsz(lun, result)
        long lun;               /* logical unit number */
        long *result;           /* file record size */
```

FORTRAN Subroutine Declaration:

```
c                               ! get file record size
        subroutine _getfrs(lun, result, status)
            integer*4 lun       ! logical unit number
            integer*4 result    ! file record size
            integer*4 status    ! result of the operation
```

Pascal Procedure Declaration:

```
procedure _getfrsz(             {** get file record size}
        lun     : longint;      {** logical unit number}
    var result  : longint;      {** file record size}
    var status  : longint       {** result of the operation}
); external;
```

Get file UIC.

Description:

Given the logical unit number of an open file, returns the user identification code (uic) which is composed of an owner id and a group id.

To successfully retrieve the uic of a file, the calling process must have successfully opened the file with read access. This system call is valid for files of any class.

Related Privileges:

None.

Parameters:

lun	- The logical unit number of the file whose uic is requested.
uic	- Address of a long word to receive the user identification code. This long word is divided into two fields. The most significant 16 bits constitute the owner id number. The least significant 16 bits constitute the group id number (identifying the group to which the user belongs).
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinvlfn	(132)	The logical unit number does not correspond to an open file.
errnoreadacc	(141)	The process does not have read-access to the specified file.

See Also:

_getduic	- Get device uic
_getuic	- Get process uic
_setduic	- Set device uic
_setfuic	- Set file uic
_setuic	- Set process uic

Assembler Calling Sequence:

```
push    lun                         ;value - logical unit number
push    uic                         ;address - user id code
push    status                      ;address - result of the operation
```

```
    jsr     _getfuic                        ;get file uic
```

C Function Declaration:

```
                                            /* get file uic */
    long                                    /* returns result of the operation */
    _getfuic(lun, uic)
            long lun;                       /* logical unit number */
            long *uic;                      /* user id code */
```

Fortran Subroutine Declaration:

```
    c                                       ! get file uic
            subroutine getfui(lun, uic, status)
                integer*4 lun               ! logical unit number
                integer*4 uic               ! user id code
                integer*4 status            ! result of the operation
```

Pascal Procedure Declaration:

```
    procedure _getfuic(                     {** get file uic}
            lun      : longint;             {** logical unit number}
        var uic      : longint;             {** user id code}
        var status   : longint              {** result of the operation}
    ); external;
```

Retrieve a global logical name.

Description:

Given an index into a system's global logical name table, returns the logical name and equivalence associated with that index.

Related Privileges:

None.

Parameters:

index	- which entry in the logical name table is desired.
siteid	- Site id of the system whose gobal logical name table is being accessed. Zero (0) corresponds to the system on which the calling process is executing.
lname	- Address of a 94 byte buffer to receive the logical name. String will be null terminated. (up to 93 valid characters plus a null)
equiv	- Address of a 94 byte buffer to receive the equivalent string associated with the logical name. (up to 93 valid characters plus a null)
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errprcsnotfnd	(2)	The specified process is not in the system process table.
errinvsiteid	(8)	The specified site id does not exist.
erridxrange	(56)	The table ends before the specified occurrence.

See Also:

_assign	- Assign a logical name
_gassign	- Assign a global logical name
_getlog	- Retreive a logical name
_trans	- Translate a logical name

Assembler Calling Sequence:

```
push    index                   ;value - index into the table
push    siteid                  ;value - system id
push    lname                   ;address - logical name
push    equiv                   ;address - equivalent
push    status                  ;address - result of the operation
jsr     _getglb                 ;retrieve a global logical name
```

C function declaration:

```
                                    /* retrieve a global logical name */
long                                /* returns result of the operation */
_getglb( index, siteid, lname, equiv)
        long index;                 /* index into the table */
        long siteid;                /* system id */
        char lname[94];             /* logical name */
        char equiv[94];             /* equivalent */
```

Fortran Subroutine Declaration:

```
c                                   ! retrieve a global logical name
        subroutine getglb(index, siteid, lname, equiv, status)
            integer*4 index         ! index into the table
            integer*4 siteid        ! system id
            character*94 lname      ! logical name
            character*94 equiv      ! equivalent
            integer*4 status        ! result of the operation
```

Pascal Procedure Declaration:

```
procedure _getglb(                  {** retrieve a global logical name}
        index   : longint;          {** index into the table}
        siteid  : longint;          {** system id}
    var lname   : string[93];       {** logical name}
    var equiv   : string[93];       {** equivalent}
    var status  : longint           {** result of the operation}
); external;
```

Get installed files.

Description:

This call is used to obtain a list of installed files. Given an index into the system table of installed files, this call returns the corresponding entry which is composed of the name of the installed file (in fcb.seq format) and a privilege mask indicating which privileges the file is granted.

Related Privileges:

None.

Parameters:

siteid - A long word containing the system id number of the system whose table of privileged images is requested. A siteid of zero (0) corresponds to the system on which the calling process is executing.

index - The index into the system table of the file whose name and privilege are requested. The first entry in the table has an index of zero.

fcbnam - Address of a string to receive the name of the file in fcb.seq format. The returned name may contain up to 93 significant characters and will be null terminated.

priv - Address of a long word to receive the privilege mask. The privilege mask is a bit mask of privileges to be assigned to process when it is created using _crproc. Privileges are bit encoded as follows:

Bit Name	Bit #	Description
pcbpvsetpriv	0	setpriv
pcbpvsystem	1	system
pcbpvreadphys	2	readphys
pcbpvwritephys	3	writephys
pcbpvsetprior	4	setprior
pcbpvchngsuper	5	chngsuper
pcbpvbypass	6	bypass
pcbpvoperator	7	operator
pcbpvaltuic	8	altuic
pcbpvworld	9	world
pcbpvgroup	10	group
	11-31	Reserved.

status - Address of a long word to receive the result of the operation.

Diagnostics:

errinvsiteid	(8)	The specified site id does not exist.
erridxrange	(56)	The table ends before the specified occurrence.
errundevnam	(131)	The MCS does not recognize the devicename. Is the device mounted?

See Also:

_deinst - Deinstall privileged file
_install - Install privileged file

Assembler Calling Sequence:

```
push    siteid                  ;value - system id
push    index                   ;value - index into table
push    fcbnam                  ;address - fcb.seq file name
push    priv                    ;address - privilege mask
push    status                  ;address - result of the operation
jsr     _getinst                ;Get installed privileged file
```

C Function Declaration:

```
                                /* get installed privileged file */
long                            /* returns result of the operation
_getinst(siteid, index, fcbnam, priv)
        long siteid;            /* system id */
        long index;             /* index into table */
        char fcbnam[94];        /* fcb.seq file name */
        long *priv;             /* privilege mask */
```

Fortran Subroutine Declaration:

```
c                               ! get installed privileged file
        subroutine getins(siteid, index, fcbnam, priv, status)
            integer*4 siteid    ! system id
            integer*4 index     ! index into table
            character*94 fcbnam ! fcb.seq file name
            integer*4 priv      ! privilege mask
            integer*4 status    ! result of the operation
```

Pascal Procedure Declaration:

```
procedure _getinst(             {** get installed privileged file}
        siteid  : longint;      {** system id}
        index   : longint;      {** index into table}
    var fcbnam  : string[93];   {** fcb.seq file name}
    var priv    : longint;      {** privilege mask}
    var status  : longint       {** result of the operation}
); external;
```

Retrieve a logical name.

Description:

Given an index into a given process's logical name table, returns the logical name and equivalence associated with that index.

Related Privileges:

None	- Allows retrieval of logical names from tables of processes with the same user and group id (uic) as the current process.
group	- Allows retrieval of logical names from tables of processes with the same group id as the current process.
world	- Allows retrieval of logical names from tables of any process in the system.

Parameters:

index	- which entry in the logical name table is desired.
pid	- Process id of the process whose logical name table is being accessed. 0=current process, -1=parent process.
lname	- Address of a 94 byte buffer to receive the logical name. String will be null terminated. (up to 93 valid characters plus a null). If an error is detected, this buffer will remain unmodified.
equiv	- Address of a 94 byte buffer to receive the equivalent string associated with the logical name. (up to 93 valid characters plus a null) If an error is detected, this buffer will remain unmodified.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
errprcsnotfnd	(2)	The specified process is not in the system process table.
erridxrange	(56)	The table ends before the specified occurrence.

See Also:

_assign	- Assign a logical name
_gassign	- Assign a global logical name

```
    _getglb  - Retreive a global logical name
    _gengy   - Get pid of ancestor process
    _trans   - Translate a logical name
```

Assembler Calling Sequence:

```
    push    index                       ;value - index into the table
    push    pid                         ;value - process id
    push    lname                       ;address - logical name
    push    equiv                       ;address - equivalent
    push    status                      ;address - result of the operation
    jsr     _getlog                     ;retrieve a logical name
```

C function declaration:

```
                                        /* retrieve a logical name */
    long                                /* returns result of the operation */
    _getlog( index, pid, lname, equiv)
            long index;                 /* index into the table */
            long pid;                   /* process id */
            char lname[94];             /* logical name */
            char equiv[94];             /* equivalent */
```

Fortran Subroutine Declaration:

```
    c                                   ! retrieve a logical name
            subroutine getlog(index, pid, lname, equiv, status)
                integer*4 index         ! index into the table
                integer*4 pid           ! process id
                character*94 lname      ! logical name
                character*94 equiv      ! equivalent
                integer*4 status        ! result of the operation
```

Pascal Procedure Declaration:

```
    procedure _getlog(                  {** retrieve a logical name}
            index   : longint;          {** index into the table}
            pid     : longint;          {** process id}
        var lname   : string[93];       {** logical name}
        var equiv   : string[93];       {** equivalent}
        var status  : longint           {** result of the operation}
    ); external;
```

getmlst - Get an entry from list of named shared memory areas

Description:

The operating system maintains a control structure for each named sharable memory area. _Getmlst is used to obtain a copy of one of these control structures. INDEX specifies an offset into the list of shared memory area control structures. A value of zero will reference the first entry in the list. If too large a value is specified, _getmlst will indicate an error. This system call is used to obtain a description of the named sharable memory areas which are defined in the system.

Related Privileges:

None.

Parameters:

siteid	- A long word containing the address of the system from whom the information is needed. If SITEID is zero, the current system is referenced.
index	- A long word which is the offset into the list of shared memory areas. A value of zero returns the first entry in the list.
bsize	- A long word containing the maximum size of the buffer BUFFER.
buffer	- Address of an area to receive a copy of the of the named sharable memory control structure.

Name	Length (bytes)	Description
nsm f_link	4	Forward link
nsm b_link	4	Back link
nsm struct size	2	Structure size in bytes
nsm id_tag	2	Structure id tag = $3542
nsm mod_pid	4	PID of last modifier
nsm ref_cnt	2	Structure reference count
nsm status	2	Status word

Bit Name	Bit #	Description
nsm linger bit	0	The linger bit
nsm node_linked_bit	1	Node linked into chain bit

nsm protection	2	Memory protection mask
nsm mem_size	4	Size of memory area in bytes

nsm lock_que	4	Access queue to region
nsm_uic	4	UIC of definer
nsm namelen	2	Length of name of memory area
nsm name	94	Name of memory area
nsm page_cnt	2	Number of pages in page list

retlen	- Address of a long word to receive the size of the control structure in units of bytes.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

erridxrange	(56)	The index is beyond the end of the table.
errinvadr	(4)	The memory address is not on a 4K page boundary.

See Also:

_defmem	- Define a named sharable memory area.
_udefmem	- Undefine a named sharable memory area.
_shrmem	- Share a named sharable memory area.
_ushrmem	- Unshare a named sharable memory area.
_setmuic	- Change owner of a named sharable memory area.
_setmprt	- Change protection of a named sharable memory area.

Assembler Calling Sequence:

```
    push    siteid                  ; value   - system site id
    push    index                   ; value   - sequence number
    push    lmtab                   ; value   - length of mtable
    push    mtable                  ; address - memory table
    push    retlen                  ; address - # of bytes transferred
    push    status                  ; address - result of the operation
    jsr     _getmlst                ; Get an entry from list of named
                                    ; shared memory areas
```

C Function Declaration:

```
                                    /* Get an entry from list of named */
                                    /* shared memory areas */
    long                            /* returns result of the operation */
    _getmlst(siteid, index, lmtab, mtable, retlen)
        long        siteid;         /* system site id */
        long        index;          /* sequence number */
        long        lmtab;          /* length of mtable */
        nsm         mtable;         /* memory table */
        long        *retlen;        /* # of bytes transferred */
```

FORTRAN Subroutine Declaration:

```
c                               ! Get an entry from list of named
c                               ! shared memory areas
        getmls(siteid, index, lmtab, mtable, retlen, status)
            integer*4 siteid    ! system site id
            integer*4 index     ! sequence number
            integer*4 lmtab     ! length of mtable
            character*1024 mtable ! memory table
            integer*4 retlen    ! # of bytes transferred
            integer*4 status    ! result of the operation
```

PASCAL Procedure Declaration:

```
                                {** get an entry from list of named}
    procedure  getmlst(         {** shared memory areas}
            siteid  : longint;  {** system site id }
            index   : longint;  {** sequence number }
            lmtab   : longint;  {** length of mtable}
            mtable  : nsm;      {** memory table}
        var retlen  : longint;  {** # of bytes transferred}
        var status  : longint   {** result of the operation}
    ); external;
```

Get nodename from site ID.

Description:

This SVC returns the name of a node that is associated with the specified site ID.

Related Privileges:

none	- No privileges are needed to execute this SVC.

Parameters:

siteid	- This parameter is a long word containing the site ID for which the nodename is desired.
nname	- This parameter is the address of a string buffer in which will be placed the name of the node for the specified siteid. A nodename always begins with two underscores. The string is null terminated.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinvsiteid	(8)	The specified site ID does not exist.
errnoclass	(185)	The device class handler was not loaded when the system was booted.

See Also:

_getnsid - Get site ID from nodename

Assembler Calling Sequence:

```
push    siteid                    ; value - site id
push    nname                     ; address - for nodename
push    status                    ; address - result of the operation
jsr     _getnnam                  ; get name of node
```

C Function Declaration:

```
                                      /* Get nodename for site id */
    long                              /* returns result of the operation */
    _getnnam(siteid, nname)
             long    siteid;          /* Site id */
             char    nname[94];       /* Returned nodename */
```

FORTRAN Subroutine Declaration:

```
    c                                  ! Get nodename from site id
             subroutine _getnna(siteid, nname, status)
                 integer*4 siteid      ! Site id
                 character*94 nname    ! Returned nodename
                 integer*4 status      ! Result of operation
```

Pascal Procedure Declaration:

```
                                       {** Get nodename from site id }
    procedure _getnnam(
            siteid  : longint;         {** Site id }
        var nname   : string[93];      {** Returned nodename }
        var status  : longint          {** Result of operation }
    );external;
```

Get site ID from nodename.

Description:

This SVC returns the site ID for a given nodename.

Related Privileges:

none	- No privileges are needed to execute this SVC.

Parameters:

nname	- This parameter is the address of a null terminated string which contains the nodename.
siteid	- This parameter is the address of a longword which will receive the site ID for the given nodename.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errnonodefnd	(53)	The nodename is not defined.
errnoclass	(185)	The device class handler was not loaded when the system was booted.

See Also:

_getnnam - Get nodename from site ID

Assembler Calling Sequence:

```
push    nname                   ; address - nodename
push    siteid                  ; address - for siteid
push    status                  ; address - result of the operation
jsr     _getnsid                ; get site id for nodename
```

C Function Declaration:

```
                                /* Get site id from nodename*/
long                            /* Returns result of the operation */
_getnsid(nname, siteid)
        char nname[94];         /* nodename */
        long *siteid;           /* Returned site id */
```

FORTRAN Subroutine Declaration:

```
c                                  !  Get site id from nodename
              subroutine _getnsi(nname, siteid, status)
                  character*94 nname ! nodename
                  integer*4 siteid   ! Returned site id
                  integer*4 status   ! Result of operation
```

Pascal Procedure Declaration:

```
                                   {** Get site id from nodename }
     procedure _getnsid(
             nname    : string[93];    {** nodename }
         var siteid   : longint;       {** Returned site id }
         var status   : longint        {** Result of operation }
     );external;
```

Get process control block.

Description:

Given the process ID (PID) of a process in the system, copy the process control block (PCB) for that process into the buffer of the calling process.

CAUTION: The format of the process control block may change with each release of the operating system. The current definition is included in each release in the file named /SYSINCL.SYS/PRCSDISP.*. The name of the record is "pcbtable", i.e. in your program, you can declare a variable whose type is "pcbtable".

The format of the PCB is as follows:

Name	Length (bytes)	Description
pcbnextlink	4	Forward link to next pcb on same priority level
pcbbacklink	4	Backward link to previous pcb on same priority level
pcbsysidnum	2	Contains the system ID number (the most significant word of the PID)
pcbidnum	2	Contains the least significant word of the PID
pcbname	16	The process name
pcbstatus	4	A bit encoded long word representing the process status. If the bit is asserted (1), the corresponding status applies.

Bit name	Bit #	Description
pcbsttoabort	0	Process is to be scheduled for deletion (i.e. the next time this process is scheduled, send it to the delete process routines)
	1	Reserved
pcbsttohibernate	2	Process is to be hibernated

pcbstabrinprgs	3	Process is currently being deleted. (i.e. process is currently executing the delete process routines)
pcbstexhinprgs	4	Process is executing its exit handler
pcbstrealtime	5	Process is in real time mode
pcbstswapped	6	Process has been swapped
pcbsthaschild	7	Process is in a child wait state
pcbstnocontc	8	Process may receive CTRL/C without aborting
	9	Reserved
pcbsterrreport	10	Process is reporting a system error
	11	Reserved
pcbstextndfcb	12	Process is extending the fcb.sys file
pcbstbadseclog	13	Process is logging a bad sector
pcbstksam	14	Process is accessing a KSAM file
	15	Reserved
pcbstcrprcs	16	Process is loading an image
pcbstcleanup	17	Set when closing files when dying
pcbstinque	18	Process is waiting in a queue
pcbstcrashdisp	19	If set, suppress crash displays
pcbstalarmset	20	An alarm has been set
pcbstsupervisor	21	The call was issued while the processor was in supervisor mode
pcbstmulcrps	22	Multiple create process is in progress.
pcbstdisperr	23	If set, a crash report has been displayed
pcbsttracing	24	If set, process is tracing
pcbstfppending	25	If set, a floating point exception is pending
pcbstsurrogate	26	If set, this is an NSP for networking

		pcbstsurrchild	27	If set, this is the child of a surrogate
			28-31	Reserved
pcbtimeslice	2	The process time slice value, i.e., the maximum amount of time (specified in .01 milliseconds. That is, a time slice of 100 represents 1 millisecond.) that the non-real time process will be allowed to run each time it is scheduled.		
pcbmathtype	1	The type of floating point hardware in use The valid types are: 1 - skyl board 2 - ndp2 board 3 - ffpl board		
pcbmathptr	1	The math pointer. Contains the index of this process's window on the hardware floating point board.		
pcbprsize	2	The number of pages of memory currently allocated to this process. Each page is 4K bytes.		
pcbprivilege	2	The privileges granted to the current process. This is a bit encoded field. The privilege is granted when the corresponding bit is set.		
		Bit Name	Bit #	Description
		pcbpvsetpriv	0	setpriv - Process may assign more privileges than it currently has.
		pcbpvsystem	1	system - Process has system access to files
		pcbpvreadphys	2	readphys - Process can do physical read operations to devices and memory
		pcbpvwritephys	3	writephys - Process can do physical write operations to devices and memory
		pcbpvsetprior	4	setprior - Process can increase the process priority
		pcbpvchngsuper	5	chngsuper - Process can change to supervisor mode of execution

		pcbpvbypass	6	bypass - Process can access files and devices independently of file protection
		pcbpvoperator	7	operator - Process can perform operator functions
		pcbpvaltuic	8	altuic - Process can have access to files as though it had the same user and group id (uic) as the owner of the process image
		pcbpvworld	9	world - Process can affect any process in the system
		pcbpvgroup	10	group - Process can affect any process with the same group id as itself
		pcbpvnetwork	11	network - Process can do network accesses
		pcbpvsetattr	12	setattr - Process can modify its attributes
			13-15	Reserved
pcbuserid	2	The owner ID of the process (most significant word of the uic)		
pcbgroupid	2	The group ID of the process (least significant word of the uic)		
pcbchildpcbadr	4	Address of the pcb for the child process of this process		
pcbparntpcbadr	4	Address of the pcb for the parent process of this process		
pcbcurpriority	2	The current priority level		
pcbalarmtime	8	The date and time at which to issue the alarm		
pcbtimeout	8	The date and time at which the current operation will time out		
pcbnondelcnt	2	Non-delete count		
pcbcriticalcnt	2	Critical code count		
pcbusp	4	The user stack pointer		
pcbssp	4	The system stack pointer		
pcbevntfl	4	The process event flags		
pcbimgsiteid	2	Site ID of the image file		

pcbattributes	2	Attributes pertaining to the current process. This is a bit encoded field. The attribute is given when the corresponding bit is set. Note that these offsets are defined for being in the high order word of a longword. Because it is only a word in the PCB, if you access the PCB directly you will have to subtract 16 from these numbers.

Bit Name	Bit #	Description
pcbattrdesencrypt	16	If set, do network encryption with DES algorithm
pcbattrfastencrypt	17	If set, do network encryption with fast algorithm
pcbattruser1	23	If set, user attribute bit 1
pcbattruser2	24	If set, user attribute bit 2
pcbattruser3	25	If set, user attribute bit 3
pcbattruser4	26	If set, user attribute bit 4
pcbattrnowatchdog	27	If set, cannot be killed by WATCHDOG utility
pcbattrnotswappable	28	If set, cannot swap this process
pcbattrprezeromem	29	If set, pages are zeroed as they are allocated
pcbattrpostzeromem	30	If set, pages are zeroed as they are released
pcbattrforceset	31	If set, other set bits will be set

pcbimgdevseqnum	2	The mount sequence number of the device that contains the image file from which this process was initiated
pcbimgfcbnum	4	The fcb number of the image file from which this process was initiated
pcbimgseqnum	2	The sequence number of the image file from which this process was initiated
pcbstacktop	4	Address of the top of the system stack
pcbparabortsts	4	Address of where to put status in parent
pcbexithdr	4	Address of the process's exit handler
pcbabortreason	4	Reason code why this process terminated

pcblogiclink	4	Address of the logical name table for process
pcblogicque	4	Queue for linking logical names
pcbdefdevadr	4	Address of the device table for the default device for this process
pcbdefdevseqnum	2	The mount sequence number of the default device for this process
pcbdeffcbnum	4	fcb number for the current default directory
pcbdefseqnum	2	sequence number for the current default directory
pcbdefstrlen	2	Length of the default device string
pcbdefdiradr	4	Address of the default directory string
pcbdefdirlen	2	Length of the default directory string
pcbofpadr	4	List head to open files
pcbkpfdadr	4	List head to open ksam files
pcbqueadr	4	Address of the pcb of next entry in whatever queue this process is waiting in.
pcbnetpcktnum	2	Network packet number
pcbtrapvecs	64	Trap handler addresses
pcb0divide	4	Divide by zero trap handler address
pcbchktrap	4	Check trap handler address
pcbtrapv	4	Overflow trap handler address
pcbtracetrap	4	Trace trap handler address
pcbline1010	4	1010 emulation trap handler address
pcbline1111	4	1111 emulation trap handler address
pcbdefexithand	4	Define exit tran handler
pcbfpinthand	4	Floating point interrupt handler
pcbtrapreserved	16	Reserved space for future trap handlers
pcbloaderaddr	4	Address of loader routine
pcbevntflque	4	Queue for event flag synchronization
pcbtrapreturn	4	Trap 0 return address
pcbtrapnum	2	The current trap number
pcbmailptr	4	Address of the head node for pending mail
pcbmailque	4	Queue for processes waiting for mail
pcbdefaultprot	2	The default protection mask
pcbaltuserid	2	The user ID number of the image file
pcbaltgroupid	2	The group ID number of the image file
pcbhibercnt	2	Count of how many times this process has been hibernated
pcbschedcnt	4	Count of how many times this process has been scheduled.
pcbnsmaddr	4	List head for named shared memory regions that are currently mapped into this process
pcbnetpageaddr	4	Holds network packet page address
pcbmldrlisthead	8	List head for control information by various MCS loaders.
pcballochdr	4	List head for devices that are allocated to this process

pcborigprivilege	2	Holds original privileges process was created with before any installed privileges were added in.
pcbdefaultnode	4	Contains siteid of current default node
pcbcurtrapnum	4	The number of current SVCs being executed
pcbcurtrapprm	4	The stack address of current trap parameters
pcbremotepid	4	If this is an NSP, this is PID of originator
pcbremoteuic	4	If this is an NSP, this is UIC of originator
pcbremotepriv	2	If this is an NSP, this is priv of originator
pcbrctadr	4	List head for remote connection table
pcbbasepriority	2	Holds base priority level
pcbcurstate	4	Index into scheduling queues for current state

Queue Name	Offset	Description
pcbcst_toswapin	0	List for processes to be swapped in
pcbcst_active	4	List for active processes
pcbcst_asleep	8	All processes above here are in normal sleeps
pcbcst_iowait	8	List for processes in I/O wait
pcbcst_hibernate	12	List for processes in hibernation
pcbcst_childwait	16	List for processes in child wait
pcbcst_sqsize	20	Holds size of this scheduling queue

pcbswaptslice	2	Holds # of timeslices after swapin to get
pcbremotetslice	2	If this is an NSP, timeslice of originator
pcbremoteattr	2	If this is an NSP, attributes of originator
pcbremoteprior	2	If this is an NSP, priority of originator
pcbnoswapcnt	2	If non-zero, process is swap critical
pcbpagecnt	2	Holds size of this pcb in pages
pcbreserved	16	Reservedspace
pcbidfield	2	Table ID tag value
pcbidtag	$3333	Table ID value
pcbmemory	1024	The process's memory mapping registers
pcbdevstr	94	The default device/directory string

Related Privileges:

None.

Parameters:

pid	- Process ID of the process whose PCB is desired.
pcbuff	- Address of the buffer to receive the PCB
len	- The number of bytes requested. This number of bytes will be copied into the users buffer.
retlen	- Address of where to return the number of bytes actually copied into the users buffer.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
errprcsnotfnd	(2)	The specified process is not in the system process table.

See Also:

_gengy	- Get PID of ancestor process
_getpid	- Get process ID (PID) from name
_getpnam	- Get process name from PID
_prclst	- Get PIDs on a priority level

Assembler Calling Sequence:

```
push    pid                     ;value - process id
push    pcbuff                  ;address - buffer to receive pcb
push    len                     ;value - length of buffer
push    retlen                  ;address - # of bytes transferred
push    status                  ;address - result of the operation
jsr     _getpcb                 ;get process control block
```

C Function Declaration:

```
#include "sys$disk/sysincl.sys/pcbdisp.h"
                                /* get process control block */
long                            /* returns result of the operation */
_getpcb(pid, pcbuff, len, retlen)
        long pid;               /* process id */
        pcbtable *pcbuff;       /* buffer to receive pcb */
        long len;               /* length of buffer */
        long *retlen;           /* # of bytes transferred */
```

FORTRAN Subroutine Declaration:

```
    c                                 ! get process control block
             subroutine _getpcb(pid, pcbuff, len, retlen, status)
                 integer*4 pid        ! process id
                 character*(*) pcbuff ! buffer to receive pcb
                 integer*4 len        ! length of buffer
                 integer*4 retlen     ! # of bytes transferred
                 integer*4 status     ! result of the operation
```

Pascal Procedure Declaration:

```
    %%sys$disk/sysincl.sys/pcbdisp.pas
    procedure _getpcb(                {** get process control block}
            pid     : longint;        {** process id}
            pcbuff  : ^array_of_char; {** buffer to receive PCB}
            len     : longint;        {** length of buffer}
        var retlen  : longint;        {** # of bytes transferred}
        var status  : longint         {** result of the operation}
    ); external;
```

Get process ID (PID) from name.

Description:

This system call returns the process id (pid) of the highest priority process whose name matches the name supplied in the call. If there is more than one process with the specified name, the pid of the process closest to being scheduled again will be returned.

Related Privileges:

None.

Parameters:

siteid	- A long word containing the siteid of the system on which the named process is executing. A siteid of zero (0) corresponds to the system on which the calling process is executing.
pname	- Address of a 17 byte buffer containing the name of the process whose pid is requested. The process name is null terminated with up to 16 valid characters.
pid	- Address of a long word to receive the process id.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errprcsnotfnd	(2)	The specified process is not in the system process table.
errinvsiteid	(8)	The specified site id does not exist.

See Also:

_gengy	- Get pid of ancestor process
_getpnam	- Get process name from pid
_prclst	- Get pid's on a priority level

Assembler Calling Sequence:

```
push    siteid                    ;value - system id
push    pname                     ;address - process name
push    pid                       ;address - process id
push    status                    ;address - result of the operation
jsr     _getpid                   ;get process id (pid) from name
```

C function declaration:

```
                                        /* get process id (pid) from name
    long                                /* returns result of the operation */
    _getpid(siteid, pname, pid)]
            long siteid;                /* system id */
            char pname[17];             /* process name */
            long *pid;                  /* process id */
```

Fortran Subroutine Declaration:

```
    c                                   ! get process id (pid) from name
            subroutine getpid(siteid, pname, pid, status)
                integer*4 siteid        ! system id
                character*17 pname      ! process name
                integer*4 pid           ! process id
                integer*4 status        ! result of the operation
```

Pascal Procedure Declaration:

```
    procedure _getpid(                  {** get process id (pid) from name}
            siteid  : longint;          {** system id}
            pname   : string[16];       {** process name}
        var pid     : longint;          {** process id}
        var status  : longint           {** result of the operation}
    ); external;
```

:

Get process name from PID.

Description:

Given a process id (pid) returns a null terminated 17 byte string containing the process name.

Related Privileges:

None.

Parameters:

pid	- Process id of the desired process.
pname	- Address of a 17 byte null terminated string to receive the process name. Allows up to 16 significant characters plus a null bytes.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errprcsnotfnd	(2)	The specified process is not in the system process table.

See Also:

_gengy	- Get pid of ancestor process
_getpid	- Get process id (pid) from name
_prclst	- Get pid's on a priority level

Assembler Calling Sequence:

```
push    pid                         ;value - process id
push    pname                       ;address - receives the process name
push    status                      ;address - result of the operation
jsr     _getpnam                    ;get process name from pid
```

C function declaration:

```
                                    /* get process name from pid */
long                                /* returns result of the operation */
_getpnam(pid, pname)
        long pid;                   /* process id */
        char pname[17];             /* receives the process name */
```

Fortran Subroutine Declaration:

```
c                                   ! get process name from pid
```

```
                subroutine getpna(pid, pname, status)
                    integer*4 pid            ! process id
                    character*17 pname       ! receives the process name
                    integer*4 status         ! result of the operation
```

Pascal Procedure Declaration:

```
    procedure _getpnam(                 {** get process name from pid}
            pid      : longint;         {** process id}
        var pname    : string[16];      {** string to receive process name}
        var status   : longint          {** result of the operation}
    ); external;
```

Get the current file position.

Description:

Given a valid logical unit number (lun), returns the current file position. This is specified as the relative record position (relative to the front of the file) of the next record to be read or written.

Related Privileges:

None.

Parameters:

lun	- Logical unit number of desired file.
recnum	- Address of a long word to receive the record position.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinvlfn	(132)	The logical unit number does not correspond to an open file.

See Also:

_read	- Read from an open file
_setpos	- Set the current file position
_write	- Write to an open file

Assembler Calling Sequence:

```
push    lun                         ;value - logical unit number
push    recnum                      ;address - position
push    status                      ;address - result of the operation
jsr     _getpos                     ;get the current file position
```

C function declaration:

```
                                    /* get the current file position */
long                                /* returns result of the operation */
_getpos(lun, recnum)
        long lun;                   /* logical unit number */
        long *recnum;               /* position */
```

Fortran Subroutine Declaration:

```
c                                   ! get the current file position
```

```
                subroutine getpos(lun, recnum, status)
                    integer*4 lun          ! logical unit number
                    integer*4 recnum       ! position
                    integer*4 status       ! result of the operation
```

Pascal Procedure Declaration:

```
    procedure _getpos(                {** get the current file position}
            lun     : longint;        {** logical unit number}
        var recnum  : longint;        {** position}
        var status  : longint         {** result of the operation}
    ); external;
```

Get a process's priority.

Description:

This call allows a process to get its own scheduler priority or the priority of another process. There are 16 priority levels numbered 0 to 15. Priority level 0 is the highest.

Related Privileges:

None.

Parameters:

pid	- A long word containing the process ID of the process whose priority is to be obtained. 0 refers to the current process, -1 refers to the parent of the current process.
priort	- Address of a long word to receive the priority level.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errprcsnotfnd	(2)	The specified process is not in the system process table.

See Also:

_prirat	- Set priority scheduling ratio
_setpri	- Set process's priority
_settmsl	- Change scheduling time slice

Assembler Calling Sequence:

```
push    pid                     ;value - process id
push    priort                  ;address - priority level
push    status                  ;address - result of the operation
jsr     _getpri                 ;get process's priority
```

C Function Declaration:

```
                                        /* get process's priority */
    long                                /* returns result of the operation */
   _getpri (pid, priort)
           long pid;                    /* process id */
           long *priort;                /* priority level */
```

FORTRAN Subroutine Declaration:

```
    c                                   ! get process's priority
           subroutine _getpri(pid, priort, status)
               integer*4 pid            ! process id
               integer*4 priort         ! priority level
               integer*4 status         ! result of the operation
```

Pascal Procedure Declaration:

```
    procedure _getpri(                  {** get process's priority }
           pid      : longint;          {** process id}
       var priort   : longint;          {** priority level}
       var status   : longint           {** result of the operation}
    ); external;
```

Get default protection mask.

Description:

Retrieves the default protection mask for the process. This is the mask that is used for any files created by the current process and any child processes of the current process.

Related Privileges:

None.

Parameters:

prot - Address of a long word to receive the file protection mask. The least significant 16 bit word of this return value is divided into 4 nibbles. Each nibble corresponds to a class of users. The bits within each nibble represent the type of access that class of user is granted for this file. If the bit is set (1) the access is granted.

From the least to the most significant nibble the user classes are:

Ownr	- file owner
Grp	- processes with the same group id as the owner
Pub	- all other processes in the system
Sys	- processes with SYSTEM privilege

```
  Sys  Pub  Grp  Ownr
 |----|----|----|----|
 |DWRE|DWRE|DWRE|DWRE|
 |-------------------|
MSB                 LSB
```

From the least to the most significant bits within the nibbles, the access privileges are:

E	- Execute access
R	- Read access
W	- Write access
D	- Delete access

Diagnostics:

None.

See Also:

_create - Create a file
_creats - Simplified file creation
_defprot - Set default protection mask
_setfprt- Set file protection

Assembler Calling Sequence:

```
push    prot                      ;address - protection mask
jsr     _getprot                  ;get default protection mask
```

C Function Declaration:

```
                                  /* get default protection mask */
void                              /* no result */
_getprot ( prot )
        long *prot;               /* protection mask */
```

Fortran Subroutine Declaration:

```
c                                 ! get default protection mask
        subroutine getpro(prot)
            integer*4 prot        ! protection mask
```

Pascal Procedure Declaration:

```
procedure _getprot(               {** get default protection mask}
    var prot    : longint         {** protection mask}
); external;
```

Get process privilege.

Description:

This call allows a process to inspect the privileges assigned in the process privilege word of any process in the system.

Related Privileges:

None.

Parameters:

pid - Process id of the process whose privileges are to be returned. A pid of 0 represents the current process. A pid of -1 represents the parent of the current process.

priv - Address of a long word to receive the privilege mask containing a bit mask of privileges assigned to the specified process.

Bit Name	Bit	Description
pcbpvsetpriv	0	setpriv
pcbpvsystem	1	system
pcbpvreadphys	2	readphys
pcbpvwritephys	3	writephys
pcbpvsetprior	4	setprior
pcbpvchngsuper	5	chngsuper
pcbpvbypass	6	bypass
pcbpvoperator	7	operator
pcbpvaltuic	8	altuic
pcbpvworld	9	world
pcbpvgroup	10	group
pcbpvnetwork	11	network
pcbpvsetattr	12	setattr
	13-32	Reserved.

status - Address of a long word to receive the result of the operation.

Diagnostics:

errprcsnotfnd (2) The specified process is not in the system process table.

See Also:

```
_crproc  - Create a new process
_gettmsl - Get scheduling time slice
_setpri  - Set process priority
_setprv  - Set process privilege
_settmsl - Change scheduling time slice
```

Assembler Calling Sequence:

```
push    pid                     ;value - process id
push    priv                    ;address - privilege mask
push    status                  ;address - result of the operation
jsr     _getprv                 ;get process privilege
```

C Function Declaration:

```
                                /* get process privilege */
long                            /* returns result of the operation */
_getprv(pid, priv)
         long pid;              /* process id */
         long *priv;            /* privilege mask */
```

FORTRAN Subroutine Declaration:

```
c                               ! get process privilege
         subroutine _getprv(pid, priv, status)
             integer*4 pid      ! process id
             integer*4 priv     ! privilege mask
             integer*4 status   ! result of the operation
```

Pascal Procedure Declaration:

```
procedure _getprv(              {** get process privilege}
        pid     : longint;      {** process id}
    var priv    : longint;      {** privilege mask}
    var status  : longint       {** result of the operation}
); external;
```

Get names of rotor list elements.

Description:

This call returns the names of all the devices assigned to a specified rotor list.

Related Privileges:

none	- No privileges are needed to execute this SVC.

Parameters:

rtrnam	- This parameter is the address of a null terminated string which contains the name of the rotor list whose elements are to be returned. The name supplied will be logically translated before use.
devnms	- This parameter is the address of a string buffer in which will be placed the names of the devices which comprise the rotor list named in the rtrnam parameter. All names are separated by commas. The string is null terminated.
maxlen	- This parameter contains the maximum length of the devnms string. Up to this number of characters will be returned.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errnamenull	(80)	The specified name must not be null.
errnoname	(82)	The specified name does not exist.

See Also:

_alloc	- Allocate an available device.
_dealloc	- Deallocate an allocated device.
_getalc	- Get names of allocated devices.
_getrtr	- Get rotor list names.
_setrtr	- Assign devicenames to a rotor list.

Assembler Calling Sequence:

```
push    rtrnam                  ; address - name of rotor list
push    devnms                  ; address - element list
```

```
     push    maxlen                   ; value - length of devnms
     push    status                   ; address - result of the operation
     jsr     _getrel                  ; get names of rotor list elements
```

C Function Declaration:

```
                                      /* get names of rotor list elements */
     long                             /* returns result of the operation */
     _getrel(rtrnam, devnms, maxlen);
             char    rtrnam[94];      /* name of rotor list */
             char    devnms[1025];    /* element list */
             long    maxlen;          /* length of devnms */
```

FORTRAN Subroutine Declaration:

```
     c                                ! get names of rotor list elements
           subroutine _getrel(rtrnam, devnms, maxlen, status);
              character*94 rtrnam     ! name of rotor list
              character*1024 devnms   ! element list
              integer*4    maxlen     ! max length of devnms in bytes
              integer*4    status     ! result of the operation
```

Pascal Procedure Declaration:

```
     procedure _getrel(               {** get names of rotor list elements}
             rtrnam  : string[93];    {** name of rotor list}
         var devnms  : string[1024];  {** element list}
             maxlen  : longint;       {** length of devnms}
         var status  : longint        {** result of the operation}
     ); external;
```

Get rotor list names.

Description:

This SVC allows a process to obtain the name of the Nth rotor list known to WMCS. The first rotor list known to WMCS has an index of 0. In order to get the name of all the rotor lists, call this SVC using increasing rotor name indices until the error "erridxrange" is returned. Because rotor lists may be defined and/or deleted between calls to the SVC, the name of the Nth rotor list may not persist over time. If a reliable record of each rotor list is desired, the calling process should be running in real-time mode between the first and last call to this SVC.

Related Privileges:

None.

Parameters:

siteid	- The site ID of the machine or node that contains the rotor list.
index	- The index into the list of rotor names where the first rotor name has an index of 0.
rtrnam	- Address of where to store the rotor name. The rotor name string will be null terminated. The string provided must be at least 10 characters long, allowing for up to 9 significant characters plus a null.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

erridxrange	(56)	The table ends before the specified occurrence.

See Also:

_alloc	- Allocate an available device
_dealloc	- Deallocate an allocated device
_getalc	- Get names of allocated devices
_getrel	- Get names of rotor list elements
_setrtr	- Assign devicenames to a rotor list

Assembler Calling Sequence:

```
push    siteid                  ;value - site ID of rotor list
push    index                   ;value - rotor name index
push    rtrnam                  ;address - name of indexth rotor
push    status                  ;address - result of the operation
jsr     _getrtr                 ;get rotor list names
```

C Function Declaration:

```
                                /* get rotor list names */
long                            /* returns result of the operation */
_getrtr(siteid, index, rtrnam)
         long siteid;           /* site ID of rotor list */
         long index;            /* rotor name index */
         char rtrnamp[10]       /* name of indexth rotor list */
```

FORTRAN Subroutine Declaration:

```
c                               ! get rotor list names
         subroutine _getrtr(siteid, index, rtrnam, status)
             integer*4 siteid   ! site ID of rotor list
             integer*4 index    ! rotor name index
             character*10 rtrnam ! name of indexth rotor list
             integer*4 status   ! result of the operation
```

Pascal Procedure Declaration:

```
procedure _getrtr(              {** get rotor list names}
        siteid  : longint;      {** site ID of rotor list}
        index   : longint;      {** rotor name index}
    var rtrnam  : string[9];    {** name of indexth rotor list}
    var status  : longint       {** result of the operation}
); external;
```

Get internal tick count.

Description:

Returns a 64-bit unsigned value which is the number of .01 second ticks since the system was last booted.

Related Privileges:

None.

Parameters:

siteid	- A long word containing the system id of the system whose tick clock is to be read. A system id of zero (0) corresponds to the system on which the calling process is executing.
mstime	- Address of a long word to receive the most significant 32 bits of the tick clock
lstime	- Address of a long word to receive the least significant 32 bits of the tick clock
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinvsiteid	(8)	The specified site id does not exist.

See Also:

_gettim - Get the current date and time
_settim - Set system date and time

Assembler Calling Sequence:

```
push    siteid                  ;value - system id
push    mstime                  ;address - most significant 4 bytes
push    lstime                  ;address - least significant 4 bytes
push    status                  ;address - result of the operation
jsr     _gettic                 ;get internal tick count
```

C function declaration:

```
                                /* get internal tick count */
long                            /* returns result of the operation */
_gettic (siteid, mstime, lstime)
        long siteid;            /* system id */
        long *mstime;           /* most significant 4 bytes */
        long *lstime;           /* least significant 4 bytes */
```

Fortran Subroutine Declaration:

```
c                                     ! get internal tick count
          subroutine gettic(siteid, mstime, lstime, status)
              integer*4 siteid        ! system id
              integer*4 mstime        ! most significant 4 bytes
              integer*4 lstime        ! least significant 4 bytes
              integer*4 status        ! result of the operation
```

Pascal Procedure Declaration:

```
procedure _gettic(                    {** get internal tick count}
        siteid  : longint;            {** system id}
    var mstime  : longint;            {** most significant 4 bytes}
    var lstime  : longint;            {** least significant 4 bytes}
    var status  : longint             {** result of the operation}
); external;
```

Get the current date and time.

Description:

Read the current system time of day clock. Returns 8 bytes containing the contents of the system time of day clock. The format of the date and time within these 8 bytes is as follows, where byte 0 is the most significant byte.

Bytes	Description
0,1	The current year (counted from A.D. 0). Example, 1983.
2,3	The day of the year (1..365 or 1..366)
4	The hour of the day (0..23)
5	The minute of the hour (0..59)
6	The second of the minute (0..59)
7	The fraction of a second (in 100ths of a second) (0..99)

Related Privileges:

None.

Parameters:

siteid	- A long word containing the system id of the system whose time of day clock is to be read. A siteid of zero (0) corresponds to the system on which the calling process is executing.
mstime	- Address of a long word to receive the most significant 32 bits of the date and time (actually the year and day of the year)
lstime	- Address of a long word to receive the least significant 32 bits of the date and time (actually the hour, minute, second and fraction of a second)
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinvsiteid	(8)	The specified site id does not exist.

See Also:

_gettic - Get internal tick count
_settim - Set system date and time

Assembler Calling Sequence:

```
push    siteid                  ;value - system id
push    mstime                  ;address - most significant 4 bytes
```

```
    push    lstime                      ;address - least significant 4 byte
    push    status                      ;address - result of the operation
    jsr     _gettim                     ;get the current date and time
```

C function declaration:

```
                                        /* get the current date and time */
    long                                /* returns result of the operation */
    _gettim(siteid, mstime, lstime)
            long siteid;                /* system id */
            long *mstime;               /* most significant 4 bytes */
            long *lstime;               /* least significant 4 bytes */
```

/

Fortran Subroutine Declaration:

```
    c                                   ! get the current date and time
            subroutine gettim(siteid, mstime, lstime, status)
                integer*4 siteid        ! system id
                integer*4 mstime        ! most significant 4 bytes
                integer*4 lstime        ! least significant 4 bytes
                integer*4 status        ! result of the operation
```

Pascal Procedure Declaration:

```
    procedure _gettim(                  {** get the current date and time}
            siteid  : longint;          {** system id}
        var mstime  : longint;          {** most significant 4 bytes}
        var lstime  : longint;          {** least significant 4 bytes}
        var status  : longint           {** result of the operation}
    ); external;
```

Get scheduling time slice.

Description:

Retrieve the scheduling time slice of a process. Time slice is the maximum amount of time the non-real time process will be allowed to execute each time it is scheduled. When the time slice is expired, other processes are allowed to execute according to the scheduling algorithm.

Each time slice increment is .01 milliseconds. A time slice value of 5000 allows the process to execute up to one twentieth of a second (50 milliseconds) each time it is scheduled. A time slice value less than 10 results in the process not running at all.

Note that processes will not always use their full time slice. When an I/O operation is performed, the process often relinquishes control and loses the rest of it's time slice.

Related Privileges:

None.

Parameters:

pid	- The process id of the process whose time slice is to be retrieved. 0 represents the current process; -1 represents the parent of the current process.
tslice	- Address of a long word to receive the time slice value (0..65535). Represents the scheduling time slice in .01 milliseconds.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errprcsnotfnd	(2)	The specified process is not in the system process table.

See Also:

_prirat	- Set priority scheduling ratio
_setpri	- Change process's priority
_settmsl	- Change scheduling time slice

Assembler Calling Sequence:

```
push    pid                        ;value - process id
```

```
    push    tslice                      ;address - time slice
    push    status                      ;address - result of the operation
    jsr     _gettmsl                    ;Get scheduling time slice
```

C function declaration:

```
                                        /* Get scheduling time slice */
    long                                /* returns result of the operation */
    _gettmsl(pid, tslice)
            long pid;                   /* process id */
            long *tslice;               /* time slice */
```

Fortran Subroutine Declaration:

```
    c                                   ! get scheduling time slice
            subroutine gettms(pid, tslice, status)
                integer*4 pid           ! process id
                integer*4 tslice        ! time slice
                integer*4 status        ! result of the operation
```

Pascal Procedure Declaration:

```
    procedure _gettmsl(                 {** get scheduling time slice}
            pid     : longint;          {** process id}
        var tslice  : longint;          {** time slice}
        var status  : longint           {** result of the operation}
    ); external;
```

Get process UIC.

Description:

Given a process id (pid) returns the user identification code (uic) which is composed of an owner id and a group id.

Related Privileges:

None.

Parameters:

pid	- The process id of the process whose uic is requested. A process id of 0 corresponds to the calling process. A process id of -1 corresponds to the parent of the calling process.
uic	- Address of a long word to receive the user identification code. This long word is divided into two fields. The most significant 16 bits constitute the owner id number. The least significant 16 bits constitute the group id number (identifying the group to which the owner belongs).
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errprcsnotfnd	(2)	The specified process is not in the system process table.

See Also:

_getduic	- Get device uic
_getfuic	- Get file uic
_setduic	- Set device uic
_setfuic	- Set file uic
_setuic	- Set process uic

Assembler Calling Sequence:

```
push    pid                         ;value - process id
push    uic                         ;address - user id code
push    status                      ;address - result of the operation
jsr     _getuic                     ;get process uic
```

C Function Declaration:

```
                                    /* get process uic */
```

```
    long                                /* returns result of the operation *,
    _getuic(pid, uic)
            long pid;                   /* process id */
            long *uic;                  /* user id code */
```

Fortran Subroutine Declaration:

```
    c                                   ! get process uic
            subroutine getuic(pid, uic, status)
                integer*4 pid           ! process id
                integer*4 uic           ! user id code
                integer*4 status        ! result of the operation
```

Pascal Procedure Declaration:

```
    procedure _getuic(                  {** get process uic}
            pid     : longint;          {** process id}
        var uic     : longint;          {** user id code}
        var status  : longint           {** result of the operation}
    ); external;
```

Get device status with LUN.

Description:

Given the LUN of a currently mounted device, this system call copies the device table and device status into user specified buffers. (Contrast this system call with _getdst).

WARNING: The format of the device table may change with each release. The current definition is included in each release in the file /SYSINCL.SYS/DEVIDISP.*. The record definition is named "devicetable", i.e. in your program you can declare a variable of type "devicetable."

The device table for a device contains the information maintained about the device by the class handler. The device table is divided into two parts. The first part is device independent, and the second part is device class dependent. The device independent part is as follows:

Name	Length (bytes)	Description
dtnextlink	4	Pointer to the next device table
dtbacklink	4	Pointer to the previous device table
dtdevname	8	The user supplied devicename
dtclass	2	Contains the device class. Valid options are:

Class Name	Value	Description
dtclassttyspc	0	Character device (ttyspc)
dtclasstty	1	Character device (tty)
dtclasstapespc	2	Tape device (tapespc)
dtclasstape	3	Tape device (tape)
dtclassdiskspc	4	Disk device (diskspc)
dtclassdisk	5	Disk device (disk)
dtclassnetspc	6	Network dev. (networkspc)
dtclassnet	7	Network device (network)
dtclasspipespc	8	Pipe device (pipespc)
dtclasspipe	9	Pipe device (pipe)
dtclasssyncspc	10	BSC device (syncspc)
dtclasssync	11	BCS device (sync)
dtclassquespc	12	Queue device (quespc)
dtclassque	13	Queue device (que)

		dtclassnondevspc 14	Non-dev device(nondevspc)
		dtclassnondev 15	Non-dev device (nondev)
dtrefcount	2	The number of files currently open on the device	
dtdriveid	4	Internal drive ID	
dtallocpid	4	The PID of the process that has this device allocated	
dtsiteid	2	The site ID of this device	
dtseqnum	2	The mount sequence number of this device. This will be unique for each device on the machine.	
dtdefuserid	2	The default userid for this device. This will be loaded into the DTUSERID variable every time the DTREFCOUNT variable goes to zero.	
dtdefgroupid	2	The default groupid for this device. This will be loaded into the DTGROUPID variable every time the DTREFCOUNT variable goes to zero.	
dtdefprotect	2	The default protection mask for this device. This will be loaded into the DTPROTECT variable every time the DTREFCOUNT variable goes to zero.	
dtclassptr	4	Address of the class handler for this device	
dtdriverptr	4	Address of the device driver for this device	
dtflags	2	Device flags. This is a bit encoded word.	

Bit Name	Bit #	Description
dtflfcbflushmode	4	Current flush mode for disk fcbs
dtflchflushmode	5	Current flush mode for disk cache
dtflflushing	6	Set if device is now being flushed
dtflwriteprot	7	Set if the device is write protected
dtflwritebuf	8	Set if the tape buffer has been modified
dtflfileopen	11	Set if a tape file is open
dtfleot	12	Set if tape is at physical end of tape
dtfleof	13	Set if tape is at logical end of file
dtflsessionestb	15	Set if a session is currently established

dtfcbptr	4	Address of the file control block of the first open file on the device. A list head pointer. (Used for disks only)
dtblksz	2	Block size for the device

dtuserid	2	Owner id portion of the uic. Corresponds to the owner of the device.
dtgroupid	2	Group id portion of the uic. Corresponds to the owner of the device.
dtprotect	2	The device protection flags. Uses the same format at the file protection flags.
dtmntmstime	4	The most significant 32 bits (year and day) of the date and time the device was mounted
dtmntlstime	4	The least significant 32 bits (hour, minute, second and tick) of the date and time the device was mounted
dtidfield	2	Table identifier flag
dtidtag	$5555	This is the table id value for this table

For TTY, PIPE, SYNC, and NONDEV class devices, the second part of the table is defined as follows:

Name	Length (bytes)	Description
dttyreadacc	1	The read access count (the number of times this device has been opened for read access)
dttyreadlock	1	The read lock count (the number of times this device has been opened with read lock)
dttywriteacc	1	The write access count (the number of times this device has been opened for write access)
dttywritelock	1	The write lock count (the number of times this device has been opened with write lock)
dttywriteqh	4	The write queue header
dttyreadqh	4	The read queue header

For TAPE class devices, the second part of the table is defined as follows:

Name	Length (bytes)	Description
dttpreadahead	2	Read ahead flag
dttpfilseqno	4	Sequence number of currently open file or next file to be opened.
dttpcachesz	2	Number of elements in tape cache
dttpcacheadr	4	Address of cache header
dttpskpcache	4	Address of special cache header for non-buffered commands, i.e., skip, get or set status, write file mark
dttpnextblk	4	Next logical block number in the currently open file
dttpreadpos	2	Actual block number to be read next physically

For DISK class devices, the second part of the table is defined as follows:

Name	Length (bytes)	Description
dtdkflags	2	Disk class flags. This is a bit encoded word. Bit Name / Bit # / Description dtdkflautoflush 0 If set do auto flushing dtdkflreadahead 1 If set do readahead dtdkflforcedwrite 2 If set do forced writes on all writes
dtdksecshfcnt	2	The sector shift count
dtdkdefalloc	2	The initial file allocation
dtdksecalloc	2	The secondary file allocation
dtdkchreadmin	2	Non-modified cache minimum size
dtdkmaxuserch	2	Number of cache elements (minus 1) that can be consumed in a single request to the OS
dtdkszmaxch	2	Size of stack area in bytes used to hold the addresses of used cache elements ((devcldsmaxcache+2)*4)
dtdkcachecolsz	2	The number of columns in the cache
dtdkcachesze	2	The number of cache sectors
dtdkchaddr	4	Address of disk cache column table
dtdkbmpos	4	Bitmap file's next allocation location
dtdkfcbbmpos	4	Fcbbitmap file's next allocation location
dtdkfcbptr	4	Address of fcb for FCB.SYS
dtdkdirptr	4	Address of fcb for ROOTDIR.DIR

dtdkfcbbitptr	4	Address of fcb for FCBBITMAP.SYS
dtdkbitptr	4	Address of fcb for BITMAP.SYS
dtdkalocseqh	4	Allocate disk queue head
dtdkalocfcbqh	4	Allocate fcb queue head

For NETWORK class devices, the second part of the table is defined as follows:

Name	Length (bytes)	Description
dtnkreadacc	1	The read access count (the number of times this device has been opened for read access)
dtnkreadlock	1	The read lock count (the number of times this device has been opened with read lock)
dtnkwriteacc	1	The write access count (the number of times this device has been opened for write access)
dtnkwritelock	1	The write lock count (the number of times this device has been opened with write lock)
dtnkflags	2	Network class flags. This is a bit encoded word. Bit Name / Bit # / Description dtnkflvcdriver / 0 / If set, this is a virtual circuit driver
dtnkwriteqh	4	The write access queue header
dtnkreadqh	4	The read access queue header
dtnkhwrite	4	Pointer to network layer write routine
dtnkhuninit	4	Pointer to network layer uninit routine

For QUEUE class devices, the second part of the table is defined as follows:

Name	Length (bytes)	Description
dtqucbptr	4	Contains the address of control block page which is the communication block between the QUEUE class handler and the queue manager process

dtqufhptr	4	Contains the address of the queue control file header page		
dtquwriteoper	4	Contains how many write operations have been performed on the QUEUE		
dtquflags	2	QUEUE class flags. Bit encoded word.		
		Bit Name	Bit #	Description
		dtqufldefcrp	0	If set, a default create process record is defined. A user can redirect I/O directly to the QUEUE.
		dtquflqmres	1	If set, the queue manager process is to remain resident at all times.
		dtquflqmnodie	2	If set, we are in critical code and the queue manager process cannot die.
		dtquflclosed	3	If set, the queue is marked as closed. No new entries may be queued.
		dtquflhalted	4	If set, the queue is marked as halted. No pending entries will be executed.
		dtquflclean	5	If set, there are no entries in the queue control files.

The device status is a device class dependent 128 byte table. It is maintained by the device driver for each device.

NOTE: The device status table may change with each release of the operating system. The current definition is included in each release in the file named /SYSINCL.SYS/DSTATDISP.*. The name of the record included in that file is "devicestatus," i.e., in your program you can declare a variable whose type is "devicestatus."

The device status table is divided into two parts. The first half is device independent and is composed of the following fields:

Name	Length (bytes)	Description
dsclassid	2	The device class. Valid classes are: (Note that these names are defined in the devtdisp.* files)

Class Name	Value	Description
dtclassttyspc	0	Character device (ttyspc)
dtclasstty	1	Character device (tty)
dtclasstapespc	2	Tape device (tapespc)
dtclasstape	3	Tape device (tape)
dtclassdiskspc	4	Disk device (diskspc)
dtclassdisk	5	Disk device (disk)
dtclassnetspc	6	Network dev. (networkspc)
dtclassnet	7	Network device (network)
dtclasspipespc	8	Pipe device (pipespc)
dtclasspipe	9	Pipe device (pipe)
dtclasssyncspc	10	BSC device (syncspc)
dtclasssync	11	BCS device (sync)
dtclassquespc	12	Queue device (quespc)
dtclassque	13	Queue device (que)
dtclassnondevspc	14	Non-dev device(nondevspc)
dtclassnondev	15	Non-dev device (nondev)

dsdriverid	2	The unique id number for this device driver
dsblksz	2	Block size of the device (e.g. sector size)
dsharderr	2	The hard error count for the device
dssofterr	2	The soft error count for the device
dsreadoper	4	The number of read operations on this device
dswriteoper	4	The number of write operations on this device
dsmaxnumdev	2	Maximum # of devices this driver can handle
dscurnumdev	2	Number of devices currently mounted using this device driver
dsnumtoretry	2	Number of times to retry before reporting a hard error
dserrorreason	4	This contains the hardware error code for the last error received on this device
dsreserved	32	Reserved
dsnexttableptr	4	Address of next device status table

The second half of the device status table is device class dependent For TAPE class devices the second part is defined as follows:

Name	Length (bytes)	Description
dstpstatus	2	Tape device status. A bit encoded word. Bit name / bit # / Description dstpready 0 Set if device ready dstpintpend 1 Set if interrupt pending dstprewinding 2 Set if tape rewinding dstpbotdetect 3 Set if device is at physical BOT dstpeotdetect 4 Set if device is at physical EOT dstpwriteprot 5 Set if tape is write protected
dstpflagsl	2	Tape status information. A bit encoded word. Bit name / bit # / Description dstpdoraw 0 0=Read after write disabled 1=Read after write enabled dstperrintenb 1 0=Error interrupts are enabled 1=Error interrupts are disabled
dstpspeed	1	Tape speed. Values are: 0 - Reserved dstpspeed12ips 1 - 12 ips dstpspeed25ips 2 - 25 ips dstpspeed30ips 3 - 30 ips dstpspeed50ips 4 - 50 ips dstpspeed90ips 5 - 90 ips dstpspeed100ips 6 - 100 ips dstpspeed125ips 7 - 125 ips
dstpdensity	1	Tape density. Values are: 0 - Reserved dstpdens800bpi 1 - 800 bpi dstpdens1600bpi 2 - 1600 bpi dstpdens3200bpi 3 - 3200 bpi dstpdens6250bpi 4 - 6250 bpi dstpdens6400bpi 5 - 6400 bpi
dstpiopbcnt	2	Number of IOPBs allocated to device
dstpcachesz	2	Number of cache elements allocated to device
dstpreserved	46	Reserved
dstpuserfield	8	User defined status

For DISK class devices the second half of the device status table is defined as follows:

Name	Length (bytes)	Description		
dsdkintfac	2	Disk interleave factor		
dsdkiopbcnt	2	Number of IOPB's allocated to the drive		
dsdknumbsect	4	The number of sectors on the volume		
dsdksectrack	2	The number of sectors on a track		
dsdkheads	2	The number of heads on the device		
dsdkcylinders	2	The number of cylinders on the volume		
dsdkflagsl	2	Disk status information. A bit encoded word.		
		Bit Name	Bit #	Description
		dsdkdensity1	0	Device density
		dsdkdensity2	1	
		dsdkdenssignle		00 - Single density
		dsdkdensdouble		01 - Double density
		dsdkdensquad		10 - Quad density
		dsdkdensreserve		11 - Reserved
		dsdkdoraw	3	If set, do Read after write verify
		dsdkwriteprot	4	If set, Device write protected
		dsdkseekdir	15	Current seek direction
		dsdkseekincr		0 - Increasing cylinder numbers
		dsdkseekdecr		$8000 - Decreasing cylinder numbers
dsdkcurcyl	2	Current cylinder position		
dsdkcachesz	2	Number of sectors in the disk cache		
dsdkentryname	16	A null terminated string containing the name of this type of drive		
dsdkreserved	20	Reserved		
dsdkuserfield	8	User Defined status		

For TTY class devices the second half of the device status table is defined as follows:

Name	Length (bytes)	Description
dstymoderegl	1	Uart mode register 1. This byte is bit encoded as follows:

Bit Name	Bit #	Description
dstymrlbaudfacl	0	Baud factor
dstymrlbaudfac2	1	
dstymrlsyncl		00 - sync 1 x clock rate
dstymrlasyncl		01 - async 1 x clock rate
dstymrlasyncl6		10 - async 16 x clock rate
dstymrlasync64		11 - async 64 x clock rate
dstymrlcharlenl	2	Character length
dstymrlcharlen2	3	definition
dstymrldw5bit		00 - 5 data bits
dstymrldw6bit		01 - 6 data bits
dstymrldw7bit		10 - 7 data bits
dstymrldw8bit		11 - 8 data bits
dstymrlparityctrl	4	Parity control
dstymrlpardis		0 - disable parity
dstymrlparenb		1 - enable parity
dstymrlparitytype	5	Parity type
dstymrlparodd		0 - odd parity
dstymrlparevn		1 - even parity
dstymrlstopbitsl	6	Async mode # of stop bits
dstymrlstopbits2	7	Async mode # of stop bits
dstymrlbinv		00 - invalid
dstymrlsbl		01 - 1 stop bit
dstymrlsbl5		10 - 1.5 stop bits
dstymrlsb2		11 - 2 stop bits
dstymrltransctrl	6	Sync mode transparent
dstymrlnormal		0 - normal
dstymrltrans		1 - transparent
dstymrlnumsync	7	Sync mode # of syncs
dstymrlsyncdouble		0 - double sync
dstymrlsyncsingle		1 - single sync

dstymodereg2 1 Uart mode register 2. This byte is bit encoded as follows:

Bit Name	Bit #	Description
dstymr2baudrtl	0	The baud rate
dstymr2baudrt2	1	Baud rate continued
dstymr2baudrt3	2	Baud rate continued
dstymr2baudrt4	3	Baud rate continued
dstymr2baud50		0000 - 50 baud
dstymr2baud75		0001 - 75 baud
dstymr2baudl10		0010 - 110 baud
dstymr2baudl345		0011 - 134.5 baud

		dstymr2baud150	0100 - 150 baud
		dstymr2baud300	0101 - 300 baud
		dstymr2baud600	0110 - 600 baud
		dstymr2baud1200	0111 - 1200 baud
		dstymr2baud1800	1000 - 1800 baud
		dstymr2baud2000	1001 - 2000 baud
		dstymr2baud2400	1010 - 2400 baud
		dstymr2baud3600	1011 - 3600 baud
		dstymr2baud4800	1100 - 4800 baud
		dstymr2baud7200	1101 - 7200 baud
		dstymr2baud9600	1110 - 9600 baud
		dstymr2baud19200	1111 - 19200 baud
		dstymr2recvclock 4	Receiver clock
		dstymr2recextclk	0 - External clock
		dstymr2recintclk	1 - Internal clock
		dstymr2transclock 5	Transmitter clock
		dstymr2trnextclk	0 - External clock
		dstymr2trnintclk	1 - Internal clock
		6-7	Reserved
dstycmdreg	1	Uart command register.	Bit encoded.
		Bit Name Bit #	Description
		dstycrtransctrl 0	Transmitter control
		dstycrtcdis	0 - Disable transmitter
		dstycrtcenb	1 - Enable transmitter
		dstycrdtr 1	Data terminal ready
		dstycrdtrhigh	0 - DTR high
		dstycrdtrlow	1 - DTR low
		dstycrrecvcrtl 2	Receiver control
		dstycrrcdis	0 - Disable receiver
		dstycrrcenb	1 - Enable receiver
		dstycrforcebrk 3	Async force break
		dstycrbrknorm	0 - normal
		dstycrbrkforce	1 - force break
		dstycrsenddle 3	Sync send DLE
		dstycrdlenorm	0 - normal
		dstycrdlesend	1 - send DLE
		dstycrreseterror 4	Reset error
		dstycrnoreset	0 - normal
		dstycrreseterr	1 - reset error
		dstycrrts 5	Request to send
		dstycrrtshigh	0 - RTS high
		dstycrrtslow	1 - RTS low
		dstycropermodel 6	Operating mode

		dstycropermode2	7	Operating mode continued
		dstycromnormal		00 - Normal operation
		dstycromautoecho		01 - Async autoecho
		dstycromstripdle		01 - Sync strip DLE
		dstycromlocallp		10 - Local loop back
		dstycromremotelp		11 - Remote loop back
dstytermtype	1	Terminal type definition. This byte contains values for each type of terminal.		
		Value Name	**Value**	**Description**
			0-15	User defined types
			16-246	Reserved
		dstywit	247	WIT terminal
		dstyhydra	248	Hydra terminal
		dstyvt100	250	VT-100 terminal
		dstyvt52	251	VT-52 terminal
		dstyt7000	252	T-7000 terminal
		dstymg8000	253	MG-8000 terminal
		dstytvi912c	254	TVI 912 C terminal
		dstyvisual200	255	Visual 200 terminal
dstystatreg	1	Uart status register. Bit encoded.		
		Bit Name	**Bit #**	**Description**
		dstysrtransrdy	0	Transmitter buffer ready
		dstysrtranfull		0 - Transmitter full
		dstysrtranempty		1 - Transmitter empty
		dstysrrecvrdy	1	Receiver buffer ready
		dstysrrecvempty		0 - Receiver empty
		dstysrrecvfull		1 - Receiver full
		dstysrdschg	2	DSR or DCD change
		dstysrdsrnormal		0 - Normal
		dstysrdsrchange		1 - Change in DSR or DCD
		dstysrparityerr	3	Parity error
		dstysrparnormal		0 - Normal
		dstysrparerror		1 - Async parity error. Sync parity error or DLE received
		dstysroverrunerr	4	Overrun error
		dstysrovernormal		0 - Normal
		dstysrovererror		1 - Overrun error
		dstysrframingerr	5	Framing error
		dstysrframnormal		0 - Normal

		dstysrframerror		1 - Async framing error. Sync SYN char
		dstysrdcddetect	6	DCD Detect
		dstysrdcdhigh		0 - DCD high
		dstysrdcdlow		1 - DCD low
		dstysrdsrdetect	7	DSR Detect
		dstysrdsrhigh		0 - DSR high
		dstysrdsrlow		1 - DSR low
dstypacketterm	1	Holds code for packet termination characters		
		Value Name	Value	Description
		dstyptnoterm	0	Do not terminate packet on any control characters
		dstyptallterm	1	Terminate packets on all control characters
		dstyptcrterm	2	Terminate packet on carriage return <CR> character
dstyflagsl	2	Terminal status information. Bit encoded.		
		Bit Name	bit #	Description
		dstycontrolc	0	Control C enable (0 = enabled)
		dstyxonxoff	1	xon xoff enable (0 = enabled)
		dstycontrolx	2	Control X enable (0 = enabled)
		dstycontrolz	3	Control Z enable (0 = enabled)
		dstycontrolo	4	Control O enable (0 = enabled)
		dstytabmap	5	Tab map enable (1 = enabled)
		dstymask8bit	6	Mask 8th bit enable (0 = enabled)
		dstycontrolu	7	Control U enable (0 = enabled)
		dstybroadcast	8	Broadcast enable (0 = enabled)
		dstyhandshakel	9	Handshaking type
		dstyhandshake2	10	
		dstyhsbell		00 - No handshake, send bell
		dstyhssoft		01 - Software handshake

		dstyhshard		10 - Hardware handshake
		dstyhsnone		11 - No handshake, no bell
		dstyduplex	11	Full/half duplex (0 = full duplex)
		dstymodemctrl	12	Modem control enable (1 = enabled)
		dstyautobaud	13	Auto baud enable (1 = enabled)
		dstyremote	14	Remote enable (1 = enabled)
dstyinputcnt	2	Count of characters in input interrupt buffer		
dstyoutptcnt	2	Count of characters in output interrupt buffer		
dstycolumnpos	2	Current column position		
dstyinbufsz	2	Input buffer size in bytes		
dstyoutbufsz	2	Output buffer size in bytes		
dstywidth	2	The width of the given terminal screen		
dstylength	2	The length of the given terminal screen		
dstysubreadoper	4	Number of sub-read operations		
dstysubwriteoper	4	Number of sub-write operations		
dstyreserved	26	Reserved		
dstyuserfield	8	User defined status		

For PIPE class devices the second part of the device status table is defined as follows:

Name	Length (bytes)	Description
dsppreaderpid	4	Process ID of pending reader
dsppwriterpid	4	Process ID of pending writer
dspppipeid	4	The pipe's ID
dsppbuffersz	2	The buffer size in bytes
dsppbuffercnt	2	Number of characters in the pipe buffer
dsppreadque	4	Address of read queue
dsppwriteque	4	Address of write queue
dsppreserved	32	Reserved
dsppuserfield	8	User defined status

For SYNC class devices the second part of the device status table is defined as follows:

Name	Length (bytes)	Description		
dssymoderegl	1	Mode register 1 of the uart (See DSTYMODEREGl for bit definitions)		
dssymodereg2	1	Mode register 2 of the uart (See DSTYMODEREG2 for bit definitions)		
dssycmdreg	1	Command register of the uart (See DSTYCMDREG for bit definitions)		
dssytermtype	1	Terminal type definition. A binary value.		
		Value Name	Value	Description
		dssyibm3741	249	IBM 3741 terminal
		dssyibm2968	250	IBM 2968 terminal
		dssyibm2770	251	IBM 2770 terminal
		dssyibm3276	252	IBM 3276 terminal
		dssyibm3275	253	IBM 3275 terminal
		dssyibm2780	254	IBM 2780 RJE
		dssyibm3780	255	IBM 3780 RJE
dssystatreg	1	Status register of uart (See DSTYSTATREG for bit definitions)		
dssynumbsync	1	Number of sync characters to write		
dssyflagsl	2	Device Status flags. Bit encoded.		
		Bit Name	Bit #	Description
		dssymultipnt	0	0=point to point 1=multipoint
		dssyebcdic	1	0=ascii line 1=ebcdic line
		dssycrcccitt	2	0=crc-16 1=crc-ccitt
		dssylrc	3	0=crc (on above types) 1=lrc
		dssyasctoebcw	4	0=no translate on write 1=translate ascii to ebcdic on write
		dssyebctoascr	5	0=no translate on read 1=translate ebcdic to ascii on read
		dssytranstbl2	6	0=translate table 1 1=translate table 2
dssyinputcnt	2	Number of characters in input interrupt buffer		
dssyoutputcnt	2	Number of characters in output interrupt buffer		
dssyinbufsz	2	Input buffer size in bytes		

dssyoutbufsz	2	Output buffer size in bytes
dssyprevrderr	4	Error from previous un-verified read
dssyprevwrerr	4	Error from previous no-wait write
dssyprevrdtype	1	Type of previous read dssynontran - 0 Non-transparent read dssytran - 1 Transparent read
dssynumbtrpad	1	The number of trailing pads to write
dssyrecsize	2	Used in transparent mode with ITBs
dssyreserved	28	Reserved
dssyuserfield	8	User defined status

For NETWORK class devices the second part of the device status table is defined as follows:

Name	Length (bytes)	Description
dsnkflags	2	Device status flags. Bit encoded. Bit Name / Bit # / Description dsnkbyte 0 0=datagram mode, 1=byte mode dsnkmodemctrl 1 0=not enabled, 1=modem ctrl enabled
dsnkwindowsize	1	Window size the circuit will use
dsnkreserved	53	Reserved
dsnkuserfield	8	User defined status

For NONDEV class devices the second part of the device status table is defined as follows:

Name	Length (bytes)	Description
dsnduserfield	64	User defined status

For QUEUE class devices the second part of the device status table is defined as follows:

Name	Length (bytes)	Description
dsquassocdev	9	A null terminated string containing the name of the physical printer device
dsqusenddev	9	A null terminated string containing the name of the physical device that control messages are to be sent to
dsquformname	10	A null terminated string containing the current form name
dsqunumexec	2	This is the maximum number of entries that can execute concurrently
dsqucurnumexec	2	This is the number of entries that are currently active
dsquflags	2	Device Status flags. Bit encoded.

Bit Name	Bit #	Description
dsquflupdating	0	If set, currently updating queue control file
dsquflqmstay	1	If set, the queue manager process will remain running even when queue is empty
dsquflnorestart	2	If set, when the queue is mounted it does not restart jobs in the queue

Name	Length (bytes)	Description
dsqulength	2	This holds the length of the forms of the printer associated with this queue
dsquwidth	2	This hold sthe width of the forms of the printer associated with this queue
dsqunextentry	4	This is the entry number of the next entry to be enqued
dsqutype	1	This contains the type of queue this is. The values are:

Value Name	Value	Description
dsqutpprint	1	This is a print type queue
dsqutpjob	2	This is a job entry type queue

Name	Length (bytes)	Description
dsqubaseprior	1	This contains the priority that entries will be queued at if they specify the default priority
dsqureserved	20	Reserved
dsquuserfield	8	User defined status

Related Privileges:

None.

Parameters:

lun	- Logical unit number (LUN) of a file on the device whose status you wish to receive.
dtable	- Address of a buffer to receive the device table. This table must be word aligned.
ldtab	- Length of the device table. Up to this many bytes of the device table will be transferred to the user buffer.
dstat	- Address of a 128 byte buffer to receive the device status.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinvlfn	(132)	The logical unit number does not correspond to an open file.
errnoreadpriv	(144)	The process does not have Read Privilege for the file.

See Also:

_dismnt	- Dismount a logical device
_getdnam	- Get devicename
_getdst	- Get device status
_mount	- Mount a logical device
_setdst	- Set device status
_siodst	- Set device status with LUN

Assembler Calling Sequence:

```
%%sys$disk/sysincl.sys/devtdisp.asm
%%sys$disk/sysincl.sys/dstatdisp.asm
push    lun                     ;value - logical unit number
push    dtable                  ;address - device table
push    ldtab                   ;value - length of device table
push    dstat                   ;address - device status
push    status                  ;address - result of the operation
jsr     _giodst                 ;get device status
```

C Function Declaration:

```
#include "sys$disk/sysincl.sys/devtdisp.h"
#include "sys$disk/sysincl.sys/dstatdisp.h"
                                /* get device status with lun*/
long                            /* returns result of the operation */
_giodst(lun, dtable, ldtab, dstat)
        long lun;               /* logical unit number */
        devicetable *dtable;    /* device table */
        long ldtab;             /* length of device table */
        devicestatus *dstat;    /* device status */
```

FORTRAN Subroutine Declaration:

```
c                               ! get device status with lun
        subroutine _giodst(lun, dtable, ldtab, dstat, status)
            integer*4 lun       ! logical unit number
            character*(*) dtable ! device table
            integer*4 ldtab     ! length of device table
            character*(*) dstat ! device status
            integer*4 status    ! result of the operation
```

Pascal Procedure Declaration:

```
%%sys$disk/sysincl.sys/devtdisp.pas
%%sys$disk/sysincl.sys/dstatdisp.pas

procedure _giodst(              {** get device status with lun}
        lun     : longint;      {** logical unit number}
        dtable  : ^array_of_char;{** device table}
        ldtab   : longint;      {** length of device table}
        dstat   : ^array_of_char;{** device status}
    var status  : longint       {** result of the operation}
); external;
```

Receive interprocess mail.

Description:

Receive a message sent from another process. The message may be up to 3952 bytes long and may contain any data.

Related Privileges:

none	- Allows the process to receive mail addressed to itself or to another process with the same owner id and group id (uic) as the calling process.
group	- Allows the process to receive mail addressed to any process with the same group id as the calling process.
world	- Allows the process to receive mail addressed to any other process in the system.

Parameters:

rpid	- Process id of the process whose mail you wish to receive. A process id of 0 represents the current process. A process id of -1 represents the parent of the current process.
mail	- Address of a buffer to receive the message. If an error is detected, this buffer is not modified.
len	- Length of the mail buffer in bytes. This is the maximum number of characters that can be received.
timout	- The maximum time to wait for mail to become available for the receiving process. The time out is specified in .01 seconds.
pid	- Address of a long word to receive the pid of the sender.
retlen	- Address of a long word to receive the length of the message that was returned. If an error is detected, the value of this long word is set to zero.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
errprcsnotfnd	(2)	The specified process is not in the system process table.
errnomail	(20)	No interprocess mail, in system message table, for the process.
errtimeout	(128)	A request was not completed within the

specified time.

See Also:

_smail - Send interprocess mail

Assembler Calling Sequence:

```
push    rpid                        ;value - intended receiver
push    mail                        ;address - message buffer
push    len                         ;value - maximum message length
push    timout                      ;value - time out
push    pid                         ;address - senders pid
push    retlen                      ;address - actual message length
push    status                      ;address - result of the operation
jsr     _gmail                      ;receive interprocess mail
```

C function declaration:

```
                                    /* receive interprocess mail */
long                                /* returns result of the operation */
_gmail (rpid, mail, len, timout, pid, retlen)
        long rpid;                  /* intended receiver */
        char mail[3952];            /* message buffer */
        long len;                   /* maximum message length */
        long timout                 /* time out */
        long *pid;                  /* senders pid */
        long *retlen;               /* actual message length */
```

Fortran Subroutine Declaration:

```
c                                   ! receive interprocess mail
        subroutine gmail(rpid, mail, len, timout, pid, retlen, status)
            integer*4 rpid          ! intended receiver
            character*(*) mail      ! message buffer
            integer*4 len           ! maximum message length
            integer*4 timout        ! time out
            integer*4 pid           ! senders pid
            integer*4 retlen        ! actual message length
            integer*4 status        ! result of the operation
```

Pascal Procedure Declaration:

```
procedure _gmail(                   {** receive interprocess mail}
        rpid    : longint;          {** intended receiver}
        mail    : ^array_of_char;   {** message buffer}
        len     : longint;          {** maximum message length}
        timout  : longint;          {** time out}
    var pid     : longint;          {** senders pid}
    var retlen  : longint;          {** actual message length}
    var status  : longint           {** result of the operation}
); external;
```

hibern - Hibernate a process.

Description:

Remove a process from consideration by the scheduler. This will increment a hibernate refrence count and set the hibernate status bit so the process can no longer be scheduled. There are two ways to wake a hibernated process. A call to _wake will set the refrence count to zero and clear the hibernate status bit. On the other hand a call to _wakec will decrement the hibernate count and clear the hibernate status bit when the count goes to zero. A hibernated process will exist indefinitely in the process table but in a dormant state until either the process is terminated by another process, or is awakened by another process.

Related Privileges:

none	- Allows process to hibernate any process with the same owner id and group id (uic) as the calling process.
group	- Allows process to hibernate any process with the same group id as the calling process.
world	- Allows process to hibernate any process in the system.

Parameters:

pid	- Process ID of the process to be hibernated. 0 refers to the calling process, -1 refers to the parent of the calling process.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
errprcsnotfnd	(2)	The specified process is not in the system process table.

See Also:

_wait	- Pause for a period of time
_wake	- Wake a hibernated process
_wakec	- Wake a hibernated process with count

Assembler Calling Sequence:

```
    push    pid                        ;value - process id
    push    status                     ;address - result of the operation
    jsr     _hibern                    ;hibernate a process
```

C function declaration:

```
                                       /* hibernate a process */
    long                               /* returns result of the operation */
    _hibern(pid)
            long pid;                  /* process id */
```

Fortran Subroutine Declaration:

```
    c                                  ! hibernate a process
            subroutine hibern(pid, status)
                integer*4 pid          ! process id
                integer*4 status       ! result of the operation
```

Pascal Procedure Declaration:

```
    procedure _hibern(                 {** hibernate a process}
            pid     : longint;         {** process id}
        var status  : longint          {** result of the operation}
    );. external;
```

Install privileged file.

Description:

Allows a process to set the image file privileges on an open file, or to establish that a certain file is a device driver.

If the file is an image file, then when a process is created from this image, it will have all of the privileges specified by the _install system call, plus whatever privileges were specified by the creating (parent) process.

If a file containing a device driver is installed, then a process can mount a device using that driver without having to have operator privilege. That is, processes which do not have operator privilege cannot mount deviceswith drivers that are not installed. Note that the driver file need not be given any privileges.

If the specified file is already installed, the function performed by this system call is to redefine the privileges for the file. No error is returned.

Note that an installed file is identified by the device on which it resides and its fcb.seq number. The filename is not used to identify the file. That is, loading a new file with the same name as an installed file does not install that file. Also, renaming an installed file does not affect the fact that the file is installed.

This operation is valid on any disk file.

To successfully set file privileges, the calling process must have operator privilege, and must have successfully opened the file for write access. The calling process can set any privileges that it (the process) already has. It must have setpriv privilege to grant more privileges than the calling process has.

Related Privileges:

none	- The process cannot successfully install any file.
operator	- Allows the calling process to install files and to grant them any privileges that the calling process has.
setpriv	- If the calling process also has operator privilege, this privilege allows the calling process to install files and to grant that file any privilege.

Parameters:

siteid	- A long word containing the site id of the system on which the privileged process is to be installed. A siteid of zero corresponds to the system on which the calling process is executing.
fname	- Address of a null terminated string containing the name of the file whose privileges are to be set. The string will be translated automatically by WMCS to its logical equivalence. This string may contain up to 93 significant characters followed by a null.
priv	- The privilege mask contains a bit mask of privileges to be given to the file. If the value of this parameter is -1, the specified file is given the same privileges as the calling process. If the value of this parameter is not -1, it represents privileges which are bit encoded as follows:

Bit Name	Bit #	Description
pcbpvsetpriv	0	setpriv
pcbpvsystem	1	system
pcbpvreadphys	2	readphys
pcbpvwritephys	3	writephys
pcbpvsetprior	4	setprior
pcbpvchngsuper	5	chngsuper
pcbpvbypass	6	bypass
pcbpvoperator	7	operator
pcbpvaltuic	8	altuic
pcbpvworld	9	world
pcbpvgroup	10	group
pcbpvnetwork	11	network
pcbpvsetattr	12	setattr
	13-32	Reserved. Must be set to zero.

status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
errinapft	(12)	The file type is inappropriate for the given operation.
errnowriteacc	(142)	The process does not have write-access to the specified file.
errinvcloper	(173)	The device class is inappropriate for the operation.

See Also:

```
_crproc  - Create a new process
_deinst  - Deinstall privileged file
_getinst - Get installed privileged file
_mount   - Mount a logical device
```

Assembler Calling Sequence:

```
push    siteid                  ;value - system id
push    fname                   ;address - file name
push    priv                    ;value - privilege mask
push    status                  ;address - result of the operation
jsr     _install                ;install privileged file
```

C Function Declaration:

```
                                /* install privileged file */
long                            /* returns result of the operation */
_install(siteid, fname, priv)
         long siteid;           /* system id */
         char fname[94];        /* file name */
         long priv;             /* privilege mask */
```

FORTRAN Subroutine Declaration:

```
c                               ! install privileged file
         subroutine _instal(siteid, fname, priv, status)
             integer*4 siteid   ! system id
             character*94 fname ! file name
             integer*4 priv     ! privilege mask
             integer*4 status   ! result of the operation
```

Pascal Procedure Declaration:

```
procedure _install(             {** install privileged file}
        siteid  : longint;      {** system id}
        fname   : string[93];   {** file name}
        priv    : longint;      {** privilege mask}
    var status  : longint       {** result of the operation}
); external;
```

Close all ksam files.

Description:

Close any ksam files that have been opened by the current process. Note that this happens automatically when the process is deleted.

Related Privileges:

None.

Parameters:

siteid	- A long word containing the system id of the system on which all ksam files are to be closed. A siteid of zero (0) corresponds to the system on which the calling process is executing.
status	- Address of a long word to receive the result of the operation

Diagnostics:

errnoclass	(185)	The device class handler was not loaded when the system was booted.
errdevwrtprot	(269)	The specified device is write-protected.
		Device integrity errors

See Also:

_kclose - Close a ksam file
_kopen - Open a ksam file

Assember calling sequence:

```
    push    siteid                      ;value - system id
    push    status                      ;address - result of the operation
    jsr     _kclall                     ;close all ksam files
```

C function declaration:

```
                                        /* close all ksam files */
    long                                /* returns result of the operation */
    _kclall(siteid)
            long siteid;                /* system id */
```

Fortran Subroutine Declaration:

```
    c                                   ! close all ksam files
```

```
                subroutine kclall(siteid, status)
                    integer*4 siteid        ! system id
                    integer*4 status        ! result of the operation
```

Pascal Procedure Declaration:

```
    procedure  _kclall(                {** close all ksam files}
           siteid  : longint;          {** system id}
       var status  : longint           {** result of the operation}
    ); external;
```

Close a KSAM file.

Description:

This SVC closes a currently open KSAM file (both data and key files) that has been opened by the calling process. Any records still locked by the closing process are automatically unlocked.

_KCLOSE writes both the key and data files to disk if the flush flag is set. If the flush flag is set on a disk device, all disk cache buffers will be written to the device. If the device is a tape, the tape buffer is written to the device.

If the delete mode bit is set, the process must have write privilege to the directories containing the data and key files and delete privilege to both files for the files to be successfully deleted.

Related Privileges:

none	- The file will be closed. Allows optional deletion of the data and key files if the process has privileges as described above. Returns a warning if the process specified delete upon closing and does not have the required privileges.
altuic	- Allows the process to delete the files upon closing if the owner of the image file for the current process has privileges to the files as described above.
bypass	- Allows the process to delete the files upon closing independent of the process's privileges to the file.
system	- Allows the process to delete the files upon closing if the system has privileges to the files as described above.

Parameters:

lun	- The logical unit number of the file to be closed. The lun is obtained from _kopen or _kcreat.
mode	- Bit encoded long word specifying action to be taken upon closing. If the bit is zero (0), no action is performed. The following actions apply when the specified bit is set to one (1).

Bit Name	Bit #	Description
cldelete	0	Delete the data and key files after closing. If the file is currently open by another process, the actual deletion of the files is delayed until after all processes have closed the files.
clnotruncfile	1	No truncate - Specifies that when the disk file is closed, the extra physical sectors allocated to the file are not to be released. For tape devices, this bit specifies that the last block written to the tape should be written as a full sized block (as opposed to a variable sized block).
clnodelete	2	No delete - Overrides the delete upon closing request specified by the _open system call.
clforcedwrite	3	Forced write - Writes to the device all data in system buffers associated with this lun. If an error occurs it will be reported as a warning to the calling process. The file is always closed.
clsupalldelete	4	Suppress all deletes - Overrides all deletes that have been set for the file, i.e., opdelete or a delete set by a different process.
clzerodelete	5	Zero delete - Zero each sector of the file before deleting the file. This bit is only valid if the file is being deleted (via cldelete or some other way).
	6-31	Reserved. Must be set to zero.

status - Address of a long word to receive the result of the operation.

Diagnostics:

errinvlfn	(132)	The logical unit number does not correspond to an open file.
errnodelpriv	(146)	The process does not have Delete Privilege for the file.

errnoclass	(185)	The device class handler was not loaded when the system was booted.
errdevwrtprot	(269)	The specified device is write-protected.

See Also:

_delete	- Delete a file
_kclall	- Close all KSAM files
_kcreat	- Create a KSAM file
_kopen	- Open a KSAM file

Assember Calling Sequence:

```
push    lun                     ;value - logical unit number
push    mode                    ;value - mode word
push    status                  ;address - result of the operation
jsr     _kclose                 ;close a KSAM file
```

C Function Declaration:

```
                                /* close a KSAM file */
long                            /* returns result of the operation */
_kclose(lun, mode)
        long lun;               /* logical unit number */
        long mode;              /* mode word */
```

FORTRAN Subroutine Declaration:

```
c                               ! close a KSAM file
         subroutine _kclose(lun, mode, status)
             integer*4 lun      ! logical unit number
             integer*4 mode     ! mode word
             integer*4 status   ! result of the operation
```

Pascal Procedure Declaration:

```
procedure _kclose(              {** close a KSAM file}
        lun     : longint;      {** logical unit number}
        mode    : longint;      {** mode word}
    var status  : longint       {** result of the operation}
); external;
```

Create a KSAM file.

Description:

This SVC creates new KSAM data and keys files and initializes the key files using information provided by the user process. The "current key" is set to zero, and the "current record pointer" is undefined (the current position pointer is just before the first record in the file) as defined by the zeroth key.

Upon successful completion of _kcreat, the KSAM file is opened and the logical unit number is returned. Use the logical unit number for all subsequent accesses to the file.

Unless the process has bypass privilege, it must have read and write privilege to the device to contain the files, execute privilege for all directories in the path leading to the files, and read and write privilege to the directories to contain the files for the file to be successfully created.

NOTE: Each key may be up to 255 bytes long. Word and longword keys and key segments must lie on word boundaries (even byte) within memory and within the data record. Word keys and key segments must be two-byte multiples, and longword keys and key segments must be four-byte multiples. Assigning either a byte value in a record definition may misalign word or longword key fields that follow. You may have to offset the other keys to align them on word or longword boundaries.

Related Privileges:

none	- Allows creation if the process has access as described above.
altuic	- Allows creation if the owner of the image file for the current process has access as described above.
bypass	- Allows the process to create the file independent of the file protection.
system	- Allows creation if the system has access as described above.

Parameters:

fname - Address of a null terminated string containing the name of the KSAM data file to be created. It may be fully qualified with device, directory, file extension and version number qualifications. An extension of .DAT is recommended. This string will be translated automatically by WMCS to its logical equivalent. This string may contain up to 93 significant characters followed by a null.

kfname - Address of a null terminated string containing the name of the KSAM key file to be created. It may be fully qualified with device, directory, file extension and version number qualifications. An extension of .KEY is recommended. This string will be translated automatically by WMCS to its logical equivalent. This string may contain up to 93 significant characters followed by a null.

mode - A bit mask that specifies the type of access allowed to this and other users during the time the KSAM files pair is open. The following bits, when set, have the following meanings:

Bit Name	Bit #	Description
opreadacc	0	Read access - Requests permission to read the file.
opwriteacc	1	Write access - Requests permission to write the file.
opreadlock	2	Read lock - Requests permission for exclusive read access to the file. Other processes may not read the file(s).
opwritelock	3	Write lock - Requests permission for exclusive write access to the file. Other processes may not write the file(s).
opdelete	4	Delete - Requests that the files be deleted upon closing.
	5	Reserved.

opfastread	6	Fast read - Specifies that the file will be read asynchronously. That is, that control returns to the user process before the data have actually been read. As records are read, they will be transferred directly into the process's logical address space bypassing the device cache. This bit is only valid for disk class devices. Other requirements are 1) Supports only requests for complete sectors only, 2) Process buffer must be on a word boundary, 3) Request cannot cross a 4 Kbyte page boundary. Use the _frdwait system call to determine when asynchronous reads are complete.
opnextfile	7	Open next file - On a tape device, specifies to open the "next" file without regard to the filename.
opnordahead	8	No read ahead - Specifies that read ahead is not to be done on the opened file.
opnotruncfile	9	No truncate - Specifies that when the file is closed the extra physical sectors allocated to the file are not to be released.
	10	Reserved.
	11	Reserved.
opzerodelete	12	Zero delete - Zero each sector of the file before deleting the file. This bit is only valid if the file is being deleted (via cldelete or some other way).
	13-31	Reserved. Must be set to zero.

reclen - Record length. A value that represents the length in bytes of each record in the KSAM data file. The record length must be in the range of 4 to 65534 bytes inclusive. The record size specified by the calling process is internally incremented by one to include a deletion flag byte.

prot - File protection mask. The least significant 16 bit word of this parameter is divided into 4 nibbles. Each nibble corresponds to a class of users. The bits within each nibble represent the type of access that class of user is granted for this file. If the bit is set (1), the access is granted.

From the least to the most significant nibble the user classes are:

```
Ownr - file owner
Grp  - processes with the same group ID as the owner
Pub  - all other processes in the system
Sys  - processes with SYSTEM privilege.
```

```
  Sys  Pub  Grp  Ownr
 |----|----|----|----|
 |DWRE|DWRE|DWRE|DWRE|
 |-------------------|
MSB                 LSB
```

From the least to the most significant bits within the nibbles, the access privileges are:

```
E - Execute access
R - Read access
W - Write access
D - Delete access
```

The value $FFFFFFFF (-1) is a reserved value that means that the user's default protection mask is to be used.

numbuf - A value that specifies the number of 1-Kbyte buffers to allocate for file manipulation. The value supplied is used as follows:
- If the number supplied is zero, the number of buffers allocated is four times the number of defined keys.
- If the number supplied is not zero, but is a multiple of four, it is used "as is."
- If the number supplied is not zero and is not divisible by four, the number of buffers allocated is the number specified, rounded up to the next multiple of four.

In general, at least four buffers per key should be available for each key defined in the key definition table (see below). Optimal throughput is achieved by allocating sufficient buffers that the top two levels of each B-tree can remain in the KSAM cache at all times. The number of buffers needed to contain the top two levels of any given B-tree is:

```
1 + (1006/(<key-length>+4))
```

where <key-length> is the length of the key in bytes rounded up to an even number.

ktable - Address of an array that describes the keys that will be used to organize the data file. This table must be word aligned. You may define as many keys as you want, each of which can contain up to 15 segments, subject to the limitation that the total length of the array may not exceed 3500 bytes. Typically, this allows you to define more than 300 keys.

The very first word in the array specifies how many keys you are defining.

> NOTE: When you are creating a file, enter the number of keys you want to define. When you later access this file, refer to the first of the keys as key 0. For example, if you place a value of 5 in the first word of the KTABLE array, specifying that you want to define 5 keys for this file, the keys will be designated key 0, key 1, key 2, key 3, and key 4.

The rest of the array contains the definitions of these keys. Thus, the array looks like:

```
---------------------------------------------
|          Number of key definitions         |
---------------------------------------------

---------------------------------------------
|                                            |
|            First key definition            |
|              (4 to 32 words)               |
|                                            |
---------------------------------------------

---------------------------------------------
|                                            |
|           Second key definition            |
|              (4 to 32 words)               |
|                                            |
---------------------------------------------
```

You must specify at least six pieces of information in the key table array for each key. These are:

- data type
- number of segments in the key
- whether duplicate values are allowed in the key
- the total length of the key in bytes
- the starting position of each segment of the key
- the length of each segment of the key

The length of the key definition is from 4 to 32 words, depending on the number of segments defined for the key. Each key definition is organized as follows:

Word 0 of key definition
This word contains the duplicate key flag bit, the data type, and the number of segments in this key.

```
-------------------------------------------------------
| 15|14|13|12|11|10|9|8| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
-------------------------------------------------------
|       Reserved       |Dup| Key type  | Number of seg |
-------------------------------------------------------
```

The field positions of these data are:

Bits 15-8	-Reserved for internal use by KSAM. See the description of the _kinfo SVC for details.
Bit 7	-Duplicate key flag. A value of zero means that duplicate key values are allowed; a value of one means no duplicates are allowed.
Bits 6-4	-Key type. The following are valid key types: 000 8 bit unsigned byte (character) 001 8 bit signed byte 010 16 bit unsigned integer 011 16 bit signed integer (integer) 100 32 bit unsigned long integer 101 32 bit signed long integer (long int.) 110 reserved for future use by WICAT 111 reserved for future use by WICAT
Bits 3-0	-Number of segments in the key. This value must be between 1 and 15 (inclusive).

Word 1 of key definition
This word contains the total length of the key in bytes. Valid values are from 1 to 255 (inclusive).

Word 2 of key definition
This word contains the starting position within the record of the first segment of the key. The first byte of the record is designated byte zero.

Word 3 of key definition
This word contains the length in bytes of the first segment of the key. The length is subject to the following restrictions:

- No key or key segment (of any type) may exceed 255 bytes in length.
- Integer key and key segment lengths must be multiples of 2.
- Long integer key and key segment lengths must be multiples of 4.
- Character key and key segment lengths may be any value from 1 to 255 characters (inclusive).

Words 4 and 5 of key definition
These words (if present) are of the same format as words 2 and 3 but contain information concerning segment 2 of the key.

Words 6 and 7 of key definition
These words (if present) are of the same format as words 2 and 3 but contain information concerning segment 3 of the key.

.
.
.

Words 30 and 31 of key definition
These words (if present) are of the same format as words 2 and 3 but contain information concerning segment 15 of the key.

Example of key definition:
If a KSAM file is defined as having two keys, the first a long word key with 1 segment and the second a character key with 4 segments, the key table array may look like this:

Position	Value	Meaning
1	$2	Number of keys to follow
		Key 0 Definition
2	$51	Duplicate keys allowed, long word key, 1 segment
3	$4	Total length of key 0 in bytes
4	$0	Starting position of the key within the record
5	$4	The length of the segment is 4 bytes
		Key 1 Definition
6	$84	No duplicate keys allowed, character key, 4 segments
7	$2A	Total length of key 1 in bytes
8	$21	Starting position of the first segment of the key within the record
9	$10	Length of first segment of key
10	$4	Starting position of the second segment of the key within the record
11	$5	Length of second segment of key
12	$40	Starting position of the third segment of the key within the record
13	$11	Length of third segment of key
14	$0	Starting position of the fourth segment of the key within the record
15	$4	Length of fourth segment of key

Note that different key definitions may be created from the same portion of the data record. In this example, bytes 0-3 of the record are used as the first segment of key 0 and the last segment of key 1.

lun	- Address of a long word to receive the logical unit number from _kcreat after successful creation of the file.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinvvernum	(129)	A file's version number cannot be greater than 65535.
errinvdevnam	(130)	The specified devicename is syntactically incorrect.
errundevnam	(131)	The MCS does not recognize the devicename. Is the device mounted?

errfilexists	(134)	The specified version of the file already exists.
errinvreclen	(138)	Edit mode 3 requires that the file's record length be set to one.
errnoexecpriv	(143)	The process does not have Execute Privilege for the file.
errnoreadpriv	(144)	The process does not have Read Privilege for the file.
errnowritepriv	(145)	The process does not have Write Privilege for the file.
errinvfnstr	(147)	The specified filename is syntactically incorrect.
errinvdirfle	(148)	The specified directory is not a directory.
errinvdirstr	(149)	The specified directory name is syntactically incorrect.
errinvcloper	(173)	The device class is inappropriate for the operation.
errnoclass	(185)	The device class handler was not loaded when the system was booted.
errkeynotinrec	(221)	One or more of the KSAM keys is not contained in the record.
errkeytablelen	(222)	The KSAM key definition table is larger than 3500 bytes.
errnumofkeys	(225)	The specified number of keys is less than or equal to zero.
errnumofsegs	(226)	The specified number of segments is less than or equal to zero.
errrecsz	(227)	The record size is less than 4) bytes or greater than 65534) bytes.
errsegalign	(228)	A KSAM key for a word or longword key type is not word aligned.
errseglen	(229)	The specified key length is not a multiple of the key-type length.
errkeynotfnd	(230)	Key number is greater than or equal to the number of defined keys.

See Also:

_delete	- Delete a file
_kclose	- Close a KSAM file
_kopen	- Open a KSAM file

Assembler Calling Sequence:

```
push    fname                   ;address - data file name
push    kfname                  ;address - key file name
push    mode                    ;value - mode word
push    reclen                  ;value - record length
push    prot                    ;value - protection mask
push    numbuf                  ;value - number of buffers
push    ktable                  ;address - key definition table
push    lun                     ;address - logical unit number
push    status                  ;address - result of the operation
jsr     _kcreat                 ;create a KSAM file
```

C Function Declaration:

```
                                /* create a KSAM file */
long                            /* returns result of the operation */
_kcreat (fname, kfname, mode, reclen, prot, numbuf, ktable, lun)
         char fname[94];        /* data file name */
         char kfname[94];       /* key file name */
         long mode;             /* mode word */
         long reclen;           /* record length */
         long prot;             /* protection mask */
         long numbuf;           /* number of buffers */
         short ktable[x];       /* where x is the size of the array */
                                /* key table */
         long *lun;             /* logical unit number */
```

FORTRAN Subroutine Declaration:

```
c                               ! create a KSAM file
         subroutine _kcreat (fname, kfname, mode, reclen, prot,
     &           numbuf, ktable, lun, status)
            character*94 fname  ! data file name
            character*94 kfname ! key file name
            integer*4 mode      ! mode word
            integer*4 reclen    ! record length
            integer*4 prot      ! protection mask
            integer*4 numbuf    ! number of buffers
            integer*2 ktable(x) ! where x is the size of the array
                                ! key table
            integer*4 lun       ! logical unit number
            integer*4 status    ! result of the operation
```

Pascal Procedure Declaration:

```
    procedure _kcreat(              {** create a KSAM file}
            fname   : string[93];   {** data file name}
            kfname  : string[93];   {** key file name}
            mode    : longint;      {** mode word}
            reclen  : longint;      {** record length}
            prot    : longint;      {** protection mask}
            numbuf  : longint;      {** number of buffers}
            ktable  : array[0..x]_of_integer; {** where x is the size
                                    {** of the array key table}
        var lun     : longint;      {** logical unit number}
        var status  : longint       {** result of the operation}
    ); external;
```

Delete a ksam record.

Description:

_kdelet removes the record pointed to by the "current record pointer" from the ksam data file and places a deleted record flag in the record.

Note the following:

- The current record must be defined by _kread or _kwrite.
- If the current record has been locked by this process, the record is automatically unlocked before it is deleted.
- After the timeout period expires, if the current record is still locked by another process, an error will result.
- If current record has become undefined because another process has deleted the record, an error will result.

After this call the current record is undefined.

To successfully delete a record, the process must have opened the file with write access.

Related Privileges:

None.

Parameters:

lun	- The logical unit number from _kopen or _kcreat.
timout	- Specifies how long to wait (in units of Ø.Ø1 second) before returning with a timeout error if the desired record is locked by another process. === NOTE === The process calling _kdelet should check for a timeout error and provide code to handle this condition.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errtimeout	(128)	A request was not completed within the specified time.
errinvlfn	(132)	The logical unit number does not correspond to an open file.
errnowriteacc	(142)	The process does not have write-access to the

		specified file.
errnoclass	(185)	The device class handler was not loaded when the system was booted.
errkeynotdef	(231)	This operation requires that the current key be defined.
errdeadlock	(234)	The specified record cannot be locked without causing a deadlock.
errreclocked	(235)	The specified record(s) are locked by another process.
errrecnotdef	(236)	This operation requires that the current record be defined.

See Also:

_kread - Read a ksam record
_kupdat - Update an existing ksam record
_kwrite - Write a new ksam record

Assembler Calling Sequence:

```
push    lun                     ;value - logical unit number
push    timout                  ;value - time out
push    status                  ;address - result of the operation
jsr     _kdelet                 ;delete a ksam record
```

C Function Declaration:

```
                                /* delete a ksam record */
long                            /* returns result of the operation */
_kdelet(lun, timout)
        long lun;               /* logical unit number */
        long timout;            /* time out */
```

FORTRAN Subroutine Declaration:

```
c                               ! delete a ksam record
        subroutine kdelet(lun, timout, status)
            integer*4 lun       ! logical unit number
            integer*4 timout    ! time out
            integer*4 status    ! result of the operation
```

Pascal Procedure Declaration:

```
procedure _kdelet(              {** delete a ksam record}
        lun     : longint;      {** logical unit number}
        timout  : longint;      {** time out}
    var status  : longint       {** result of the operation}
); external;
```

Locate a ksam record.

Description:

Finds a record given a key number and the value of a key to search for.

This procedure can change the current key and will leave the current record undefined because its associated pointer will be left pointing between two records.

The record following the pointer is either the record that was found, or, if the key does not exist, is the record that has a key that is alphabetically or numerically greater than the key specified to find the desired record.

Note that "key value not found" error indicates that no perfect key match was made, but the current record pointer is positioned before the record that is alphabetically or numerically greater than the search key.

When a partial key is used, either a normal status will be returned if the record is found that matches the partial key or a "key value not found" error will be returned if no exact match was found.

Related Privileges:

None.

Parameters:

lun	- The logical unit number from _kopen or _kcreat.
keynum	- The number of the key on which to search. The first key specified in the 'ktable' array supplied to _kcreat is designated key zero.
keybuf	- The address of a buffer containing the value of the key or partial key that is used to search the file. If the key specified is a word or long word key, the buffer must begin on an even byte address boundary.
buflen	- The length of the key or partial key in 'keybuf'. The key length is restricted as follows: - All search keys must be less than or equal to the length specified at the time the ksam file was created. - Search keys for character keys may be of any length less than or equal to the defined length

```
              - Integer search keys must be multiples of two
              - long word integer search keys must be multiples of four
   status  - Address of a long word to receive the result of
             the operation.
```

Diagnostics:

errinvlfn	(132)	The logical unit number does not correspond to an open file.
errnoclass	(185)	The device class handler was not loaded when the system was booted.
errseglen	(229)	The specified key length is not a multiple of the key-type length.
errkeynotfnd	(230)	Key number is greater than or equal to the number of defined keys.
errsrchnotfnd	(241)	An exact match for the specified key value was not found.

See Also:

_kmovfb - Position to front or back of file

Assember Calling Sequence:

```
push    lun                         ;value - logical unit number
push    keynum                      ;value - key number
push    keybuf                      ;address - key value
push    buflen                      ;value - length of the key
push    status                      ;address - result of the operation
jsr     _kfind                      ;locate a ksam record
```

C Function Declaration:

```
                                    /* locate a ksam record */
long                                /* returns result of the operation */
_kfind(lun, keynum, keybuf, buflen)
        long lun;                   /* logical unit number */
        long keynum;                /* key number */
        char *keybuf;               /* key value */
        long buflen;                /* length of the key */
```

Fortran Subroutine Declaration:

```
c                                   ! locate a ksam record
        subroutine kfind(lun, keynum, keybuf, buflen, status)
            integer*4 lun           ! logical unit number
            integer*4 keynum        ! key number
            character*(*) keybuf    ! key value
            integer*4 buflen        ! length of the key
            integer*4 status        ! result of the operation
```

Pascal Procedure Declaration:

```
procedure _kfind(                      {** locate a ksam record}
        lun     : longint;             {** logical unit number}
        keynum  : longint;             {** key number}
        keybuf  : ^array_of_char;      {** key value}
        buflen  : longint;             {** length of the key}
    var status  : longint              {** result of the operation}
); external;
```

Write modified ksam buffers.

Description:

Flushes (writes) all currently unused ksam buffers to the file management system. _kflush is also executed by a call to the regular file system SVC _flush.

Related Privileges:

None.

Parameters:

siteid	- A long word containing the system id number of the system whose ksam buffers are to be flushed. A siteid of zero correponds to the system on which the calling process is executing.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinvsiteid	(8)	The specified site id does not exist.
errnoclass	(185)	The device class handler was not loaded when the system was booted.

See Also:

_flush - Flush I/O buffers to the device

Assembler Calling Sequence:

```
push    siteid                      ;value - system id
push    status                      ;address - result of the operation
jsr     _kflush                     ;write modified ksam buffers
```

C Function Declaration:

```
                                    /* write modified ksam buffers */
long                                /* returns result of the operation */
_kflush(siteid)
        long    siteid;             /* system id */
```

Fortran Subroutine Declaration:

```
c                                   ! write modified ksam buffers
        subroutine kflush(siteid, status)
            integer*4 siteid        ! system id
```

```
                 integer*4 status        ! result of the operation
```

Pascal Procedure Declaration:

```
    procedure _kflush (                  {** write modified ksam buffers}
            siteid : longint;            {** system id}
        var status : longint             {** result of the operation }
    ); external;
```

Retrieve ksam information file.

Description:

Provides the user program with information about a ksam file. It allows a program to work with a ksam file without knowing its attributes when it opens the file. For example, a process accessing a file it did not create can use _kinfo to obtain the information it needs to use the file.

_kinfo returns information about the keys in the keys file, the data records in the data file and identifies the last operation performed on the file.

Related Privileges:

None.

Parameters:

lun	- The logical unit number from _kopen or _kcreat.
option	- A value that describes what information is desired. The values that can be passed to the routine through this parameter and what information they will cause the routine to return are described below.

Negative Integer - Any negative integer for this parameter causes _kinfo to copy information into a 28-byte block of memory starting at the address specified by 'ktable'. The format of this block is:

Long word 0 - Contains the size of the data record as defined in the call to _kcreat.

Long word 1 - Contains the number of active data records in the KSAM data file.

Long word 2 - Contains the number of inactive data records in the KSAM data file.

Long word 3 - Contains the number of active key blocks in the KSAM key file.

Long word 4 - Contains the number of inactive key blocks in the KSAM key file.

Long word 5 - Contains the number of keys defined in the file.

Long word 6 - Contains a function code which represents the last successfully completed KSAM function call. The function code can assume the following values:

0	_kcreat
1	_kopen
2	_kread
3	_kwrite
4	_kupdat
5	_kdelet
6	_kfind
7	_kmovfb
8	_kinfo
9	_kunlck
10	_kflush

Key Number - A positive integer or zero for this value is interpreted as a key number. From 8 to 64 bytes of information (depending on the number of segments defined for this key) are copied into memory beginning at ktable. This information is identical to the information in the key table array passed to _kcreat for the specified key except for the high order byte of word 0, which contains the number of levels currently in use in the B-tree for this key. The format of this array is:

Word 0 - Contains the number of levels currently in use in the B-tree, the duplicate key flag bit, the data type, and the number of segments in this key.

```
---------------------------------------------------------------
|15 |14 |13 |12 |11 |10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
---------------------------------------------------------------
|        Number of levels        |Dup|  Key type | Number of seg |
---------------------------------------------------------------
```

The field positions of these data are:

Bits 15-8 - Contains the number of levels currently in use in the B-tree for this key.

Bit 7 - Duplicate key flag. A value of zero means that duplicate key values are allowed; a value of one means no duplicates are allowed.

Bits 6-4 - Key type. The following are valid key types:
000 - 8-bit unsigned byte (character)
001 - 8-bit signed byte
010 - 16-bit unsigned integer

011 - 16-bit signed integer (integer)
100 - 32-bit unsigned long integer
101 - 32-bit signed long integer (long integer)
110 - reserved for future use by WICAT
111 - reserved for future use by WICAT

Bits 3-0 - Number of segments in the key. This value must be between 1 and 15 (inclusive).

Word 1 - Contains the total length of the key in bytes. Valid values are from 1 to 255 (inclusive).

Word 2 - Contains the starting position within the record of the first segment of the key. The first byte of the record is designated byte zero (0).

Word 3 - Contains the length in bytes of the first segment of the key. The length is subject to the following restrictions:

- No key or key segment (of any kind) may exceed 255 bytes.
- Integer key and key segment lengths must be multiples of two.
- Long integer key and key segment lengths must be multiples of four.
- Character key and key segment lengths may be any value from 1 to 255 characters (inclusive).

Words 4 and 5 - These words (if present) are of the same format as words 2 and 3 but contain information concerning segment two of the key.

Words 6 and 7 - These words (if present) are of the same format as words 2 and 3 but contain information concerning segment three of the key.

.
.
.

Words 30 and 31 - These words (if present) are of the same format as words 2 and 3 but contain information concerning segment 15 of the key.

ktable - Address at which the information returned by this call will be placed. Twenty-eight bytes are required if option is a negative number, and from 8 to 64 bytes are required to copy the key table for a specified key.

status - Address of a long word to receive the result of the operation.

Diagnostics:

errinvlfn	(132)	The logical unit number does not correspond to an open file.
errnoclass	(185)	The device class handler was not loaded when the system was booted.
errkeynotfnd	(230)	Key number is greater than or equal to the number of defined keys.

See Also:

_kcreat - Create a ksam file

Assember Calling Sequence:

```
push    lun                       ;value - logical unit number
push    option                    ;value - function code
push    ktable                    ;address - returned data
push    status                    ;address - result of the operation
jsr     _kinfo                    ;retrieve ksam file information
```

C Function Declaration:

```
                                  /* retrieve ksam file information */
long                              /* returns result of the operation */
_kinfo(lun, option, ktable)
        long lun;                 /* logical unit number
        long option;              /* function code */
        char *ktable;             /* returned data */
```

Fortran Subroutine Declaration:

```
c                                 ! retrieve ksam file information
        subroutine kinfo(lun, option, ktable, status)
            integer*4 lun         ! logical unit number
            integer*4 option      ! function code
            character*(*) ktable  ! returned data
            integer*4 status      ! result of the operation
```

Pascal Procedure Declaration:

```
procedure _kinfo(                 {** retrieve ksam file information}
        lun     : longint;        {** logical unit number}
        option  : longint;        {** function code}
        ktable  : ^array_of_char; {** returned data}
    var status  : longint         {** result of the operation}
); external;
```

Position to front or back of file.

Description:

Allows the user program to position the current record pointer to just after the last record or just before the first record in the file for the specified key. After a call to _kmovfb, the current record is undefined.

- If a forward read is executed immediately after a move to back of file request is performed, an end-of-file condition would be encountered.
- The same is true if a reverse read is executed immediately after a move to the front request is performed.
- A "read current record" performed after either of these calls would result in a "Current record is undefined" error.

_kmovfb makes it easy for a program to read a KSAM file sequentially in either direction.

_kmovfb sets the "current key" to the key number specified in the call.

Related Privileges:

None.

Parameters:

lun	- The logical unit number from _kopen or _kcreat.
keynum	- The number of the key for which the current record pointer is positioned to the beginning-of-file or end-of-file. The first key defined is key 0.
mode	- Specifies the direction of the move. Zero means move to the beginning of the file. Non-zero means move to the end of the file.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinvlfn	(132)	The logical unit number does not correspond to an open file.
errnoclass	(185)	The device class handler was not loaded when the system was booted.
errkeynotfnd	(230)	Key number is greater than or equal to the number of defined keys.

See Also:

_kfind	- Locate a ksam record
_kread	- Read a ksam record

Assembler Calling Sequence:

```
push    lun                          ;value - logical unit number
push    keynum                       ;value - key number
push    mode                         ;value - mode word
push    status                       ;address - result of the operation
jsr     _kmovfb                      ;position to front or back of file
```

C Function Declaration:

```
                                     /* position to front or back of file */
long                                 /* returns result of the operation */
_kmovfb(lun, keynum, mode)
        long lun;                    /* logical unit number */
        long keynum;                 /* key number */
        long mode;                   /* mode word */
```

Fortran Subroutine Declaration:

```
c                                    ! position to front or back of fil
        subroutine kmovfb(lun, keynum, mode, status)
            integer*4 lun            ! logical unit number
            integer*4 keynum         ! key number
            integer*4 mode           ! mode word
            integer*4 status         ! result of the operation
```

Pascal Procedure Declaration:

```
procedure _kmovfb(                   {** position to front or back of file}
        lun      : longint;          {** logical unit number}
        keynum   : longint;          {** key number}
        mode     : longint;          {** mode}
    var status   : longint           {** result of the operation}
); external;
```

Open a KSAM file.

Description:

_KOPEN opens one KSAM file (consisting of a data and a key file) with the given names and positions the file at the beginning-of-file for the first key (key 0). The current key is set to zero, and the the current record pointer is undefined (the current position pointer is just in front of the first record in the file) as defined by the zeroth key.

Unless the process has bypass privilege, it must have read/write privilege to the device(s) containing the files, execute privilege to all directories in the path leading to the files, read privilege to the directory containing the files, and read/write privilege to the files themselves in order for the files to be successfully opened.

If fname (or kfname) is specified in fcb.seq number format, the process must have read/write privilege to the device(s) containing the files, and read/write privilege to the files themselves in order for the files to be successfully opened.

Related Privileges:

none	- Allows opening if the process has access to the files as described above.
altuic	- Allows opening if the owner of the image file for the current process has access to the files as described above.
bypass	- Allows opening independent of file protection.
system	- Allows opening if the system has access to the files as described above.

Parameters:

fname	- Address of a null terminated string containing the name of the KSAM data file to be opened. This string will be translated automatically by the MCS into its logical equivalence. This string may contain up to 93 valid characters followed by a null.

kfname - Address of a null terminated string containing the name of the KSAM key file to be opened. This string will be translated automatically by the MCS into its logical equivalent. This string may contain up to 93 valid characters followed by a null.

mode - A bit encoded mask that specifies the access allowed to this and other users during the time the KSAM files pair is open. The following bits, when set, mean the following:

Bit Name	Bit	Description
opreadacc	0	Read - Current process is allowed read access
opwriteacc	1	Write - Current process is allowed write access
opreadlock	2	Read lock - Other processes may not read the file(s)
opwritelock	3	Write lock - Other processes may not write the file(s)
opdelete	4	Delete the file when closed
	5	Reserved.
opfastread	6	Fast read - Specifies that the file will be read asynchronously. That is, that control returns to the user process before the data have actually been read. As records are read, they will be transferred directly into the process's logical address space bypassing the device cache. This bit is only valid for disk class devices. Other requirements are 1) Supports only requests for complete sectors only, 2) Process buffer must be on a word boundary, 3) Request cannot cross a 4 Kbyte page boundary. Use the _frdwait system call to determine when asynchronous reads are complete.
opnextfile	7	Open next file - On a tape device, specifies to open the "next" file without regard to the filename.
opnordahead	8	No read ahead - Specifies that read ahead is not to be done on the opened file.
opnotruncfile	9	No truncate - Specifies that when the file is closed the extra physical sectors allocated to the file are not to be released.

	10	Reserved. Must be set to zero.
	11	Reserved. Must be set to zero.
opzerodelete	12	Zero delete - Zero each sector of the file before deleting the file. This bit is only valid if the file is being deleted (via cldelete or some other way).
	13-31	Reserved. Must be set to zero.

numbuf - A value that specifies the number of 1K buffers to allocate for file manipulation. The value supplied is used as follows:

- If the number supplied is zero, the number of buffers allocated is four times the number of defined keys.
- If the number supplied is not zero, but is a multiple of four, it is used "as is".
- If the number supplied is not zero and is not divisible by four, the number of buffers allocated is the number specified rounded up to the next multiple of four.

In general, at least four buffers per key should be available for each key defined in the key definition table (see below). Optimal throughput is achieved by allocating sufficient buffers that the top two levels of each B-tree can remain in the KSAM cache at all times. The number of buffers needed to contain the top two levels of any given B-tree is:

```
1 + (1006/(<key-length>+4))
```

where <key-length> is the length of the key in bytes, rounded up to an even number.

lun - Address of the variable that will contain the logical unit number of the successfully opened file.

status - Address of a long word to receive the result of the operation.

Diagnostics:

errnomemavail	(7)	All available memory has been allocated.
errinvdevnam	(130)	The specified devicename is syntactically incorrect.
errundevnam	(131)	The MCS does not recognize the devicename. Is the device mounted?
errfilnotfnd	(133)	The specified file could not be found.
errnoexecpriv	(143)	The process does not have Execute Privilege for the file.

errnoreadpriv	(144)	The process does not have Read Privilege for the file.
errnowritepriv	(145)	The process does not have Write Privilege for the file.
errinvfnstr	(147)	The specified filename is syntactically incorrect.
errinvdirfle	(148)	The specified directory is not a directory.
errinvdirstr	(149)	The specified directory name is syntactically incorrect.
errinvcloper	(173)	The device class is inappropriate for the operation.
errdirnotfnd	(177)	The specified directory does not exist.
errinvseqnum	(178)	The file's FCB.SEQ number in the directory file is incorrect.
errnoclass	(185)	The device class handler was not loaded when the system was booted.

See Also:

_kclall - Close all KSAM files
_kclose - Close a KSAM file
_kcreat - Create a KSAM file

Assembler Calling Sequence:

```
push    fname                   ;address - data file name
push    kfname                  ;address - key file name
push    mode                    ;value - mode word
push    numbuf                  ;value - number of buffers
push    lun                     ;address - logical unit number
push    status                  ;address - result of the operation
jsr     _kopen                  ;open a KSAM file
```

C Function Declaration:

```
                                /* open a KSAM file */
long                            /* returns result of the operation */
_kopen(fname, kfname, mode, numbuf, lun)
         char fname[94];        /* data file name */
         char kfname[94];       /* key file name */
         long mode;             /* mode word */
         long numbuf;           /* number of buffers */
         long *lun;             /* logical unit number */
```

FORTRAN Subroutine Declaration:

```
    c                                   ! open a KSAM file
            subroutine _kopen(fname, kfname, mode, numbuf, lun, status)
                character*94 fname ! data file name
                character*94 kfname ! key file name
                integer*4 mode     ! mode word
                integer*4 numbuf   ! number of buffers
                integer*4 lun      ! logical unit number
                integer*4 status   ! result of the operation
```

Pascal Procedure Declaration:

```
    procedure _kopen(                   {** open a KSAM file}
            fname   : string[93];       {** data file name}
            kfname  : string[93];       {** key file name}
            mode    : longint;          {** mode}
            numbuf  : longint;          {** number of buffers}
       var  lun     : longint;          {** logical unit number}
       var  status  : longint           {** result of the operation}
    ); external;
```

Read a ksam record.

Description:

_kread reads the next, current, or previous (as defined by the "current key" and "current record") record from the KSAM file into the user process's buffer. If the read is successful, the record that is read becomes the "current record".

This routine reads the file sequentially forwards/backwards, in ascending/descending order, alphabetically, or numerically.

If the "lock bit" (see below under OPTION) is set, the specified record will be write locked before _kread reads the record into the buffer. If the record has been previously locked by another process, _kread waits until the TIMOUT period expires before returning with a timeout error. (If the file becomes unlocked before the TIMOUT period expires, the read continues normally and no error occurs.) The data transfer inhibit bit allows the file pointers to be moved without actually transferring data.

Deadlock detection is performed on the record to be read. If no deadlock is detected but the record is locked, the process will wait until the record can be successfully locked or until the TIMOUT period has expired.

After the record is read, the key value found in the key file is typically compared to the key found within the data just transferred. Use the "key compare inhibit" bit in OPTION to inhibit this comparison.

Related Privileges:

None.

Parameters:

```
lun    - The logical unit number from _kopen or _kcreat.
option - A bitmask that specifies options to be used
         for this call to _kread. The bit fields are:
         Bits Ø-3 - Specifies which record to read, as follows:

           ØØØØ - Read current record ("current record" must
                  be defined)
           ØØØ1 - Read next (ascending) record
           ØØ1Ø - Read previous (descending) record
           ØØ11 to 1111 - Reserved
```

Bit 4 - Lock Request Bit. If this bit equals one, write lock the record before reading. If the record is already locked, the lock request will be queued until it can be granted or until the TIMOUT period expires.

Bit 5 - Key Compare Inhibit. If bit five is zero, the key from the key file is compared to the key constructed from the data just read. If the two disagree, an error is returned. A key compare error indicates that the key file now disagrees with the data file and that the key file should be rebuilt. If this bit is one, no keys are compared. KSAM ignores this bit when the data transfer inhibit bit is set.

Bit 6 - Data Transfer Inhibit Bit. When Ø, data transfers are done. If set to 1, pointers are updated, but no actual data transfer occurs. KSAM ignores the compare inhibit bit if this bit is set.

Bits 7 to 31 Reserved (Must be set to zeros (Ø)).

timout - Specifies how long to wait for successful completion before returning with a timeout error if the desired record is locked by another process. timout is specified in Ø.Ø1 of a second.

=== NOTE ===
The process calling _kread should check for a timeout error and provide code to handle this condition.

buf - Address of a buffer into which the data record from the ksam file can be read. The buffer must be large enough to contain the entire record because the entire data record is transferred.

status - Address of a long word to receive the result of the operation.

Diagnostics:

errnomemavail	(7)	All available memory has been allocated.
errtimeout	(128)	A request was not completed within the specified time.
errinvlfn	(132)	The logical unit number does not correspond to an open file.
errreadleof	(14Ø)	The process tried to read past the logical end

		of a file.
errnoreadacc	(141)	The process does not have read-access to the file.
errnoclass	(185)	The device class handler was not loaded when the system was booted.
errkeynotdef	(231)	This operation requires that the current key be defined.
errdeadlock	(234)	The specified record cannot be locked without causing a deadlock.
errreclocked	(235)	The specified record(s) are locked by another process.
errrecnotdef	(236)	This operation requires that the current record be defined.

See Also:

_kdelet - Delete a ksam record
_kfind - Locate a ksam record
_kmovfb - Position to front or back of file
_kunlck - Unlock specified ksam records
_kupdat - Update an existing ksam record
_kwrite - Write a new ksam record

Assembler calling sequence:

```
push    lun                     ;value - logical unit number
push    option                  ;value - mode word
push    timout                  ;value - time out
push    buf                     ;address - read buffer
push    status                  ;address - result of the operation
jsr     _kread                  ;read a ksam record
```

C Function Declaration:

```
                                /* read a ksam record */
long                            /* returns result of the operation */
_kread(lun, option, timout, buf)
        long lun;               /* logical unit number */
        long option;            /* mode word */
        long timout;            /* time out */
        char *buf;              /* read buffer */
```

Fortran Subroutine Declaration:

```
c                               ! read a ksam record
        subroutine kread(lun, option, timout, buf, status)
            integer*4 lun       ! logical unit number
            integer*4 option    ! mode word
            integer*4 timout    ! time out
```

```
                    character*(*) buf      ! read buffer
                    integer*4 status       ! result of the operation
```

Pascal Procedure Declaration:

```
procedure _kread(                        {** read a ksam record}
        lun     : longint;               {** logical unit number}
        option  : longint;               {** mode}
        timout  : longint;               {** time out}
        buf     : ^array_of_char;        {** read buffer}
    var status  : longint                {** result of the operation}
); external;
```

Unlock specified ksam records.

Description:

Unlocks the current record or unlocks all records locked on the lun specified. If the records specified are already unlocked, nothing happens, and no error is returned.

Related Privileges:

None.

Parameters:

lun	- The logical unit number from _kopen or _kcreat.
option	- Specifies the action to be taken. Only bit 0 is currently used. If bit 0 is zero, the current record is unlocked. If bit 0 is one, all records locked by the given LUN are unlocked.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinvlfn	(132)	The logical unit number does not correspond to an open file.
errnoclass	(185)	The device class handler was not loaded when the system was booted.
errkeynotdef	(231)	This operation requires that the current key be defined.
errksamnorec	(237)	The specified record(s) is not locked.

See Also:

_kread	- Read a ksam record
_kupdat	- Update an existing ksam record
_kwrite	- Write a new ksam record

Assembler Calling Sequence:

```
push    lun                           ;value - logical unit number
push    option                        ;value - option
push    status                        ;address - result of the operation
jsr     _kunlck                       ;Unlock specified ksam records
```

C function Declaration:

```
                                      /* unlock specified ksam records */
long                                  /* returns result of the operation */
```

```
    _kunlck(lun, option)
            long lun;                   /* logical unit number */
            long option;                /* option */
```

Fortran Subroutine Declaration:

```
    c                                   ! unlock specified ksam records
            subroutine kunlck(lun, option, status)
                integer*4 lun           ! logical unit number
                integer*4 option        ! option
                integer*4 status        ! result of the operation
```

Pascal Procedure Declaration:

```
    procedure _kunlck(                  {** unlock specified ksam records}
            lun     : longint;          {** logical unit number}
            option  : longint;          {** option}
        var status  : longint           {** result of the operation}
    ); external;
```

Update an existing ksam record.

Description:

Updates the record that is pointed to by "current record". It allows a program to change a record that has already been written to the ksam file. If the value of any of the keys is changed, the key in the keys file is changed to reflect the new key value.

Any key may be updated, however, if a key is defined as disallowing duplicate values and the value of the key has changed, the new value is checked to see if it is already in the file. If it is, then the record is not updated, and an error results.

Records that are locked by another process (or by the same process under another LUN) may not be updated until they are unlocked. If the record is locked, the SVC will wait up to the value supplied in timout before returning with a timeout error.

Related Privileges:

None.

Parameters:

```
lun     - The logical unit number from _kopen or _kcreat.
timout  - Specifies how long to wait (in Ø.Øl of a second)
          before returning with a timeout error if the
          desired record is locked by another process.

                      === NOTE ===
                The process calling _kupdat should check
                for a timeout error and provide code to
                handle this condition.

buf     - The address of a buffer from which the data
          record is written to the KSAM file.  The buffer
          must be large enough to contain the entire record
          because the entire data record is transferred.
status  - Address of a long word to receive the result of
          the operation.
```

Diagnostics:

errnomemavail	(7)	All available memory has been allocated.
errtimeout	(128)	A request was not completed within the specified time.

errinvlfn	(132)	The logical unit number does not correspond to an open file.
errnowriteacc	(142)	The process does not have write-access to the specified file.
errnospace	(154)	All available disk space has been allocated.
errinvcloper	(173)	The device class is inappropriate for the operation.
errnoclass	(185)	The device class handler was not loaded when the system was booted.
errkeynotdef	(231)	This operation requires that the current key be defined.
errnodupkey	(232)	Duplicate key was attempted in a field disallowing duplicate keys.
errdeadlock	(234)	The specified record cannot be locked without causing a deadlock.
errreclocked	(235)	The specified record(s) are locked by another process.
errrecnotdef	(236)	This operation requires that the current record be defined. Device integrity error

See Also:

_kdelet - Delete a ksam record
_kread - Read a ksam record
_kunlck - Unlock specified ksam records
_kwrite - Write a new ksam record

Assembler Calling Sequence:

```
push    lun                     ;value - logical unit number
push    timout                  ;value - time out
push    buf                     ;address - record to update
push    status                  ;address - result of the operation
jsr     _kupdat                 ;update an existing ksam record
```

C Function Declaration:

```
                                /* update an existing ksam record */
long                            /* returns result of the operation */
_kupdat(lun, timout, buf)
        long lun;               /* logical unit number */
        long timout;            /* time out */
        char *buf;              /* record to update */
```

Fortran Subroutine Declaration:

```
c                              ! update an existing ksam record
        subroutine kupdat(lun, timout, buf, status)
```

```
                    integer*4 lun            ! logical unit number
                    integer*4 timout         ! time out
                    character *(*) buf       ! record to update
                    integer*4 status         ! result of the operation
```

Pascal Procedure Declaration:

```
procedure _kupdat(                {** update an existing ksam record}
        lun     : longint;        {** logical unit number}
        timout  : longint;        {** time out}
        buf     : ^array_of_char; {** record to update}
    var status  : longint         {** result of the operation}
); external;
```

Write a new ksam record.

Description:

Writes a record to the ksam file. Upon successful completion, the record becomes the current record. If a key is defined as disallowing duplicate values, KWRITE checks to see if the key values are already in the file. If so, then the record is not written, and an error results.

Related Privileges:

None.

Parameters:

lun	- The logical unit number from _kopen or _kcreat.
timout	- Specifies how long to wait (in Ø.Ø1 of a second) before returning with a timeout error if the write is unsuccessful. === Note === The process calling _kwrite should check for a timeout error and provide code to handle this condition.
buf	- Contains the address of a buffer from which the data record is written to the KSAM file.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errnomemavail	(7)	All available memory has been allocated.
errtimeout	(128)	A request was not completed within the specified time.
errinvlfn	(132)	The logical unit number does not correspond to an open file.
errnowriteacc	(142)	The process does not have write-access to the specified file.
errnospace	(154)	All available disk space has been allocated.
errinvcloper	(173)	The device class is inappropriate for the operation.
errnoclass	(185)	The device class handler was not loaded when the system was booted.
errnodupkey	(232)	Duplicate key was attempted in a field disallowing duplicate keys.
errdeadlock	(234)	The specified record cannot be locked without

		causing a deadlock.
errreclocked	(235)	The specified record(s) are locked by another process.
errrecnotdef	(236)	This operation requires that the current record be defined.
		Device integrity error

See Also:

_kdelet - Delete a ksam record
_kread - Read a ksam record
_kunlck - Unlock specified ksam records
_kwrite - Write a new ksam record

Assember Calling Sequence:

```
push    lun                          ;value - logical unit number
push    timout                       ;value - time out
push    buf                          ;address - record to be written
push    status                       ;address - result of the operation
jsr     _kwrite                      ;write a new ksam record
```

C Function Declaration:

```
                                     /* write a new ksam record */
long                                 /* returns result of the operation */
_kwrite(lun, timout, buf)
        long lun;                    /* logical unit number */
        long timout;                 /* time out */
        char *buf;                   /* record to be written */
```

Fortran Subroutine Declaration:

```
c                                    ! write a new ksam record
        subroutine kwrite(lun, timout, buf, status)
            integer*4 lun            ! logical unit number
            integer*4 timout         ! time out
            character*(*) buf        ! record to be written
            integer*4 status         ! result of the operation
```

Pascal Procedure Declaration:

```
procedure _kwrite(                   {** write a new ksam record}
        lun      : longint;          {** logical unit number}
        timout   : longint;          {** time out}
        buf      : ^array_of_char;   {** record to be written}
    var status   : longint           {** result of the operation}
); external;
```

Lock records within an open file.

Description:

_lock is a mechanism which will allow multiple processes to successfully have read and/or write access to the same file without interfering with one another. It provides controlled access to specified records within an open file. When a process locks one or more records, those records are not accessible to other processes in the system. Other processes which attempt to lock, read or write the locked area will be suspended with an I/O wait until the area is unlocked, or until the timout is exceeded. Deadlocks are detected and if found, control is returned to the calling process immediately.

The process can lock a group of records and then later unlock specific records within the group. A process can lock records that are beyond the logical end of file.

All types of files can be locked (including user defined file types, and system files). Note, however, that the Operating system does not check for locked records when it is updating system file (bitmap, fcb, directories). Records may only be locked on disk class devices.

Note that named semaphores may be implemented by creating files which are only manipulated by locks. One file is capable of containing a large number of semaphores. Since a process can lock records beyond the logical end of file, the 'semaphore' file need not contain any data.

Related Privileges:

None.

Parameters:

lun	- The logical unit number of the open file containing the records to be locked.
recnum	- The record number of the first record to be locked. Record number 0 corresponds to the first record in the file. A record number of $FFFFFFFF (-1) corresponds to the current record. Records can be locked beyond the logical end of file.
nrecs	- The number of records to be locked. This value is an unsigned integer. A value of zero means to lock from the current position to the logical end of file.
timout	- The wait count is in 100'ths of a second and represents

the maximum amount of time to wait for the specified region to become available for locking.

status - The address of a long word to receive the result of the operation.

Diagnostics:

errnomemavail	(7)	All available memory has been allocated.
errtimeout	(128)	A request was not completed within the specified time.
errinvlfn	(132)	The logical unit number does not correspond to an open file.
errinvcloper	(173)	The device class is inappropriate for the operation.
errdeadlock	(234)	The specified record cannot be locked without causing a deadlock.
errreclocked	(235)	The specified record(s) are locked by another process.
errlockint	(254)	(MCS error) A discrepency in the Record Locking code has been detected.

See Also:

_read	- Read from an open file
_unlock	- Unlock records in an open file
_write	- Write to an open file

Assembler Calling Sequence:

```
push    lun                        ;value - logical unit number
push    recnum                     ;value - starting record number
push    nrecs                      ;value - number of records
push    timout                     ;value - time out
push    status                     ;address - result of the operation
jsr     _lock                      ;lock records within an open file
```

C function declaration:

```
                                   /* lock records within an open file */
long                               /* returns result of the operation */
_lock(lun,recnum,nrecs,timout)
        long lun;                  /* logical unit number */
        long recnum;               /* starting record number */
        long nrecs;                /* number of records */
        long timout;               /* time out */
```

Fortran Subroutine Declaration:

```
c                                  ! lock records within an open file
        subroutine lock(lun, recnum, nrecs, timout, status)
            integer*4 lun          ! logical unit number
```

```
                integer*4 recnum       ! starting record number
                integer*4 nrecs        ! number of records
                integer*4 timout       ! time out
                integer*4 status       ! result of the operation
```

Pascal Procedure Declaration:

```
    procedure _lock(                      {** lock records within an open file}
            lun     : longint;            {** logical unit number}
            recnum  : longint;            {** starting record number}
            nrecs   : longint;            {** number of records}
            timout  : longint;            {** time out}
        var status  : longint             {** result of the operation}
    ); external;
```

mapfp - Map floating point hardware

Description:

Map the physical address space of the specified type of hardware floating point into the calling process's logical address space.

Related Privileges:

None.

Parameters:

fptype	- A constant representing the type of hardware to be mapped into the process's logical space. Valid values are:

Name	Value	Description
fpunmap	0	unmap the given logical address
fpskyl	1	skyl
fpndp2	2	ndp2
fpffpl	3	ffpl

These are defined in the file sys$disk/sysincl.sys/sysequ.asm

adr	- The logical address into which the hardware will be mapped. Adr must be aligned on a 4 Kbyte boundary.
size	- The number of bytes to be mapped. That is, the size of the physical segment of memory to be mapped. This value will be rounded up to the hardware page size since only full pages can be mapped.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinvadr	(4)	The logical/physical address, for the memory requested, is invalid.
errmemalloc	(5)	The process requested a logical page that was already allocated.
errmemdeall	(9)	The process attempted to affect memory that does not exist.
errhavemath	(24)	The process has already allocated floating

```
                             point hardware.
    ermoclass         (185)  The device class handler was not loaded when
                             the system was booted.
    ermohardware      (312)  The PC board for the specified device is not
                             installed.
```

See Also:

```
    _mapphys         - Map physical address into logical address
    _fremem          - Free memory
```

Assembler Calling Sequence:

```
    push    fptype                  ; value - type of hardware
    push    adr                     ; value - logical address
    push    size                    ; value - length in bytes
    push    status                  ; address - result of the operation
    jsr     _mapfp                  ; map floating point hardware
```

C function declaration:

```
                                    /* map floating point hardware */
    long                            /* returns result of the operation */
    _mapfp ( fptype, adr, size)
            long fptype;            /* type of math hardware */
            long adr;               /* logical address to map into */
            long size;              /* length of bytes to map */
```

Fortran Subroutine Declaration:

```
    c                               ! map floating point hardware
            subroutine mapfp(fptype, adr, size, status)
                integer*4 fptype    ! type of math hardware
                integer*4 adr       ! logical address to map into
                integer*4 size      ! length of bytes to map
                integer*4 status    ! result of the operation
```

Pascal Procedure Declaration:

```
    procedure  mapfp(               {** map floating point hardware  }
            fptype  : longint;      {** type of math hardware        }
            adr     : longint;      {** logical address to map into  }
            size    : longint;      {** length of bytes to map       }
        var status  : longint       {** result of the operation}
    ); external;
```

mapphys - Map physical address into process's logical space

Description:

Map the given physical address into the process's logical space at the given address.

Related Privileges:

none	- The calling process cannot map physical memory into its logical space with this system call.
chngsuper	- Allows a process to map physical memory into its logical address space.

Parameters:

physad	- Physical address which is to be mapped into the process's address space. This address must be on a 4 Kbyte address boundary.
adr	- The logical address into which the physical address will be mapped. Adr must be aligned on a 4 Kbyte boundary.
size	- The number of bytes to be mapped. That is, the size of the physical segment of memory to be mapped. This value will be rounded up to the hardware page size since only full pages can be mapped.
prot	- Protection. 0 indicates that the page(s) are not to be protected. 1 indicates that the page should be write protected.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1) The process lacks the privileges required to perform the operation.
errinvadr	(4) The logical/physical address, for the memory requested, is invalid.
errmemalloc	(5) The process requested a logical page that was already allocated.

See Also:

_mapfp	- Map floating point hardware into logical space
_fremem	- Free memory

Assembler Calling Sequence:

```
push    physad                    ;value - physical memory address
push    adr                       ;value - logical address
push    size                      ;value - length in bytes
push    prot                      ;value - protection code
push    status                    ;address - result of the operation
jsr     _mapphys                  ;map physical address
```

C function declaration:

```
                                  /* map physical address */
long                              /* returns result of the operation */
_mapphys (physad, adr, size, prot)
        long physad;              /* physical memory address */
        long adr;                 /* logical address */
        long size;                /* length in bytes */
        long prot;                /* protection code */
```

Fortran Subroutine Declaration:

```
c                                 ! map physical address
        subroutine mapphy (physad, adr, size, prot, status)
            integer*4 physad      ! physical memory address
            integer*4 adr         ! logical address
            integer*4 size        ! length in bytes
            integer*4 prot        ! protection code
            integer*4 status      ! result of the operation
```

Pascal Procedure Declaration:

```
procedure  mapphys(                {** map physical address        }
        physad  : longint;         {** physical memory address     }
        adr     : longint;         {** logical address             }
        size    : longint;         {** length in bytes             }
        prot    : longint;         {** protection code             }
    var status  : longint          {** result of the operation     }
); external;
```

Mount a logical device from memory.

Description:

This system call is similar to the _mount system call except that in this call, the device driver used to mount the device is obtained from a buffer in the process's memory (as opposed to a disk file). This system call should be used when the process does not have access to any device which contains the device driver.

This system call is used to announce the existence of a device to the system. The system mounts the device by loading a driver and initializing the device. If a device is already mounted with the specified driver, a new driver is not loaded, rather the current driver is shared.

For disk and tape class devices which are not mounted "special", the owner of the volume and the protection specification for each class of user is specified in the volume label.

For TTY, pipe and sync class devices, the owner of the device becomes the UIC of the process issuing the call to _memmnt. The protection mask for the device will be the default protection mask associated with the calling process.

For devices mounted "special", the owner of the device becomes the UIC of the process issuing the call to _memmnt. The protection mask for the device will be the default protection mask associated with the calling process.

The process issuing this system call must have operator privilege.

In addition, the process must have delete access to the device being mounted according to the owner and group ID (UIC) of the volume and its protection mask. Note that any process with operator privilege can mount a TTY, pipe or sync class device with this system call.

Related Privileges:

none	- The calling process cannot mount a device with this system call.
altuic	- If the calling process also has operator privilege, this privilege allows mounting of devices to which the owner of image file for this process has access as described above.
bypass	- If the calling process also has operator privilege, this privilege allows mounting of device independent of the device protection.
operator	- Allows mounting of devices to which the calling process has access as described above.
system	- If the calling process also has operator privilege, this privilege allows mounting of device if the system has access to the device as described above.

Parameters:

dname - Address of a null terminated string containing the name by which the device will be known. This string is translated automatically by the MCS to its logical equivalent. This string may contain up to 93 valid characters followed by a null.

driver - Address of a buffer in the user process memory that contains the device driver to be used to mount the device. If a driver with the same identifier is found in the system, the driver is not loaded.

class - The device class. Valid classes are:

0,1	- Character class device (TTYSpecial, TTY)
2,3	- Tape class device (TapeSpecial, Tape)
4,5	- Disk class device (DiskSpecial, Disk)
6,7	- Network class device (NetworkSpecial, Network)
8,9	- Pipe class device (PipeSpecial, Pipe)
10,11	- Sync class device (SyncSpecial, Sync)
12,13	- Queue class device (QueueSpecial, Queue)
14,15	- Nondev class device (NonDevSpecial, NonDev)

dstat - The address of a 128 byte buffer containing the initial device status to be assigned the device when it is mounted. If this parameter is zero the default device status is used.

This parameter has two purposes:
1) To provide an opportunity to set device characteristics that, unless set properly, would not allow the device to be mounted, e.g., the tape block size)

2) To set device characteristics that could otherwise not be changed once the device is mounted.

This parameter is not meant to be a substitute for _setdst. As such, not all of the values that can be specified with _setdst can be specified in this parameter.

The device status table is divided into two parts. The first half is device independent and is composed of the following fields:

Name	Length (bytes)	Settable	Description
dsclassid	2	No	The device class. Uses the class parameter to the _mount system call.
dsdriverid	2	No	Unique ID number for this device driver
dsblksz	2	Yes	block size of the device, e.g., sector size. For disks, the sector size is either 512 bytes or 1024 bytes, determined by the driver. Note that disk sector size cannot be changed. For tapes, the default is 1024 bytes. Specify zero to accept the default.
dsharderr	2	No	The hard error count for the device
dssofterr	2	No	The soft error count for the device
dsreadoper	4	No	Number of read operations on this device
dswriteoper	4	No	Number of write operations on this device
dsmaxnumdev	2	No	Maximum # of devices this driver can handle
dscurnumdev	2	No	Number of devices currently mounted using this device driver
dsnumtoretry	2	Yes	Number of times to retry before reporting a hard error. The default is determined by the driver. Specify zero (0) to accept the default.
dserrorreason	4	No	Hardware error code for the last error
dsreserved	30	No	Reserved
dsnexttableptr	4	No	Address of next device status table

The second half of the device status table is device class dependent. For TAPE class devices the second half is defined as follows:

Name	Length (bytes)	Settable	Description
dstpstatus	2	No	Tape device status
dstpflags1	2	Yes	Tape status information. Bit encoded. If zero is specified, the default is used.
			Bit name / bit # / Description
			dstpdoraw / 0 / 0=Read after write disabled 1=Read after write enabled
			dstpreadahead / 1 / 0=Read ahead enabled 1=Read ahead disabled
			dstperrintenb / 2 / 0=Error interrupts are enabled 1=Error interrupts are disabled
dstpspeed	1	Yes	Tape speed - Specify zero to use default
			0 - Reserved
			dstpspeed12ips 1 - 12 ips
			dstpspeed25ips 2 - 25 ips
			dstpspeed30ips 3 - 30 ips
			dstpspeed50ips 4 - 50 ips
			dstpspeed90ips 5 - 90 ips
			dstpspeed100ips 6 - 100 ips
			dstpspeed125ips 7 - 125 ips
dstpdensity	1	Yes	Tape density - Specify 0 to use default
			0 - Reserved
			dstpdens800bpi 1 - 800 bpi
			dstpdens1600bpi 2 - 1600 bpi
			dstpdens3200bpi 3 - 3200 bpi
			dstpdens6250bpi 4 - 6250 bpi
			dstpdens6400bpi 5 - 6400 bpi
dstpiopbcnt	2	Yes	Number of IOPB's allocated to the drive. The default is determined by the driver. Specify zero to use the default
dstpcachesz	2	Yes	# of sectors in disk cache. Default is determined by the value in the boot block. Specify 0 to use the default.
dstpreserved	46	No	Reserved
dstpuserfield	8	Yes	User defined status. The default is determined by the driver. Specify zero to use the default.

For DISK class devices the second half of the device status table is defined as follows:

Name	Length (bytes)	Settable	Description
dsdkintfac	2	No	Disk interleave factor
dsdkiopbcnt	2	Yes	Number of IOPB's allocated to the drive The default is determined by the driver Specify zero to use the default
dsdknumbsect	4	No	The number of sectors on the volume
dsdksectrack	2	No	The number of sectors on a track
dsdkheads	2	No	The number of heads on the device
dsdkcylinders	2	No	The number of cylinders on the volume
dsdkflags1	2	No	Disk status information. Bit encoded word
dsdkcurcyl	2	No	Current cylinder position
dsdkcachesz	2	Yes	# of sectors in disk cache. Default is determined by the value in the boot block. Specify 0 to use the default.
dsdkentryname	16	No	The name of the drive type
dsdkreserved	20	No	Reserved
dsdkuserfield	8	Yes	User defined status. The default is determined by the driver. Specify zero to use the default.

For TTY class devices the second half of the device status table is defined as follows:

Name	Length (bytes)	Settable	Description
dstymodereg1	1	No	Uart mode register 1
dstymodereg2	1	No	Uart mode register 2
dstycmdreg	1	No	Uart command register
dstytermtype	1	No	Terminal type definition
dstystatreg	1	No	Uart status register
dstypacketterm	1	No	Packet termination conditions
dstyflags1	2	No	Terminal status information
dstyinputcnt	2	No	Characters in input interrupt buffer
dstyoutptcnt	2	No	Characters in output interrupt buffer
dstycolumnpos	2	No	Current column position
dstyinbufsz	2	Yes	Input buffer size in bytes. The default is determined by the driver. Specify zero to use the default.
dstyoutbufsz	2	Yes	Output buffer size in bytes. The default is determined by the driver. Specify zero to use the default.

dstywidth	2	No	Holds terminal width
dstylength	2	No	Holds terminal length
dstysubreadoper	4	No	Holds sub-read operations count
dstysubwriteoper	4	No	Holds sub-write operations count
dstyreserved	26	No	Reserved
dstyuserfield	8	Yes	User defined status. The default is determined by the driver. To use the default, specify zero.

For PIPE class devices the second half of the device status table is defined as follows:

Name	Length (bytes)	Settable	Description
dsppreaderpid	4	No	Process ID of pending reader
dsppwriterpid	4	No	Process ID of pending writer
dspppipeid	4	No	The pipe's ID
dsppbuffersz	2	No	The buffer size in bytes
dsppbuffercnt	2	No	Number of characters in the pipe buffer
dsppreadque	4	No	Address of read queue
dsppwriteque	4	No	Address of write queue
dsppreserved	32	No	Reserved
dsppuserfield	8	Yes	User defined status. The default is determined by the driver. Specify zero to use the default.

For SYNC class devices the second half of the device status table is defined as follows:

Name	Length (bytes)	Settable	Description
dssymodereg1	1	No	Mode register 1 of the uart
dssymodereg2	1	No	Mode register 2 of the uart
dssycmdreg	1	No	Command register of the uart
dssytermtype	1	No	Terminal type definition
dssystatreg	1	No	Status register of uart
dssynumbsync	1	No	Number of sync characters to write
dssyflags1	2	No	Device Status flags. Bit encoded.
dssyinputcnt	2	No	Number of characters in input interrupt buffer
dssyoutputcnt	2	No	Number of characters in output interrupt buffer
dssyinbufsz	2	Yes	Input buffer size in bytes. The default is determined by the driver. Specify zero to use the default.

dssyoutbufsz	2	Yes	Output buffer size in bytes. The default is determined by the driver. Specify zero to use the default.
dssyprevrderr	4	No	Error from previous un-verified read
dssyprevwrerr	4	No	Error from previous no-wait write
dssyprevrdtype	1	No	Type of previous read
dssynumbtrpad	1	No	Number of trailing pads to write
dssyrecsize	2	No	Used in transparent mode with ITBs
dssyreserved	28	No	Reserved
dssyuserfield	8	Yes	User defined status. The default is determined by the driver. Specify zero to use the default.

For NETWORK class devices the second half of the device status table is defined as follows:

Name	Length (bytes)	Settable	Description
dsnkflags	2	No	Device status flags. Bit encoded. Bit Name / Bit # / Description dsnkbyte 0 0=datagram mode 1=byte mode dsnkmodemctrl 1 0=not enabled 1=modem ctrl enabled
dsnkwindowsize	1	No	Window size the circuit will use
dsnkreserved	53	No	Reserved
dsnkuserfield	8	No	User defined status

For NONDEV class devices the second half of the device status table is defined as follows:

Name	Length (bytes)	Settable	Description
dsnduserfield	64	No	User defined status

For QUEUE class devices the second half of the device status table is defined as follows:

Name	Length (bytes)	Settable	Description
dsquassocdev	9	No	A null terminated string containing the name of the physical printer device
dsqusenddev	9	No	A null terminated string containing the name of the physical device that control messages are to be sent to

dsquformname	10	No	A null terminated string containing the current form name
dsqunumexec	2	No	This is the maximum number of entries that can execute concurrently
dsqucurnumexec	2	No	This is the number of entries that are currently active
dsquflags	2	Yes	Device Status flags. Bit encoded.

Bit Name	Bit #	Description
dsquflupdating	0	Updating current queue control file
dsquflqmstay	1	Queue manager process will remain running even when the queue is empty
dsquflnorestart	2	When the queue is mounted it does not restart jobs in queue

dsqulength	2	No	Holds the length of the forms of the printer associated with this queue
dsquwidth	2	No	Holds the width of the forms of the printer associated with this queue
dsqunextentry	4	No	Entry number of the next entry queued
dsqutype	1	Yes	The type of queue this is. Values are:

Value Name	Value	Description
dsqutpprint	1	Print type queue
dsqutpjob	2	Job entry queue

dsqubaseprior	1	No	Priority that entries will be queued at if they specify the default priority
dsqureserved	20	No	Reserved
dsquuserfield	8	No	User defined status

label	- Address of a 17 byte string to receive the device label. The returned string will be null terminated (up to 16 valid characters and a null).
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
errnomemavail	(7)	All available memory has been allocated.

errinvdevnam	(130)	The specified devicename is syntactically incorrect.
errnoexecpriv	(143)	The process does not have Execute Privilege for the file.
errnoreadpriv	(144)	The process does not have Read Privilege for the file.
errimprdmnt	(164)	This device was improperly dismounted.
errinvcloper	(173)	The operation is inappropriate for the device class.
errdevnamexs	(179)	The specified device is already mounted.
errinvclass	(180)	The MCS does not recognize the specified device class.
errnobbfound	(181)	The specified volume has no valid boot block.
errinvdmreq	(183)	The process requested more than 3964 bytes of dynamic memory.
errnoclass	(185)	The device class handler was not loaded when the system was booted.
errprevinit	(188)	The specified device is already mounted, and has another name.
errmntasync	(190)	The specified device has already been mounted for synchronous use.
errmntsync	(191)	The specified device has already been mounted for asynchronous use.
errinvdriver	(216)	The specified file does not contain a device driver.
errdevwrtprot	(269)	The specified device is write-protected.
errcantreaddsr	(308)	The size of the device driver does not match its expected size.
errinvdrvnum	(311)	A value in at least one field of the devicename is disallowed.
errnohardware	(312)	The PC board for the specified device is not installed.

See Also:

_dismnt	- Dismount a logical device
_flush	- Flush I/O buffers to the device
_getdnam	- Get devicename
_getdst	- Get device status
_giodst	- Get device status with LUN
_setdst	- Set device status
_siodst	- Set device status with LUN

Assembler Calling Sequence:

```
%%sys$disk/sysincl.sys/dstatdisp.asm
push    dname                  ;address - devicename
push    driver                 ;address - buffer containing driver
push    class                  ;value - device class
push    dstat                  ;address - initial device status
push    label                  ;address - device label
push    status                 ;address - result of the operation
jsr     _memmnt                ;mount a logical device from memory
```

C Function Declaration:

```
#include "sys$disk/sysincl.sys/dstatdisp.h"
                               /* mount a logical device from memory */
long                           /* returns result of the operation */
_memmnt(dname, driver, class, dstat, label)
        char dname[94];        /* devicename */
        char *driver;          /* buffer containing driver */
        long class;            /* device class */
        devicestatus dstat     /* initial device status */
        char label[17];        /* device label */
```

FORTRAN Subroutine Declaration:

```
c                              ! mount a logical device from memory
        subroutine _memmnt(dname, driver, class, dstat, label,
                          status)
            character*94 dname ! devicename
            character*(*) driver ! buffer containing driver
            integer*4 class    ! device class
            character*(*) dstat ! initial device status
            character*17 label ! device label
            integer*4 status   ! result of the operation
```

Pascal Procedure Declaration:

Note - If passing a device status block, use the following expression: cast(vloc(devicestatus),longint)

```
%%sys$disk/sysincl.sys/dstatdisp.pas
procedure _memmnt(              {** mount a logical device from memory}
        dname    : string[93];   {** devicename}
        driver   : ^array_of_char; {** buffer containing driver}
        class    : longint;      {** device class}
        dstat    : longint;      {** initial device status}
    var vlabel   : string[16];   {** device label}
    var status   : longint       {** result of the operation}
); external;
```

Mount a logical device.

Description:

This system call is used to announce the existence of a device to the system. The system then mounts the device by loading a driver and initializing the device. If the device driver is already present in memory, a new one is not loaded, rather the current driver is shared.

For disk and tape class devices which are not mounted "special", the owner of the volume and the protection specification for each class of user is specified in the volume label.

For TTY, pipe and sync class devices, the owner of the device becomes the UIC of the process issuing the call to _mount. The protection mask for the device will be the default protection mask associated with the calling process.

For devices mounted "special", the owner of the device becomes the UIC of the process issuing the call to _mount. The protection mask for the device will be the default protection mask associated with the calling process.

The process must have read privilege to the device containing the device driver, execute privilege to all directories in the path to the device driver, read privilege to the directory containing the device driver and read privilege to the file containing the device driver.

If the process has operator privilege, it can mount a device using a device driver that is not installed. If the process does not have operator privilege, it can only mount devices using installed device drivers. In either case, the process must satisfy the following requirements.

If the driver is specified by fcb.seq number, the process must have read privilege to the device containing the driver and read privilege to the file containing the driver.

In addition, the process must have execute access to the device being mounted according to the owner and group ID (UIC) of the volume and its protection mask. Note that any process can mount a TTY, pipe or sync class device.

The process must have operator privilege in order to mount any device as "special" (diskspc, ttyspc, etc.).

Related Privileges:

none	- Allows mounting of device if the process has privileges as described above and the driver has been installed.
altuic	- Allows mounting of device if the owner of the image file of the current process has access to the file and device as described above.
bypass	- Allows mounting of device independent of the file protection.
operator	- Allows mounting of devices as 'special'. Also allows mounting devices with uninstalled drivers.
system -	Allows mounting of device if the system has access to the file and device as described above.

Parameters:

dname	- Address of a null terminated string containing the name by which the device will be known. This string will be translated automatically by the MCS to its logical equivalent. This string may contain up to 93 valid characters followed by a null.
driver	- Address of a null terminated string containing the name of the file which contains the device driver. If a driver with the same identifier (irrespective of file name) is found in the system, the driver is not loaded. This string will be translated automatically by the MCS to its logical equivalent. This string may contain up to 93 valid characters followed by a null.

class - The device class. Valid classes are:

0,1	- Character class device (TTYSpecial, TTY)
2,3	- Tape class device (TapeSpecial, Tape)
4,5	- Disk class device (DiskSpecial, Disk)
6,7	- Network class device (NetworkSpecial, Network)
8,9	- Pipe class device (PipeSpecial, Pipe)
10,11	- Sync class device (SyncSpecial, Sync)
12,13	- Queue class device (QueueSpecial, Queue)
14,15	- Nondev class device (NondevSpecial, Nondev)

dstat - The address of a 128 byte buffer containing the initial device status to be assigned the device when it is mounted. If this parameter is zero the default device status is used.

This parameter has two purposes:

1) To provide an opportunity to set device characteristics that, unless set properly, would not allow the device to be mounted, e.g., the tape block size.
2) To set device characteristics that could otherwise not be changed once the device is mounted, e.g., disk cache size.

This parameter is not meant to be a substitute for _setdst. As such, not all of the values that can be specified with _setdst can be specified in this parameter.

The device status table is divided into two parts. The first half is device independent and is composed of the following fields:

Name	Length (bytes)	Settable	Description
dsclassid	2	No	The device class. Uses the class parameter to the _mount system call.
dsdriverid	2	No	Unique ID number for this device driver
dsblksz	2	Yes	block size of the device, e.g., sector size. For disks, the sector size is either 512 bytes or 1024 bytes, determined by the driver. Note that disk sector size cannot be changed. For tapes, the default is 1024 bytes. Specify zero to accept the default.
dsharderr	2	No	The hard error count for the device
dssofterr	2	No	The soft error count for the device
dsreadoper	4	No	Number of read operations on this device

dswriteoper	4	No	Number of write operations on this device
dsmaxnumdev	2	No	Maximum # of devices this driver can handle
dscurnumdev	2	No	Number of devices currently mounted using this device driver
dsnumtoretry	2	Yes	Number of times to retry before reporting a hard error. The default is determined by the driver. Specify zero (0) to accept the default.
dserrorreason	4	No	Hardware error code for the last error
dsreserved	30	No	Reserved
dsnexttableptr	4	No	Address of next device status table

The second half of the device status table is device class dependent. For TAPE class devices the second part is defined as follows:

Name	Length (bytes)	Settable	Description
dstpstatus	2	No	Tape device status
dstpflagsl	2	Yes	Tape status information. Bit encoded. If zero is specified, the default is used.
dstpspeed	1	Yes	Tape speed - Specify zero to use default

dstpflagsl bits:

Bit name	bit #	Description
dstpdoraw	0	0=Read after write disabled 1=Read after write enabled
dstpreadahead	1	0=Read ahead enabled 1=Read ahead disabled
dstperrintenb	2	0=Error interrupts are enabled 1=Error interrupts are disabled

dstpspeed values:

	0 - Reserved
dstpspeed12ips	1 - 12 ips
dstpspeed25ips	2 - 25 ips
dstpspeed30ips	3 - 30 ips
dstpspeed50ips	4 - 50 ips
dstpspeed90ips	5 - 90 ips
dstpspeed100ips	6 - 100 ips
dstpspeed125ips	7 - 125 ips

dstpdensity	1	Yes	Tape density - Specify 0 to use default 0 - Reserved dstpdens800bpi 1 - 800 bpi dstpdens1600bpi 2 - 1600 bpi dstpdens3200bpi 3 - 3200 bpi dstpdens6250bpi 4 - 6250 bpi dstpdens6400bpi 5 - 6400 bpi
dstpiopbcnt	2	Yes	Number of IOPB's allocated to the drive. The default is determined by the driver. Specify zero to use the default
dstpcachesz	2	Yes	# of sectors in disk cache. Default is determined by the value in the boot block. Specify 0 to use the default.
dstpreserved	46	No	Reserved
dstpuserfield	8	Yes	User defined status. The default is determined by the driver. Specify zero to use the default.

For DISK class devices the second half of the device status table is defined as follows:

Name	Length (bytes)	Settable	Description
dsdkintfac	2	No	Disk interleave factor
dsdkiopbcnt	2	Yes	Number of IOPB's allocated to the drive The default is determined by the driver Specify zero to use the default
dsdknumbsect	4	No	The number of sectors on the volume
dsdksectrack	2	No	The number of sectors on a track
dsdkheads	2	No	The number of heads on the device
dsdkcylinders	2	No	The number of cylinders on the volume
dsdkflagsl	2	No	Disk status information. Bit encoded word
dsdkcurcyl	2	No	Current cylinder position
dsdkcachesz	2	Yes	# of sectors in disk cache. Default is determined by the value in the boot block. Specify 0 to use the default.
dsdkentryname	16	No	The name of the drive type
dsdkreserved	20	No	Reserved
dsdkuserfield	8	Yes	User defined status. The default is determined by the driver. Specify zero to use the default.

For TTY class devices the second half of the device status table is defined as follows:

Name	Length (bytes)	Settable	Description
dstymodereg1	1	No	Uart mode register 1
dstymodereg2	1	No	Uart mode register 2
dstycmdreg	1	No	Uart command register
dstytermtype	1	No	Terminal type definition
dstystatreg	1	No	Uart status register
dstypacketterm	1	No	Packet termination conditions
dstyflags1	2	No	Terminal status information
dstyinputcnt	2	No	Characters in input interrupt buffer
dstyoutptcnt	2	No	Characters in output interrupt buffer
dstycolumnpos	2	No	Current column position
dstyinbufsz	2	Yes	Input buffer size in bytes. The default is determined by the driver. Specify zero to use the default.
dstyoutbufsz	2	Yes	Output buffer size in bytes. The default is determined by the driver. Specify zero to use the default.
dstywidth	2	No	Holds terminal width
dstylength	2	No	Holds terminal length
dstysubreadoper	4	No	Holds sub-read operations count
dstysubwriteoper	4	No	Holds sub-write operations count
dstyreserved	26	No	Reserved
dstyuserfield	8	Yes	User defined status. The default is determined by the driver. To use the default, specify zero.

For PIPE class devices the second half of the device status table is defined as follows:

Name	Length (bytes)	Settable	Description
dsppreaderpid	4	No	Process ID of pending reader
dsppwriterpid	4	No	Process ID of pending writer
dspppipeid	4	No	The pipe's ID
dsppbuffersz	2	No	The buffer size in bytes
dsppbuffercnt	2	No	Number of characters in the pipe buffer
dsppreadque	4	No	Address of read queue
dsppwriteque	4	No	Address of write queue
dsppreserved	32	No	Reserved
dsppuserfield	8	Yes	User defined status. The default is determined by the driver. Specify zero to use the default.

For SYNC class devices the second half of the device status table is defined as follows:

Name	Length (bytes)	Settable	Description
dssymodereg1	1	No	Mode register 1 of the uart
dssymodereg2	1	No	Mode register 2 of the uart
dssycmdreg	1	No	Command register of the uart
dssytermtype	1	No	Terminal type definition
dssystatreg	1	No	Status register of uart
dssynumbsync	1	No	Number of sync characters to write
dssyflags1	2	No	Device Status flags. Bit encoded.
dssyinputcnt	2	No	Number of characters in input interrupt buffer
dssyoutputcnt	2	No	Number of characters in output interrupt buffer
dssyinbufsz	2	Yes	Input buffer size in bytes. The default is determined by the driver. Specify zero to use the default.
dssyoutbufsz	2	Yes	Output buffer size in bytes. The default is determined by the driver. Specify zero to use the default.
dssyprevrderr	4	No	Error from previous un-verified read
dssyprevwrerr	4	No	Error from previous no-wait write
dssyprevrdtype	1	No	Type of previous read
dssynumbtrpad	1	No	Number of trailing pads to write
dssyrecsize	2	No	Used in transparent mode with ITBs
dssyreserved	28	No	Reserved
dssyuserfield	8	Yes	User defined status. The default is determined by the driver. Specify zero to use the default.

For NETWORK class devices the second half of the device status table is defined as follows:

Name	Length (bytes)	Settable	Description		
dsnkflags	2	No	Device status flags. Bit encoded.		
			Bit Name	Bit #	Description
			dsnkbyte	0	0=datagram mode 1=byte mode
			dsnkmodemctrl	1	0=not enabled 1=modem ctrl enabled
dsnkwindowsize	1	No	Window size the circuit will use		
dsnkreserved	53	No	Reserved		
dsnkuserfield	8	No	User defined status		

For NONDEV class devices the second half of the device status table is defined as follows:

Name	Length (bytes)	Settable	Description
dsnduserfield	64	No	User defined status

For QUEUE class devices the second half of the device status table is defined as follows:

Name	Length (bytes)	Settable	Description
dsquassocdev	9	No	A null terminated string containing the name of the physical printer device
dsqusenddev	9	No	A null terminated string containing the name of the physical device that control messages are to be sent to
dsquformname	10	No	A null terminated string containing the current form name
dsqunumexec	2	No	This is the maximum number of entries that can execute concurrently
dsqucurnumexec	2	No	This is the number of entries that are currently active
dsquflags	2	Yes	Device Status flags. Bit encoded.
dsqulength	2	No	Holds the length of the forms of the printer associated with this queue
dsquwidth	2	No	Holds the width of the forms of the printer associated with this queue
dsqunextentry	4	No	Entry number of the next entry queued

Bit Name	Bit #	Description
dsquflupdating	0	Updating current queue control file
dsquflqmstay	1	Queue manager process will remain running even when the queue is empty
dsquflnorestart	2	When the queue is mounted it does not restart jobs in queue

dsqutype	1	Yes	The type of queue this is. Values are:

Value Name	Value	Description
dsqutpprint	1	Print type queue
dsqutpjob	2	Job entry queue

dsqubaseprior	1	No	Priority that entries will be queued at if they specify the default priority
dsqureserved	20	No	Reserved
dsquuserfield	8	No	User defined status

label	- Address of a 17 byte string to receive the device label. The returned string will be null terminated. (up to 16 valid characters and a null)
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
errnomemavail	(7)	All available memory has been allocated.
errinvdevnam	(130)	The specified devicename is syntactically incorrect.
errundevnam	(131)	The MCS does not recognize the devicename. Is the device mounted?
errfilnotfnd	(133)	The specified file could not be found.
errreadlock	(135)	The specified file is read-locked.
errreadleof	(140)	The process tried to read past the logical end of a file.
errnoexecpriv	(143)	The process does not have Execute Privilege for the file.
errnoreadpriv	(144)	The process does not have Read Privilege for the file.
errinvfnstr	(147)	The specified filename is syntactically incorrect.
errinvdirfle	(148)	The specified directory is not a directory type file
errinvdirstr	(149)	The specified directory name is syntactically incorrect.
errimprdmnt	(164)	This device was improperly dismounted.
errinvcloper	(173)	The operation is inappropriate for the device class.
errdirnotfnd	(177)	The specified directory does not exist.
errdevnamexs	(179)	The specified device is already mounted.
errinvclass	(180)	The MCS does not recognize the specified device class.
errnobbfound	(181)	The specified volume has no valid boot block.

errinvdmreq	(183)	The process requested more than 3964 bytes of dynamic memory.
errnoclass	(185)	The device class handler was not loaded when the system was booted.
errprevinit	(188)	The specified device is already mounted, and has another name.
errmntasync	(190)	The specified device has already been mounted for synchronous use.
errmntsync	(191)	The specified device has already been mounted for asynchronous use.
errinvdriver	(216)	The specified file does not contain a device driver.
errdevwrtprot	(269)	The specified device is write-protected.
errcantreaddsr	(308)	The size of the device driver does not match its expected size.
errinvdrvnum	(311)	A value in at least one field of the devicename is disallowed.
errnohardware	(312)	The PC board for the specified device is not installed.
		Device integrity errors

See Also:

_dismnt - Dismount a logical device
_flush - Flush I/O buffers to the device
_getdnam- Get devicename
_getdst - Get device status
_giodst - Get device status with lun
_setdst - Set device status
_siodst - Set device status with lun

Assembler Calling Sequence:

```
%%sys$disk/sysincl.sys/dstatdisp.asm
push    dname                     ;address - devicename
push    driver                    ;address - driver file name
push    class                     ;value - device class
push    dstat                     ;address - initial device status
push    label                     ;address - device label
push    status                    ;address - result of the operation
jsr     _mount                    ;mount a logical device
```

C Function Declaration:

```
    #include "sys$disk/sysincl.sys/dstatdisp.h"
                                      /* mount a logical device */
    long                              /* returns result of the operation */
    _mount(dname, driver, class, dstat, label)
            char dname[94];           /* devicename */
            char driver[94];          /* driver file name */
            long class;               /* device class */
            devicestatus *dstat;      /* initial device status */
            char label[17];           /* device label */
```

FORTRAN Subroutine Declaration:

```
    c                                 ! mount a logical device
            subroutine mount(dname, driver, class, dstat, label, status)
                character*94 dname    ! devicename
                character*94 driver   ! driver file name
                integer*4 class       ! device class
                character*(*) dstat   ! initial device status
                character*17 label    ! device label
                integer*4 status      ! result of the operation
```

Pascal Procedure Declaration:

Note - If passing a device status block, use the following expression: cast(vloc(devicestatus),longint)

```
    %%sys$disk/sysincl.sys/dstatdisp.pas
    procedure _mount(                 {** mount a logical device}
            dname   : string[93];     {** devicename}
            driver  : string[93];     {** driver file name}
            class   : longint;        {** device class}
            dstat   : longint;        {** initial device status}
        var vlabel  : string[16];     {** device label}
        var status  : longint         {** result of the operation}
    ); external;
```

Multiple create process.

Description:

This call is similar to _crproc except that it creates several instances of the process. It is to be used in the situation where you want to start up multiple instances of the same process on several different terminals. The image file from which the processes are created is the same for all the processes.

Each process under control of the operating system must be created by a call to the operating system. When a process is created, it is called a child process. The process that created it is called its parent process.

This system call only allows forking of child processes. Forked processes run in parallel with the parent process.

The calling process must have read privilege to the device containing the image file, execute privilege to all directories in the path leading to the directory containing the image file, read privilege to the directory containing the image file and execute privilege to the file containing the child process image for successful creation of the child process.

If the image file is specified by fcb.seq number then the process must have read privilege to the device containing the image file and execute privilege to the file containing the image.

Without the setpriv privilege, the child processes may not be given more privileges than the parent has.

Related Privileges:

none	- Allows the parent process to create a child from an image file to which the parent has access as described above.
bypass	- Allows the parent process to create a child process independent of the file protection.
setpriv	- Allows the parent process to give the child process more privileges than thos possessed by the parent.

setprior - Allows the parent process to initate a child at a higher priority level and/or with a higher time slice than the parent.

Parameters:

siteid - The siteid of the system on which the processes are to be created. If the siteid is zero, the processes will be created on the same system as the calling process.

fname - Address of a 94 byte null terminated string specifying the name of the file containing the process image. (93 significant characters plus a null). This string will be translated to its logical equivalent.

repit - The number of repetitions, i.e. the number of identical processes to intiate from the image file.

pname - Address of an array of pointers to 17 byte null terminated strings containing the process names to be given the new processes. These strings are used for human identification. There must be as many pointers as there are processes to be created. (as indicated by the parameter repit) They may contain up to 16 valid characters followed by a null.

priv - The privilege mask contains a bit mask of privileges to be given to the child processes. Each child process receives the same privileges. A -1 ($FFFFFFFF) indicates that the children should receive the same privileges that the parent has. Privileges are bit encoded as follows:

Bit Name	Bit	Description
pcbpvsetpriv	0	setpriv
pcbpvsystem	1	system
pcbpvreadphys	2	readphys
pcbpvwritephys	3	writephys
pcbpvsetprior	4	setprior
pcbpvchngsuper	5	chngsuper
pcbpvbypass	6	bypass
pcbpvoperator	7	operator
pcbpvaltuic	8	altuic
pcbpvworld	9	world
pcbpvgroup	10	group
pcbpvnetwork	11	network
pcbpvsetattr	12	setattr
	13-31	Reserved. Must be set to zero

priort	- The priority level (0..3) at which the child processes will execute. Level 0 is the highest priority. A minus one (-1 or $FFFFFFFF) in this parameter means to use the same priority as the parent process.
tslice	- The time slice value. The maximum amount of time each of the child processes will be allowed to run each time they are scheduled. This time is specified in .01 milliseconds. A time slice of 100 represents 1 millisecond. A minus one (-1 or $FFFFFFFF) means to use the same time slice as the parent process.
uic	- The user identification code of the child processes. The most significant 16 bits represent the owner id and the least significant 16 bits represent the group id. Each child process will have the same uic.
sysin	- Address of an array of pointers to null terminated strings containing the names of the standard input files for each of the child processes. Each string will be translated by MCS to its logical equivalent. There must be as many pointers as there are processes to be created. The 1st pointer points to the string containing the name of the standard input file for the first process to be created. The 2nd pointer points to the string containing the name of the standard input file for the second process to be created, and so on. The strings will be assigned to the logical name "SYS$INPUT in the logical name table of the corresponding child process. The strings passed are not checked for validity. Each string may contain up to 93 significant characters followed by a null.
sysout	- Address of an array of pointers to null terminated strings containing the names of the standard output files for each of the child processes. Each string will be translated by MCS to its logical equivalent. There must be as many pointers as there are processes to be created. The 1st pointer points to the string containing the name of the standard output file for the first process to be created. The 2nd pointer points to the string containing the name of the standard output file for the second process to be created, and so on. The strings will be assigned to the logical name "SYS$OUTPUT in the logical name table of the corresponding child process. The strings passed are not checked for validity. Each string may contain up to 93 significant characters followed by a null.

syserr	- Address of an array of pointers to null terminated strings containing the names of the standard error files for each of the child processes. Each string will be translated by MCS to its logical equivalent. There must be as many pointers as there are processes to be created. The 1st pointer points to the string containing the name of the standard error file for the first process to be created. The 2nd pointer points to the string containing the name of the standard error file for the second process to be created, and so on. The strings will be assigned to the logical name "SYS$ERROR in the logical name table of the corresponding child process. The strings passed are not checked for validity. Each string may contain up to 93 significant characters followed by a null.
cmd	- Address of an array of pointers to the command lines for each process. Each command line may contain up to 3072 bytes. The command lines may contain any data whatever to be passed from the parent to the children. There must be as many pointers to command lines as there are child processes to create. The first pointer points to the command line for the first process to be created. The 2nd pointer points to the command line for the second process to be created, and so on. The command lines will appear on the top of the child process's stack as each child process begins. The long word at the top of the child's stack is the length in bytes of the command line. At the location (USP+4) on the child's stack is a long word which contains the starting address of the command line.
cmdlen	- Address of an array of long words containing the length of each of the command lines. The length is specified in bytes.
pid	- Address of an array of long words to receive the pids of the child processes. This array is assumed to be long enough to contain the pids of as many processes as were requested to be created. (see repit)
prccnt	- Address of a long word to receive the number of processes that were successfully created. If this number is less than the number of desired processes (see repit), then the status variable indicates the error that prevented the "next" process from being created.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
errnomemavail	(7)	All available memory has been allocated.
errinvsiteid	(8)	The specified site id does not exist.
errnotimfle	(21)	The specified file is not an image file.
errimagebmbad	(53)	(MCS error) The bitmap changed during the creation of the process.
errinvdevnam	(130)	The specified devicename is syntactically incorrect.
errundevnam	(131)	The MCS does not recognize the devicename. Is the device mounted?
errfilnotfnd	(133)	The specified file could not be found.
errreadleof	(140)	The process tried to read past the logical end of a file.
errnoexecpriv	(143)	The process does not have Execute Privilege for the file.
errnoreadpriv	(144)	The process does not have Read Privilege for the file.
errinvfnstr	(147)	The specified filename is syntactically incorrect.
errinvdirfle	(148)	The specified directory is not a directory.
errinvdirstr	(149)	The specified directory name is syntactically incorrect.
errdirnotfnd	(177)	The specified directory does not exist.
errfilopen	(202)	The process tried to simultaneously open more than one tape file.

See Also:

_clone	- Clone an existing process
_crprcs	- Simplified create process
_crproc	- Create a new process
_exproc	- Terminate the specified process
_setpnam	- Change process name
_setpri	- Change priority level
_settmsl	- Change scheduling time slice
_setuic	- Set process uic

Assembler Calling Sequence:

```
push    siteid                    ;value - system id
push    fname                     ;address - name of image file
push    repit                     ;value - number of instances
push    pname                     ;address - process names
push    priv                      ;value - process privilege
push    priort                    ;value - process priority
push    tslice                    ;value - process time slice
push    uic                       ;value - user identification code
push    sysin                     ;address - standard input files
push    sysout                    ;address - standard output files
push    syserr                    ;address - standard error files
push    cmd                       ;address - command lines
push    cmdlen                    ;address - length of command lines
push    pid                       ;address - child's pid
push    prccnt                    ;address - process count
push    status                    ;address - result of the operation
jsr     _mulcrps                  ;multiple create process
```

C Function Declaration:

```
                                  /* multiple create process */
long                              /* returns result of the operation */
_mulcrps (siteid, fname, repit, pname, priv, priort, tslice, uic,
            sysin, sysout, syserr, cmd, cmdlen, pid, prccnt)
         long siteid;             /* system id */
         char fname[94];          /* name of image file */
         long repit;              /* number of instances */
         char *pname[];           /* process name */
         long priv;               /* process privilege */
         long priort;             /* process priority */
         long tslice;             /* process time slice */
         long uic;                /* user identification code */
         char *sysin[];           /* standard input files */
         char *sysout[];          /* standard output files */
         char *syserr[];          /* standard error files */
         char *cmd[];             /* command lines */
         long cmdlen[];           /* length of command lines */
         long pid[];              /* child's pid */
         long *prccnt;            /* process count */
```

FORTRAN Function Declaration:

NOTE: This system call is not directly accessible from FORTRAN.

Pascal Procedure Declaration:

```
procedure _mulcrps(              {** multiple create process}
        siteid  : longint;       {** system id}
        fname   : string[93];    {** name of image file}
        repit   : longint;       {** number of instances}
        pname   : ^array_of_char;{** process name}
        priv    : longint;       {** process privilege}
        priort  : longint;       {** process priority}
        tslice  : longint;       {** process time slice}
        uic     : longint;       {** user identification code}
        sysin   : ^array_of_char;{** standard input files}
        sysout  : ^array_of_char;{** standard output files}
        syserr  : ^array_of_char;{** standard error files}
        cmd     : ^array_of_char;{** command line}
        cmdlen  : array[1..1] of longint; {** length of command lines}
    var pid     : array[1..1] of longint; {** child's pid}
    var prccnt  : longint;       {** process count}
    var status  : longint        {** result of the operation}
); external;
```

Open a file.

Description:

After logical name translation, the specified file is made available to the calling process for the type(s) of I/O requested. Upon successful completion it returns the logical unit number (lun) which is used to identify the file during subsequent operations.

Unless the process has bypass privilege, it must have read/write privilege to the device containing the file, execute privilege to all directories in the path leading to the file, read privilege to the directory containing the file, and read/write privilege to the file itself in order for the file to be successfully opened. If the "opdelete" mode bit is set (delete file upon closing), the process must also have delete privilege to the file.

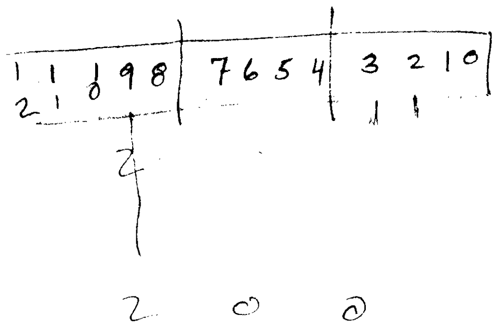
If fname is specified in the fcb.seq number format, the process must have read/write privilege to the device containing the file and read/write privilege to the file itself in order for the file to be successfully opened.

Related Privileges:

none	- Allows opening if process has access to the file as described above.
altuic	- Allows opening if the owner of the image file for the current process has access as to the file as described above.
bypass	- Allows the process to open the file independent of the file protection.
system	- Allows opening if the system has access as described above.

Parameters:

fname	- Address of a null terminated string containing the name of the file to be opened. The string will be translated automatically by WMCS to its logical equivalence. This string may contain up to 93 valid characters followed by a null.
mode	- Bit encoded long word specifying the type of access required. The following description explains the options when the specified bit is set (1):



Bit Name	Bit #	Description
opreadacc	0	Read access - Requests permission to read the file.
opwriteacc	1	Write access - Requests permission to write the file.
opreadlock	2	Read lock - Requests permission for exclusive read access to the file.
opwritelock	3	Write lock - Requests permission for exclusive write access to the file.
opdelete	4	Delete - Requests that the file be deleted upon closing.
opappend	5	Append - Specifies that the initial file position be at the logical end of file.
opfastread	6	Fast read - Specifies that the file will be read asynchronously. That is, that control returns to the user process before the data have actually been read. As records are read, they will be transferred directly into the process's logical address space bypassing the device cache. This bit is only valid for disk class devices. Other requirements are 1) Supports only requests for complete sectors only, 2) Process buffer must be on a word boundary, 3) Request cannot cross a 4 Kbyte page boundary. Use the _frdwait system call to determine when asynchronous reads are complete.
opnextfile	7	Open next file - On a tape device, specifies to open the "next" file without regard to the filename.
opnordahead	8	No read ahead - Specifies that read ahead is not to be done on the opened file.
opnotruncfile	9	No truncate - Specifies that when the file is closed the extra physical sectors allocated to the file are not to be released.
used by create	10	Reserved. Must be set to zero.

cropenshared	11	Open shared - Specifies that if the current process or any ancestor of the current process has a file with the specified name (fname) and with the same access modes currently open, this process will share the file with the ancestor, including the default file position. As the file is read or written, the default position is adjusted for both the current process and the ancestor.
opzerodelete	12	Zero delete - Zero each sector of the file before deleting the file. This bit is only valid if the file is being deleted (via cldelete or some other way).
	13-31	Reserved. Must be set to zero.

reclen	- Record length. If this parameter is between 1 and 65534 inclusive, it overrides the default record length specified for the file. Specifying a zero or $FFFFFFFF (-1) for this parameter causes the file to be open with the default record length.
lun	- Address of a long word to receive the logical unit number of the open file.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

[illegible]

errnomemavail	(7)	All available memory has been allocated.
errinvdevnam	(130)	The specified devicename is syntactically incorrect.
errundevnam	(131)	The MCS does not recognize the devicename. Is the device mounted?
errfilnotfnd	(133)	The specified file could not be found.
errreadlock	(135)	The specified file is read-locked.
errwritelock	(136)	The specified file is write-locked.
errinvreclen	(138)	Edit mode 3 requires that the file's record length be set to one.
errnoexecpriv	(143)	The process does not have Execute Privilege for the file.
errnoreadpriv	(144)	The process does not have Read Privilege for the file.
errnowritepriv	(145)	The process does not have Write Privilege for the file.
errnodelpriv	(146)	The process does not have Delete Privilege for the file.

error 94

errinvfnstr	(147)	The specified filename is syntactically incorrect.
errinvdirfle	(148)	The specified directory is not a directory.
errinvdirstr	(149)	The specified directory name is syntactically incorrect.
errinvcloper	(173)	The device class is inappropriate for the operation.
errdirnotfnd	(177)	The specified directory does not exist.
errdirnotfnd	(178)	The file's FCB.SEQ number in the directory file is incorrect.
errfilopen	(202)	The process tried to simultaneously open more than one tape file.

See Also:

_close	- Close a file
_create	- Create a file
_physio	- Perform physical I/O operation
_read	- Read from on open file
_write	- Write to an open file

Assembler Calling Sequence:

```
%%sys$disk/sysincl.sys/sysequ.asm
push    fname                     ;address - file name
push    mode                      ;value - type of access requested
push    reclen                    ;value - record length
push    lun                       ;address - logical unit number
push    status                    ;address - result of the operation
jsr     _open                   ;open a file
```

C Function Declaration:

```
#include "sys$disk/sysincl.sys/sysequ.h"
                                  /* open a file */
long                              /* returns result of the operation */
_open (fname, mode, reclen, lun)
        char fname[94];           /* file name */
        long mode;                /* type of access requested */
        long reclen;              /* record length */
        long *lun;                /* logical unit number */
```

Fortran Subroutine Declaration:

```
    c                                 ! open a file
              subroutine _open (fname, mode, reclen, lun, status)
                  character*94 fname ! file name
                  integer*4 mode     ! type of access requested
                  integer*4 reclen   ! record length
                  integer*4 lun      ! logical unit number
                  integer*4 status   ! result of the operation
```

Pascal Procedure Declaration:

```
    %%sys$disk/sysincl.sys/sysequ.pas
    procedure _open(                {** open a file}
            fname   : string[93];   {** file name}
            mode    : longint;      {** type of access requested}
            reclen  : longint;      {** record length}
            lun     : longint;      {** logical unit number}
        var status  : longint       {** result of the operation}
    ); external;
```

orevnt - Wait for OR of event flags.

Description:

Suspends process execution and waits for any one of a set of event flags to be set. When any of the flags is set, or when the time out value is exceeded, processing resumes.

Related Privileges:

none	- Allows waiting on the event flags of any process with the same owner id and group id (uic) as the calling process.
group	- Allows waiting on the event flags of any process with the same group id as the calling process.
world	- Allows waiting on the event flags of any process.

Parameters:

pid	- Process id of the process whose event flags are to be monitored. A pid of 0 represents the current process. A pid of -1 represents the parent of the current process.
efmask	- Event flag mask. The process will be suspended until one of the bits in the event flag of that corresponds to a one bit in this mask is set.
timout	- Time out value specified in 100ths of a second. Represents the maximum time to wait for one of the specified event flags to be set.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
errprcsnotfnd	(2)	The specified process is not in the system process table.
errtimeout	(128)	A request was not completed within the specified time.

See Also:

_andevnt - Wait for AND of event flags
_clrevnt - Clear event flags
_getevnt - Read event flags

_setevnt - Set event flags

Assembler Calling Sequence:

```
push    pid                         ;value - process id
push    efmask                      ;value - event flag mask
push    timout                      ;value - time out
push    status                      ;address - result of the operation
jsr     _orevnt                     ;wait for OR of event flags
```

C function declaration:

```
                                    /* wait for OR of event flags */
long                                /* returns result of the operation */
_orevnt(pid, efmask, timout)
        long pid;                   /* process id */
        long efmask;                /* event flag mask */
        long timout;                /* time out */
```

Fortran Subroutine Declaration:

```
c                                   ! wait for OR of event flags
        subroutine orevnt(pid, efmask, timout, status)
            integer*4 pid           ! process id
            integer*4 efmask        ! event flag mask
            integer*4 timout        ! time out
            integer*4 status        ! result of the operation
```

Pascal Procedure Declaration:

```
procedure  orevnt(                  {** wait for OR of event flags}
        pid     : longint;          {** process id}
        efmask  : longint;          {** event flag mask}
        timout  : longint;          {** time out value}
    var status  : longint           {** result of the operation}
); external;
```

Get original process privileges.

Description:

Allows a process to inspect the privileges assigned to a process before any installed privileges were added at process creation time.

Related Privileges:

None.

Parameters:

pid - Process ID of the process whose original privileges are to be returned. A PID of 0 represents the current process. A PID of -1 represents the parent of the current process.

priv - Address of a long word to receive the original privilege mask containing a bit mask of privileges assigned to the specified process.

Bit Name	Bit #	Description
pcbpvsetpriv	0	setpriv
pcbpvsystem	1	system
pcbpvreadphys	2	readphys
pcbpvwritephys	3	writephys
pcbpvsetprior	4	setprior
pcbpvchngsuper	5	chngsuper
pcbpvbypass	6	bypass
pcbpvoperator	7	operator
pcbpvaltuic	8	altuic
pcbpvworld	9	world
pcbpvgroup	10	group
pcbpvnetwork	11	network
pcbpvsetattr	12	setattr
	13-32	Reserved.

status - Address of a longword to receive the result of the operation.

Diagnostics:

errprcsnotfnd (2) The specified process is not in the system process table.

See Also:

_crproc	- Create a new process
_getpri	- Get process priority
_getprv	- Get process privileges
_gettmsl	- Get scheduling time slice
_install	- Install a privileged file
_setpri	- Set process priority
_setprv	- Set process privilege
_settmsl	- Change scheduling time slice

Assembler Calling Sequence:

```
push    pid                     ;value - process id
push    priv                    ;address - privilege mask
push    status                  ;address - result of the operation
jsr     _origprv                ;get original process privileges
```

C Function Declaration:

```
                                /* get original process privilege */
long                            /* returns result of the operation */
_origprv(pid, priv)
        long pid;               /* process id */
        long *priv;             /* privilege mask */
```

FORTRAN Subroutine Declaration:

```
c                               ! get original process privilege
        subroutine _origpr(pid, priv, status)
            integer*4 pid       ! process id
            integer*4 priv      ! privilege mask
            integer*4 status    ! result of the operation
```

Pascal Procedure Declaration:

```
procedure _origprv(              {** get original process privilege}
        pid     : longint;       {** process id}
    var priv    : longint;       {** privilege mask}
    var status  : longint        {** result of the operation}
); external;
```

physio - Perform physical I/O operation.

Description:

Performs a direct call to the device driver associated with a successfully opened file or device bypassing the file structure. Allows physical I/O on mounted devices. The device may be mounted as a 'special' (e.g. diskspc).

This is similar to physop, except with this call the device is identified by a logical unit number as opposed to a device name. Accesses via logical unit number are faster than accesses via device name.

To successfully perform the operation, the calling process must have read or write privilege to the device (depending on the operation) and either be the owner of the device (the process uic and the device uic are the same) or have readphys or writephys privilege (depending on the operation).

Related Privileges:

none	- Allows the process to access the device if the owner id and group id (uic) of the process are the same as the uic of the device and the process has read/write privilege as described above.
altuic	- Allows the process to access the device if the owner of the image file for the current process has access to the device as described above.
bypass	- Allows the process to access the device independent of the file protection. This does not obviate the need for either readphys or writephys privilege. This only applies to read or write privilege to the device.
readphys	- Allows physical access for read operations to devices as described above. This does not obviate the need for either read or write privilege to the device.
system	- Allows the process to access the device if the system has access to the device as described above. This does not obviate the need for either readphys or writephys privilege. This only applies to read or write privilege to the device.
writephys	- Allows physical access for write operations to devices as described above. This does not obviate the need for either read or write privilege to the device.

Parameters:

lun - A long word containing the logical unit number of the device to which the I/O operation is to be performed.

func - Which operation to perform. Valid operations are: (Note that some of the commands are device class specific. When ever a class is specified, it also applies to the special form of that class. Commands described for tty class devices also apply to pipe, sync, and nondev class devices.)
The names of these functions are defined in the file /sysincl.sys/contcmd.*

Read from the device
(2) diskreadcmd
(2) tapereadcmd
(2) ttyreadcmd

This will read the specified number of blocks from the given device. This command is valid on this list of devices: Disk, Nondev, Pipe, Sync, Tape, TTY.

Requires read privilege to the device, and if the calling process is not the owner of the device, also requires readphys privilege, unless this is a nondev class device.

parm1 - Address of a buffer to receive the data read. This buffer must be word aligned.
parm2 - A long word containing the block number of the first block to be read. This parameter is not used for tape or tty class devices.
parm3 - A long word containing the number of blocks to read. On tape devices, this parameter represents how many bytes to read. On tapes, this function will never read more than one block.
parm4 - Address of a long word to receive the number of blocks actually read.

Write to the device
(3) diskwritecmd
(3) tapewritecmd
(3) ttywritecmd

This will write the specified number of blocks to the given device. This command is valid on this list of devices: Disk, Nondev, Pipe, Sync, Tape, TTY.

Requires write privilege to the device, and if the calling process is not the owner of the device, also requires writephys privilege, unless this is a nondev class device.

- parm1 - Address of a buffer containing data to be written. This buffer must be word aligned.
- parm2 - A long word containing the block number of the first block to be written. This parameter is not used for tape or tty class devices.
- parm3 - A long word containing the number of blocks to write. On tape devices, this parameter represents the number of bytes to write.
- parm4 - Address of a long word to receive the number of blocks actually written.

Format the device
(4) diskformatcmd

Reformat the given media. This command is valid on this list of devices: Disk.

Requires write privilege to the device, and if the calling process is not the owner of the device, also requires writephys privilege.

- parm1 - Not used.
- parm2 - Not used.
- parm3 - Not used.
- parm4 - Not used.

Erase the device
(4) tapeerasecmd

Erase the data off of the given device. This command is valid on this list of devices: Tape.

Requires write privilege to the device, and if the calling process is not the owner of the device, also requires writephys privilege.

- parm1 - A long word containing subfunction number:
  - (0) tapeerstartvar - Variable length erase start.
  - (1) tapeerstopvar - Variable length erase stop.
  - (2) tapeersecurity - Security erase.

(3) tapeerfixedlen - Fixed length erase.
parm2 - Not used.
parm3 - Not used.
parm4 - Not used.

Purge input buffer (Not implemented)
(4) ttypurgeinputbuffer

This will delete all data in the input typeahead buffer. This command is valid on this list of devices: Nondev, Pipe, Sync, TTY.

Requires write privilege to the device, and if the calling process is not the owner of the device, also requires writephys privilege, unless this is a nondev class device.
parm1 - Not used.
parm2 - Not used.
parm3 - Not used.
parm4 - Not used.

Read device status
(5) diskgetstatuscmd
(5) tapegetstatuscmd
(5) ttygetstatuscmd
(5) quegetstatuscmd

This will read the device status from the given device. This command is valid on this list of devices: Disk, Nondev, Pipe, Queue, Sync, Tape, TTY.

Requires read privilege to the device, and if the calling process is not the owner of the device, also requires readphys privilege, unless this is a nondev class device.
parm1 - Address of a 128 byte buffer to receive the device status. This buffer must be word aligned.
parm2 - Not used.
parm3 - Not used.
parm4 - Not used.

Set device status
(6) disksetstatuscmd
(6) tapesetstatuscmd
(6) ttysetstatuscmd
(6) quesetstatuscmd

This will set the device status on the given device. This command is valid on this list of devices: Disk, Nondev, Pipe, Queue, Sync, Tape, TTY.

Requires write privilege to the device, and if the calling process is not the owner of the device, also requires writephys privilege, unless this is a nondev class device.

- parm1 - Address of a 128 byte buffer containing the new device status. This buffer must be word aligned.
- parm2 - Not used.
- parm3 - Not used.
- parm4 - Not used.

Format specified track(s) (Not implemented)
(7) diskformattrackcmd

This will format a given list of tracks on the device. This command is valid on this list of devices: Disk.

Requires write privilege to the device, and if the calling process is not the owner of the device, also requires writephys privilege.

- parm1 - A long word containing the cylinder number.
- parm2 - A long word containing the head number.
- parm3 - A long word containing the number of cylinders to format.
- parm4 - Address of a long word to receive the number of cylinders actually formatted.

Skip, position the device.
(7) tapeskipcmd

Skip to the specified position on the device. This command is valid on this list of devices: Tape.

Requires read privilege to the device, and if the calling process is not the owner of the device, also requires readphys privilege.

- parm1 - A long word containing subfunction number (type of skip):
  - (0) tapeskiprecords - Skip records
  - (1) tapeskipfiles - Skip files
  - (2) tapeskipbot - Skip to beginning of volume
  - (3) tapeskipeot - Skip to end of volume
- parm2 - Not used.

parm3 - A long word containing the number of units to skip. If parml specifies a skip to beginning of volume then this parameter indicates whether the tape should be positioned before or after the volume label.
- (0) tapeskipbefheader - Leave tape positioned before the volume label. (load point)
- (1) tapeskipaftheader - Leave tape positioned after the volume label. (the position the tape would normally be at after a mount.

parm4 - Address of a long word to receive the number of units actually skipped.

Dial a modem
(7) ttydialcmd

With the given string this will dial out on the given device. This command is valid on this list of devices: Nondev, Pipe, Sync, TTY.

Requires write privilege to the device, and if the calling process is not the owner of the device, also requires writephys privilege, unless this is a nondev class device.
- parml - Address of a dial buffer. Contains characters meaningful to the device. Representing the number to be dialed.
- parm2 - Not used.
- parm3 - The number of bytes to be used in the dial buffer.
- parm4 - Not used.

Insert an entry into the device
(7) queenquecmd

This will insert a given create process record into the queue for execution at the correct time. This command is valid on this list of devices: Queue.

Requires write privilege to the device.
- parml - Address of the queue create process buffer.
- parm2 - Address of the queue entry buffer.
- parm3 - Not used.
- parm4 - Address of a longword to receive the queue entry number.

Set drive configuration table
(8) disksetdrivetblcmd

This will define a new drive configuration table for this device. This describes the media to the driver. This command is valid on this list of devices: Disk.

Requires write privilege to the device, and if the calling process is not the owner of the device, also requires writephys privilege.
parm1 - Address of the new drive configuration table.
parm2 - Not used.
parm3 - Not used.
parm4 - Not used.

Write tape mark
(8) tapewritetapemark

Write a tape mark on the given device. This command is valid on this list of devices: Tape.

Requires write privilege to the device, and if the calling process is not the owner of the device, also requires writephys privilege.
parm1 - Not used.
parm2 - Not used.
parm3 - Not used.
parm4 - Not used.

Hangup a modem
(8) ttyhangupcmd

This will send a hangup command out to the given device. This command is valid on this list of devices: Nondev, Pipe, Sync, TTY.

Requires write privilege to the device, and if the calling process is not the owner of the device, also requires writephys privilege, unless this is a nondev class device.
parm1 - Not used.
parm2 - Not used.
parm3 - Not used.
parm4 - Not used.

Get entry information by index number
(8) quelistcmd

This will receive all of the information about a given entry in the queue. It will access that entry by the current index from the front of the queue. This command is valid on this list of devices: Queue.

Requires read privilege to the device.
- parm1 - Index number from front of file for which entry we want.
- parm2 - Address of buffer to receive the queue create process buffer.
- parm3 - Address of buffer to receive the queue entry buffer.
- parm4 - Address of a longword to receive this entry's entry number.

Set [CONTROL] C pid
(9) ttysetcontcpid

This will specify that the calling process is to be terminated on the next [CONTROL] C character that is received. This command is valid on this list of devices: Nondev, Pipe, Sync, TTY.

Requires write privilege to the device, and if the calling process is not the owner of the device, also requires writephys privilege, unless this is a nondev class device.
- parm1 - Not used.
- parm2 - Not used.
- parm3 - Not used.
- parm4 - Not used.

Delete an entry from the device
(9) quedequecmd

This will delete the given entry from the queue. This command is valid on this list of devices: Queue.

Requires write privilege to the device. Requires that the entry have the same owner id and group id as the caller. or the same group id as the caller and the caller has group privilege, or the caller has world privilege.

- parm1 - Entry number of which entry to delete.
- parm2 - Not used.
- parm3 - Not used.
- parm4 - Not used.

Reset the device
(10) ttyresetcmd

This will reset the device. This command is valid on this list of devices: Nondev, Pipe, Sync, TTY.

Requires write privilege to the device, and if the calling process is not the owner of the device, also requires writephys privilege, unless this is a nondev class device.
parm1 - Not used.
parm2 - Not used.
parm3 - Not used.
parm4 - Not used.

Halt the device
(10) quehaltcmd

This will halt the given queue. This means that no more entries will be executed. Entries can still be added to the queue. This command is valid on this list of devices: Queue.

Requires write privilege to the device, and if the calling process is not the owner of the device, also requires operator privilege.
parm1 - Not used.
parm2 - Not used.
parm3 - Not used.
parm4 - Not used.

Start the device after a halt
(11) questartcmd

This will start the queue after a halt command has been given. This means that ready entries will be executed. This command is valid on this list of devices: Queue.

Requires write privilege to the device, and if the calling process is not the owner of the device, also requires operator privilege.
parm1 - Not used.
parm2 - Not used.
parm3 - Not used.
parm4 - Not used.

Restart an entry on the device
(12) querestartcmd

This will take an entry that is already executing and will terminate the process. It will then restart the entry. This command is valid on this list of devices: Queue.

Requires write privilege to the device. Requires that the entry have the same owner id and group id as the caller, or the same group id as the caller and the caller has group privilege, or the caller has world privilege.
  parml - Entry number of which entry to restart.
  parm2 - Not used.
  parm3 - Not used.
  parm4 - Not used.

Send a break character
  (13) ttysendbreak

This will send a break character out of the given device. This command is valid on this list of devices: Nondev, Pipe, Sync, TTY.

Requires write privilege to the device, and if the calling process is not the owner of the device, also requires writephys privilege, unless this is a nondev class device.
  parml - Not used.
  parm2 - Not used.
  parm3 - Not used.
  parm4 - Not used.

Wait for an entry to complete
  (13) quewaitcmd

This will wait for the given entry to complete. If the entry does not exist, it will return immediatly. This command is valid on this list of devices: Queue

Requires read privilege to the device.
  parml - Entry number of which entry to wait on.
  parm2 - Not used.
  parm3 - Not used.
  parm4 - Not used.

Hold an entry on the device
  (15) queholdcmd

This will hold the given entry. This means that it will not be executed. This command is valid on this list of devices: Queue.

Requires write privilege to the device. Requires that the entry have the same owner id and group id as the caller, or the same group id as the caller and the caller has group privilege, or the caller has world privilege.

parm1 - Entry number of which entry to hold.
parm2 - Not used.
parm3 - Not used.
parm4 - Not used.

Wake an entry after a hold command
(16) quewakecmd

This will wake the given entry after a hold command. This means the entry is available for execution again. This command is valid on this list of devices: Queue.

Requires write privilege to the device. Requires that the entry have the same owner id and group id as the caller, or the same group id as the caller and the caller has group privilege, or the caller has world privilege.

parm1 - Entry number of which entry to wake.
parm2 - Not used.
parm3 - Not used.
parm4 - Not used.

Modify an entry on the device
(18) quemodifycmd

This will modify an entry that is already queued. This command is valid on this list of devices: Queue.

Requires write privilege to the device. Requires that the entry have the same owner id and group id as the caller, or the same group id as the caller and the caller has group privilege, or the caller has world privilege.

parm1 - Entry number of which entry to modify.
parm2 - Address of the new queue create process buffer.
parm3 - Address of the new queue entry buffer.
parm4 - Not used.

Close the given device
(19) queclosecmd

This will close the queue. This means that no more entries can be entered into the queue. Entries that have already been queued will continue to be executed as their turn arrives. This command is valid on this list of devices: Queue.

Requires write privilege to the device, and if the calling process is not the owner of the device, also requires operator privilege.
  parm1 - Not used.
  parm2 - Not used.
  parm3 - Not used.
  parm4 - Not used.

Open the given device after a close
(20) queopencmd

This will open the queue after a close command. This means that more entries may be queued. This command is valid on this list of devices: Queue.

Requires write privilege to the device, and if the calling process is not the owner of the device, also requires operator privilege.
  parm1 - Not used.
  parm2 - Not used.
  parm3 - Not used.
  parm4 - Not used.

Get entry information by entry number
(21) quegetentrycmd

This will receive all of the information about a given entry in the queue. It will access that entry by entry number. This command is valid on this list of devices: Queue.

Requires read privilege to the device.
  parm1 - Entry number of which entry to get.
  parm2 - Address of buffer to receive the queue create process buffer.
  parm3 - Address of buffer to receive the queue entry buffer.
  parm4 - Address of a longword to receive this entry's entry number.

Get default create process record
(22) quegetdefcrpcmd

This will get the default create process record. This record is used when a user redirects output directly to the queue device. This command is valid on this list of devices: Queue.

Requires read privilege to the device.

parm1	- Address of buffer to receive the default queue create process record.
parm2	- Not used.
parm3	- Not used.
parm4	- Not used.

Set default create process record
(23) quesetdefcrpcmd

This will set the default create process record. This record is used when a user redirects output directly to the queue device. This command is valid on this list of devices: Queue.

Requires write privilege to the device, and if the calling process is not the owner of the device, also requires operator privilege.

parm1	- Address of the new default queue create process buffer.
parm2	- Not used.
parm3	- Not used.
parm4	- Not used.

timout	- Maximum amount of time to wait for the operation to complete. Expressed in 100'ths of a second
parm1	- A parameter defined by the function
parm2	- A parameter defined by the function
parm3	- A parameter defined by the function
parm4	- A parameter defined by the function
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
errtimeout	(128)	A request was not completed within the specified time.
errinvdevnam	(130)	The specified devicename is syntactically incorrect.
errundevnam	(131)	The MCS does not recognize the devicename. Is the device mounted?
errinvcloper	(173)	The operation is inappropriate for the device class.
errprevinit	(188)	The specified device is already mounted,

		and has another name.
errinvskpcmd	(206)	The specified skip or erase tape-function is undefined.
errinvdrvnum	(311)	A value in at least one field of the devicename is disallowed.
		Device errors

See Also:

_dismnt	- Dismount a logical device
_getdnam	- Get device name
_getdst	- Get device status
_giodst	- Get device status with lun
_mount	- Mount a logical device
_open	- Open a file
_physop	- Perform physical device operation
_setdst	- Set device status
_siodst	- Set device status with lun
_skip	- Position tape

Assembler Calling Sequence:

```
%%sys$disk/sysincl.sys/sysequ.asm
    push    lun                         ;value - logical unit number
    push    func                        ;value - which function
    push    timout                      ;value - time out
    push    parm1                       ;address/value - 1st parameter
    push    parm2                       ;address/value - 2nd parameter
    push    parm3                       ;address/value - 3rd parameter
    push    parm4                       ;address/value - 4th parameter
    push    status                      ;address - result of the operation
    jsr     _physio                     ;perform physical I/O operation
```

C function declaration:

```
    #include "sys$disk/sysincl.sys/sysequ.h"
                                        /* perform physical I/O operation */
    long                                /* returns result of the operation */
    _physio(lun, func, timout, parm1, parm2, parm3, parm4)
            long lun;                   /* logical unit number */
            long func;                  /* which function */
            long timout;                /* time out */
            long parm1;                 /* 1st parameter */
            long parm2;                 /* 2nd parameter */
            long parm3;                 /* 3rd parameter */
            long parm4;                 /* 4th parameter */
```

Fortran Subroutine Declaration:

```
    c                                   ! perform physical I/O operation
            subroutine physio(lun, func, timout, parm1, parm2,
&                     parm3, parm4, status)
            integer*4 lun           ! logical unit number
            integer*4 func          ! which function
            integer*4 timout        ! time out
            integer*4 parm1         ! 1st parameter
            integer*4 parm2         ! 2nd parameter
            integer*4 parm3         ! 3rd parameter
            integer*4 parm4         ! 4th parameter
            integer*4 status        ! result of the operation
```

Pascal Procedure Declaration:

Note that all of the parm components are defined as long integers. Where the address of a variable is to be passed, use the following function. _physio(...,cast(vloc(variable),longint),...)

```
%%sys$disk/sysincl.sys/sysequ.pas
procedure  physio(                 {** perform physical I/O operation}
        lun     : longint;         {** logical unit number}
        func    : longint;         {** which function}
        timout  : longint;         {** timeout value}
        parm1   : longint;         {** 1st parameter}
        parm2   : longint;         {** 2nd parameter}
        parm3   : longint;         {** 3rd parameter}
        parm4   : longint;         {** 4th parameter}
    var status  : longint          {** result of the operation}
); external;
```

physop - Perform physical device operation.

Description:

Performs a direct call to the device driver of the named device bypassing the file structure. Allows physical I/O on mounted devices. The device may be mounted as a 'special' (e.g. diskspc).

To successfully perform the operation, the calling process must have read or write privilege to the device (depending on the operation) and either be the owner of the device (the process uic and the device uic are the same) or have readphys or writephys privilege (depending on the operation).

For the nondev and nondevspc class devices, the readphys and writephys privilege are not required.

Related Privileges:

none	- Allows the process to access the device if the owner id and group id (uic) of the process are the same as the uic of the device and the process has read/write privilege as described above. Or the class is nondev or nondevspc.
altuic	- Allows the process to access the device if the owner of the image file for the current process has access to the device as described above.
bypass	- Allows the process to access the device independent of the file protection. This does not obviate the need for either readphys or writephys privilege. This only applies to read or write privilege to the device.
readphys	- Allows physical access for read operations to devices as described above. This does not obviate the need for either read or write privilege to the device.
system	- Allows the process to access the device if the system has access to the device as described above. This does not obviate the need for either readphys or writephys privilege. This only applies to read or write privilege to the device.
writephys	- Allows physical access for write operations to devices as described above. This does not obviate the need for either read or write privilege to the device.

Parameters:

dname - Address of null terminated string containing the name of the device involved. This string is translated automatically by the MCS to its logical equivalent. This string may contain up to 93 significant characters followed by a null. If this string contains a file designation, the devicename portion of the file designation is used for this parameter.

func - Which operation to perform. Valid operations are: (Note that some of the commands are device class specific. When ever a class is specified, it also applies to the special form of that class. Commands described for tty class devices also apply to pipe, sync, and nondev class devices.)
The names of these functions are defined in the file /sysincl.sys/contcmd.*

Read from the device
(2) diskreadcmd
(2) tapereadcmd
(2) ttyreadcmd

This will read the specified number of blocks from the given device. This command is valid on this list of devices: Disk, Nondev, Pipe, Sync, Tape, TTY.

Requires read privilege to the device, and if the calling process is not the owner of the device, also requires readphys privilege, unless this is a nondev class device.

parm1 - Address of a buffer to receive the data read. This buffer must be word aligned.
parm2 - A long word containing the block number of the first block to be read. This parameter is not used for tape or tty class devices.
parm3 - A long word containing the number of blocks to read. On tape devices, this parameter represents how many bytes to read. On tapes, this function will never read more than one block.
parm4 - Address of a long word to receive the number of blocks actually read.

Write to the device
(3) diskwritecmd
(3) tapewritecmd
(3) ttywritecmd

This will write the specified number of blocks to the given device. This command is valid on this list of devices: Disk, Nondev, Pipe, Sync, Tape, TTY.

Requires write privilege to the device, and if the calling process is not the owner of the device, also requires writephys privilege, unless this is a nondev class device.

```
  parml - Address of a buffer containing data
          to be written. This buffer must be
          word aligned.
  parm2 - A long word containing the block
          number of the first block to be
          written. This parameter is not used
          for tape or tty class devices.
  parm3 - A long word containing
          the number of blocks to write.
          On tape devices, this parameter
          represents the number of bytes to
          write.
  parm4 - Address of a long word to receive
          the number of blocks actually written.
```

Format the device
(4) diskformatcmd

Reformat the given media. This command is valid on this list of devices: Disk.

Requires write privilege to the device, and if the calling process is not the owner of the device, also requires writephys privilege.

```
  parml - Not used.
  parm2 - Not used.
  parm3 - Not used.
  parm4 - Not used.
```

Erase the device
(4) tapeerasecmd

Erase the data off of the given device. This command is valid on this list of devices: Tape.

Requires write privilege to the device, and if the calling process is not the owner of the device, also requires writephys privilege.

```
  parml - A long word containing subfunction
          number:
          (0) tapeerstartvar - Variable length
```

erase start.
(1) tapeerstopvar - Variable length erase stop.
(2) tapeersecurity - Security erase.
(3) tapeerfixedlen - Fixed length erase.

parm2 - Not used.
parm3 - Not used.
parm4 - Not used.

Purge input buffer (Not implemented)
(4) ttypurgeinputbuffer

This will delete all data in the input typeahead buffer. This command is valid on this list of devices: Nondev, Pipe, Sync, TTY.

Requires write privilege to the device, and if the calling process is not the owner of the device, also requires writephys privilege, unless this is a nondev class device.

parm1 - Not used.
parm2 - Not used.
parm3 - Not used.
parm4 - Not used.

Read device status
(5) diskgetstatuscmd
(5) tapegetstatuscmd
(5) ttygetstatuscmd
(5) quegetstatuscmd

This will read the device status from the given device. This command is valid on this list of devices: Disk, Nondev, Pipe, Queue, Sync, Tape, TTY.

Requires read privilege to the device, and if the calling process is not the owner of the device, also requires readphys privilege, unless this is a nondev class device.

parm1 - Address of a 128 byte buffer to receive the device status. This buffer must be word aligned.
parm2 - Not used.
parm3 - Not used.
parm4 - Not used.

Set device status
(6) disksetstatuscmd
(6) tapesetstatuscmd
(6) ttysetstatuscmd

(6) quesetstatuscmd

This will set the device status on the given device. This command is valid on this list of devices: Disk, Nondev, Pipe, Queue, Sync, Tape, TTY.

Requires write privilege to the device, and if the calling process is not the owner of the device, also requires writephys privilege, unless this is a nondev class device.

parm1 - Address of a 128 byte buffer containing the new device status. This buffer must be word aligned.
parm2 - Not used.
parm3 - Not used.
parm4 - Not used.

Format specified track(s) (Not implemented)
(7) diskformattrackcmd

This will format a given list of tracks on the device. This command is valid on this list of devices: Disk.

Requires write privilege to the device, and if the calling process is not the owner of the device, also requires writephys privilege.

parm1 - A long word containing the cylinder number.
parm2 - A long word containing the head number.
parm3 - A long word containing the number of cylinders to format.
parm4 - Address of a long word to receive the number of cylinders actually formatted.

Skip, position the device.
(7) tapeskipcmd

Skip to the specified position on the device. This command is valid on this list of devices: Tape.

Requires read privilege to the device, and if the calling process is not the owner of the device, also requires readphys privilege.

parm1 - A long word containing subfunction number (type of skip):
(0) tapeskiprecords - Skip records
(1) tapeskipfiles - Skip files

```
                        (2) tapeskipbot - Skip to beginning of
                            volume
                        (3) tapeskipeot - Skip to end of volume
                  parm2 - Not used.
                  parm3 - A long word containing
                          the number of units to skip.
                          If parm1 specifies a skip to beginning
                          of volume then this parameter indicates
                          whether the tape should be positioned
                          before or after the volume label.
                          (0) tapeskipbefheader - Leave tape
                              positioned before the volume label.
                              (load point)
                          (1) tapeskipaftheader - Leave tape
                              positioned after the volume label.
                              (the position the tape would
                              normally be at after a mount.
                  parm4 - Address of a long word to receive the
                          number of units actually skipped.
```

Dial a modem
(7) ttydialcmd

With the given string this will dial out on the given device. This command is valid on this list of devices: Nondev, Pipe, Sync, TTY.

Requires write privilege to the device, and if the calling process is not the owner of the device, also requires writephys privilege, unless this is a nondev class device.

```
  parm1 - Address of a dial buffer. Contains
          characters meaningful to the device.
          Representing the number to be dialed.
  parm2 - Not used.
  parm3 - The number of bytes to be used in the
          dial buffer.
  parm4 - Not used.
```

Insert an entry into the device
(7) queenquecmd

This will insert a given create process record into the queue for execution at the correct time. This command is valid on this list of devices: Queue.

Requires write privilege to the device.

```
  parm1 - Address of the queue create process
          buffer.
  parm2 - Address of the queue entry buffer.
```

parm3 - Not used.
parm4 - Address of a longword to receive the queue entry number.

Set drive configuration table
(8) disksetdrivetblcmd

This will define a new drive configuration table for this device. This describes the media to the driver. This command is valid on this list of devices: Disk.

Requires write privilege to the device, and if the calling process is not the owner of the device, also requires writephys privilege.
parm1 - Address of the new drive configuration table.
parm2 - Not used.
parm3 - Not used.
parm4 - Not used.

Write tape mark
(8) tapewritetapemark

Write a tape mark on the given device. This command is valid on this list of devices: Tape.

Requires write privilege to the device, and if the calling process is not the owner of the device, also requires writephys privilege.
parm1 - Not used.
parm2 - Not used.
parm3 - Not used.
parm4 - Not used.

Hangup a modem
(8) ttyhangupcmd

This will send a hangup command out to the given device. This command is valid on this list of devices: Nondev, Pipe, Sync, TTY.

Requires write privilege to the device, and if the calling process is not the owner of the device, also requires writephys privilege, unless this is a nondev class device.
parm1 - Not used.
parm2 - Not used.
parm3 - Not used.
parm4 - Not used.

Get entry information by index number
(8) quelistcmd

This will receive all of the information about a given entry in the queue. It will access that entry by the current index from the front of the queue. This command is valid on this list of devices: Queue.

Requires read privilege to the device.
- parml - Index number from front of file for which entry we want.
- parm2 - Address of buffer to receive the queue create process buffer.
- parm3 - Address of buffer to receive the queue entry buffer.
- parm4 - Address of a longword to receive this entry's entry number.

Set [CONTROL] C pid
(9) ttysetcontcpid

This will specify that the calling process is to be terminated on the next [CONTROL] C character that is received. This command is valid on this list of devices: Nondev, Pipe, Sync, TTY.

Requires write privilege to the device, and if the calling process is not the owner of the device, also requires writephys privilege, unless this is a nondev class device.
- parml - Not used.
- parm2 - Not used.
- parm3 - Not used.
- parm4 - Not used.

Delete an entry from the device
(9) quedequecmd

This will delete the given entry from the queue. This command is valid on this list of devices: Queue.

Requires write privilege to the device. Requires that the entry have the same owner id and group id as the caller, or the same group id as the caller and the caller has group privilege, or the caller has world privilege.

- parml - Entry number of which entry to delete.

parm2 - Not used.
parm3 - Not used.
parm4 - Not used.

Reset the device
(10) ttyresetcmd

This will reset the device. This command is valid on this list of devices: Nondev, Pipe, Sync, TTY.

Requires write privilege to the device, and if the calling process is not the owner of the device, also requires writephys privilege, unless this is a nondev class device.
parm1 - Not used.
parm2 - Not used.
parm3 - Not used.
parm4 - Not used.

Halt the device
(10) quehaltcmd

This will halt the given queue. This means that no more entries will be executed. Entries can still be added to the queue. This command is valid on this list of devices: Queue.

Requires write privilege to the device, and if the calling process is not the owner of the device, also requires operator privilege.
parm1 - Not used.
parm2 - Not used.
parm3 - Not used.
parm4 - Not used.

Start the device after a halt
(11) questartcmd

This will start the queue after a halt command has been given. This means that ready entries will be executed. This command is valid on this list of devices: Queue.

Requires write privilege to the device, and if the calling process is not the owner of the device, also requires operator privilege.
parm1 - Not used.
parm2 - Not used.
parm3 - Not used.

parm4 - Not used.

Restart an entry on the device
(12) querestartcmd

This will take an entry that is already executing and will terminate the process. It will then restart the entry. This command is valid on this list of devices: Queue.

Requires write privilege to the device. Requires that the entry have the same owner id and group id as the caller, or the same group id as the caller and the caller has group privilege, or the caller has world privilege.
parm1 - Entry number of which entry to restart.
parm2 - Not used.
parm3 - Not used.
parm4 - Not used.

Send a break character
(13) ttysendbreak

This will send a break character out of the given device. This command is valid on this list of devices: Nondev, Pipe, Sync, TTY.

Requires write privilege to the device, and if the calling process is not the owner of the device, also requires writephys privilege, unless this is a nondev class device.
parm1 - Not used.
parm2 - Not used.
parm3 - Not used.
parm4 - Not used.

Wait for an entry to complete
(13) quewaitcmd

This will wait for the given entry to complete. If the entry does not exist, it will return immediatly. This command is valid on this list of devices: Queue

Requires read privilege to the device.
parm1 - Entry number of which entry to wait on.
parm2 - Not used.
parm3 - Not used.
parm4 - Not used.

Hold an entry on the device

(15) queholdcmd

This will hold the given entry. This means that it will not be executed. This command is valid on this list of devices: Queue.

Requires write privilege to the device. Requires that the entry have the same owner id and group id as the caller. or the same group id as the caller and the caller has group privilege, or the caller has world privilege.

parm1 - Entry number of which entry to hold.
parm2 - Not used.
parm3 - Not used.
parm4 - Not used.

Wake an entry after a hold command
(16) quewakecmd

This will wake the given entry after a hold command. This means the entry is available for execution again. This command is valid on this list of devices: Queue.

Requires write privilege to the device. Requires that the entry have the same owner id and group id as the caller. or the same group id as the caller and the caller has group privilege, or the caller has world privilege.

parm1 - Entry number of which entry to wake.
parm2 - Not used.
parm3 - Not used.
parm4 - Not used.

Modify an entry on the device
(18) quemodifycmd

This will modify an entry that is already queued. This command is valid on this list of devices: Queue.

Requires write privilege to the device. Requires that the entry have the same owner id and group id as the caller. or the same group id as the caller and the caller has group privilege, or the caller has world privilege.

parm1 - Entry number of which entry to modify.
parm2 - Address of the new queue create process buffer.
parm3 - Address of the new queue entry buffer.
parm4 - Not used.

Close the given device
(19) queclosecmd

This will close the queue. This means that no more entries can be entered into the queue. Entries that have already been queued will continue to be executed as their turn arrives. This command is valid on this list of devices: Queue.

Requires write privilege to the device, and if the calling process is not the owner of the device, also requires operator privilege.

parm1 - Not used.
parm2 - Not used.
parm3 - Not used.
parm4 - Not used.

Open the given device after a close
(20) queopencmd

This will open the queue after a close command. This means that more entries may be queued. This command is valid on this list of devices: Queue.

Requires write privilege to the device, and if the calling process is not the owner of the device, also requires operator privilege.

parm1 - Not used.
parm2 - Not used.
parm3 - Not used.
parm4 - Not used.

Get entry information by entry number
(21) quegetentrycmd

This will receive all of the information about a given entry in the queue. It will access that entry by entry number. This command is valid on this list of devices: Queue.

Requires read privilege to the device.

parm1 - Entry number of which entry to get.
parm2 - Address of buffer to receive the queue create process buffer.
parm3 - Address of buffer to receive the queue entry buffer.
parm4 - Address of a longword to receive this entry's entry number.

Get default create process record
(22) quegetdefcrpcmd

This will get the default create process record. This record is used when a user redirects output directly to the queue device. This command is valid on this list of devices: Queue.

Requires read privilege to the device.
- parml - Address of buffer to receive the default queue create process record.
- parm2 - Not used.
- parm3 - Not used.
- parm4 - Not used.

Set default create process record
(23) quesetdefcrpcmd

This will set the default create process record. This record is used when a user redirects output directly to the queue device. This command is valid on this list of devices: Queue.

Requires write privilege to the device, and if the calling process is not the owner of the device, also requires operator privilege.
- parml - Address of the new default queue create process buffer.
- parm2 - Not used.
- parm3 - Not used.
- parm4 - Not used.

timout	- Maximum amount of time to wait for the operation to complete. Expressed in 100'ths of a second
parml	- A parameter defined by the function
parm2	- A parameter defined by the function
parm3	- A parameter defined by the function
parm4	- A parameter defined by the function
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
errtimeout	(128)	A request was not completed within the specified time.
errinvdevnam	(130)	The specified devicename is syntactically incorrect.
errundevnam	(131)	The MCS does not recognize the devicename.

		Is the device mounted?
errinvcloper	(173)	The operation is inappropriate for the device class.
errprevinit	(188)	The specified device is already mounted, and has another name.
errinvskpcmd	(206)	The specified skip or erase tape-function is undefined.
errinvdrvnum	(311)	A value in at least one field of the devicename is disallowed.
		Device errors

See Also:

```
    _dismnt - Dismount a logical device
    _getdnam- Get device name
    _getdst - Get device status
    _giodst - Get device status with lun
    _mount  - Mount a logical device
    _physio - Perform physical I/O operation
    _setdst - Set device status
    _siodst - Set device status with lun
    _skip   - Position tape
```

Assembler Calling Sequence:

```
%%sys$disk/sysincl.sys/sysequ.asm
    push    dname                       ;address - device name
    push    func                        ;value - which function
    push    timout                      ;value - time out
    push    parm1                       ;address/value - 1st parameter
    push    parm2                       ;address/value - 2nd parameter
    push    parm3                       ;address/value - 3rd parameter
    push    parm4                       ;address/value - 4th parameter
    push    status                      ;address - result of the operation
    jsr     _physop                     ;perform physical device operation
```

C function declaration:

```
    #include "sys$disk/sysincl.sys/sysequ.h"
                                    /* perform physical device operation */
    long                            /* returns result of the operation */
    _physop(dname. func, timout, parm1, parm2, parm3, parm4)
            char dname[94];         /* device name */
            long func;              /* which function */
            long timout;            /* time out */
            long parm1;             /* 1st parameter */
            long parm2;             /* 2nd parameter */
            long parm3;             /* 3rd parameter */
            long parm4;             /* 4th parameter */
```

Fortran Subroutine Declaration:

```
c                                      ! perform physical device operation
        subroutine physop(dname, func, timout, parm1, parm2,
     &                    parm3, parm4, status)
            character*94 dname     ! device name
            integer*4 func         ! which function
            integer*4 timout       ! time out
            integer*4 parm1        ! 1st parameter
            integer*4 parm2        ! 2nd parameter
            integer*4 parm3        ! 3rd parameter
            integer*4 parm4        ! 4th parameter
            integer*4 status       ! result of the operation
```

Pascal Procedure Declaration:

Note that all of the parm components are defined as long integers. Where the address of a variable is to be passed, use the following function. _physop(...,cast(vloc(variable),longint),...)

```
%%sys$disk/sysincl.sys/sysequ.pas
procedure  physop(                  {** perform physical device operation}
        dname   : string[93];       {** device name}
        func    : longint;          {** which function}
        timout  : longint;          {** timeout value}
        parm1   : longint;          {** 1st parameter}
        parm2   : longint;          {** 2nd parameter}
        parm3   : longint;          {** 3rd parameter}
        parm4   : longint;          {** 4th parameter}
    var status  : longint           {** result of the operation}
); external;
```

Return a list of all known process ID numbers on the system.

Description:

Return a list of process ID numbers for all processes running on the system.

Related Privileges:

None.

Parameters:

siteid	- The site ID for which the list of process IDs is being requested.
pidlst	- Address of buffer to receive the process IDs known about in the system. This buffer must be word aligned.
len	- Maximum number of process IDs that can be contained in the pidlst buffer.
retlen	- Address of a long word to receive the number of process IDs that were written into pidlst.
total	- Address of a long word to receive the total number of processes running on the system. This number may be greater than the number returned in retlen.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

erroddbufaddr	(3)	The process's buffer does not begin on a word boundary.
errinvsiteid	(8)	The specified site ID does not exist.

See Also:

_prclst - Get process IDs on a priority level.

Assembler Calling Sequence:

```
push    siteid                  ;value - system id
push    pidlst                  ;address - process id buffer
push    len                     ;value - length of pidlst
push    retlen                  ;address - number of PIDs returned
push    total                   ;address - total number of processes
push    status                  ;address - result of the operation
jsr     _pidlst                 ;get list of process ids
```

C Function Declaration:

```
                                /* get list of process ids */
long                            /* returns result of the operation */
_pidlst(siteid, pidlst, len, retlen, total, status)
        long siteid;            /* system id */
        long *pidlst;           /* PID buffer */
        long len;               /* length of pidlst */
        long *retlen;           /* number of PIDs returned */
        long *total;            /* total number of process ids */
```

FORTRAN Subroutine Declaration:

```
c                               ! get list of process ids
        subroutine _pidlst(siteid, pidlst, len, retlen, total,
                  status)
            integer*4 siteid    ! system id
            integer*4 pidlst    ! PID buffer
            integer*4 len       ! length of pidlst
            integer*4 retlen    ! number of PIDs returned
            integer*4 total     ! total number of process ids
            integer*4 status    ! result of the operation
```

Pascal Procedure Declaration:

```
procedure _pidlst(              {** get list of process ids
        siteid  : longint;      {** system id}
        pidlst  : ^array_of_char; {** PID buffer}
        len     : longint;      {** length of pidlst}
    var retlen  : longint;      {** number of PIDs returned}
    var total   : longint       {** total number of process ids}
    var status  : longint       {** result of the operation}
); external;
```

Get PIDs on a priority level.

Description:

Given a priority level this call returns a list of process ids (pids) of the processes assigned to that priority.

Related Privileges:

None.

Parameters:

siteid	- A long word containing the system id of the system whose process list is being requested. A siteid of zero corresponds to the system on which the calling process is executing.
priort	- Desired priority level. If it is not in the range of valid priorities (0..3) it is used modulo 4.
pidlst	- Address of buffer to receive the process id's of processes at the above priority level. This buffer must be word aligned.
len	- Maximum number of process id's that can be contained in the pidlst buffer
retlen	- Address of a long word to receive the number of pid's that were written into pidlst. If an error is encountered, the retlen will be set to 0.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

erroddbufaddr	(3)	The process's buffer does not begin on a word boundary.
errinvsiteid	(8)	The specified site id does not exist.

See Also:

_gengy	- Get pid of ancestor process
_getpid	- Get process id (pid) from name
_getpnam	- Get process name from pid

Assembler Calling Sequence:

```
push    siteid                  ;value - system id
push    priort                  ;value - priority level
push    pidlst                  ;address - pid buffer
push    len                     ;value - length of pidlst
```

```
    push    retlen                      ;address - number of pid's returned
    push    status                      ;address - result of the operation
    jsr     _prclst                     ;get pids on a priority level
```

C function declaration:

```
                                        /* get pids on a priority level */
    long                                /* returns result of the operation */
    _prclst(siteid, priort, pidlst, len, retlen)
            long siteid;                /* system id */
            long priort;                /* priority level */
            long *pidlst;               /* pid buffer */
            long len;                   /* length of pidlst */
            long *retlen;               /* number of pids returned */
```

Fortran Subroutine Declaration:

```
    c                                   ! get pids on a priority level
            subroutine prclst(siteid, priort, pidlst, len, retlen, status)
                integer*4 siteid        ! system id
                integer*4 priort        ! priority level
                integer*4 pidlst        ! pid buffer
                integer*4 len           ! length of pidlst
                integer*4 retlen        ! number of pids returned
                integer*4 status        ! result of the operation
```

Pascal Procedure Declaration:

```
    procedure _prclst(                  {** get pids on a priority level}
            siteid  : longint;          {** system id}
            priort  : longint;          {** priority level}
            pidlst  : ^array_of_char;   {** pid buffer}
            len     : longint;          {** length of pidlst}
        var retlen  : longint;          {** number of pid's returned}
        var status  : longint           {** result of the operation}
    ); external;
```

Set priority scheduling ratio.

Description:

This system call allows the process with operator privilege to set the priority level refresh counts for the scheduler. By default the refresh counts are set to 10.

For each process executing at priority 1, ratio[1] processes will execute at priority 0.

For each process executing at priority 2, ratio[2] processes will execute at priority 1.

For each process executing at priority 3, ratio[3] processes will execute at priority 2.

For each process executing at priority 4, ratio[4] processes will execute at priority 3.

For each process executing at priority 5, ratio[5] processes will execute at priority 4.

For each process executing at priority 6, ratio[6] processes will execute at priority 5.

For each process executing at priority 7, ratio[7] processes will execute at priority 6.

For each process executing at priority 8, ratio[8] processes will execute at priority 7.

For each process executing at priority 9, ratio[9] processes will execute at priority 8.

For each process executing at priority 10, ratio[10] processes will execute at priority 9.

For each process executing at priority 11, ratio[11] processes will execute at priority 10.

For each process executing at priority 12, ratio[12] processes will execute at priority 11.

For each process executing at priority 13, ratio[13] processes will execute at priority 12.

For each process executing at priority 14, ratio[14] processes will execute at priority 13.

For each process executing at priority 15, ratio[15] processes will execute at priority 14.

Related Privileges:

none	- Process not allowed to set scheduling ratio
operator	- Allows process to set scheduling ratio

Parameters:

siteid	- A long word containing the system id of the system whose priority ratio is to be set. A siteid of zero corresponds to the system on which the calling process is executing.
ratio	- Address of an array of 15 integers (16 bit words) containing the scheduling ratios for each priority level. This array must be word aligned. Each of the 15 integers may contain a value between 1 and 32767. Negative values are prohibited.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
errinvsiteid	(8)	The specified site id does not exist.
errpriorratio	(58)	The priority ratio for the scheduler is less than or equal to zero.

See Also:

_setpri - Change process's priority

Assembler Calling Sequence:

```
push    siteid                  ;value - system id
push    ratio                   ;address - array of ratios
push    status                  ;address - result of the operation
jsr     _prirat                 ;set priority scheduling ratio
```

C Function Declaration:

```
                                /* set priority scheduling ratio */
long                            /* returns result of the operation */
_prirat(siteid, ratio)
        long siteid;            /* system id */
        int ratio[15];          /* array of ratios
```

FORTRAN Subroutine Declaration:

```
c                               ! set priority scheduling ratio
        subroutine _prirat(siteid, ratio, status)
            integer*4 siteid    ! system id
            integer*2(15) ratio ! array of ratios
            integer*4 status    ! result of the operation
```

Pascal Procedure Declaration:

```
procedure _prirat(              {** set priority scheduling ratio}
       siteid  : longint;       {** system id}
       ratio   : array[1..15] of integer; {** array of ratios}
   var status  : longint        {** result of the operation}
); external;
```

Change memory page protection.

Description:

Modify the write protection on a specified page of logical memory owned by the current process.

Related Privileges:

none	- Allows modification of memory protection on any owned page of memory assigned to the calling process. Note that shareable pages are not owned by any process.
writephys	- Allows modification of memory protection on any page of memory assigned to the calling process.

Parameters:

adr	- Address of the page of logical memory which is to be protected or unprotected. This address must be on a 4K byte boundary.
prot	- protection. 0 indicates that the page is not to be write protected. 1 indicates that the page is to be write protected. Other values are reserved and should not be used.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
errinvadr	(4)	The logical address, for the memory requested, is invalid.
errnonowned	(6)	The process tried to affect a page in memory it did not own.

See Also:

_allmem - Allocate dynamic memory
_fremem - Deallocate a page of memory

Assembler Calling Sequence:

```
push    adr                     ;value - address of memory page
push    prot                    ;value - protection value
push    status                  ;address - result of the operation
jsr     _protmem                ;change memory page protection
```

C function declaration:

```
                                        /* change memory page protection */
    long                                /* returns result of the operation */
    _protmem(adr, prot)
            long adr;                   /* address of memory page */
            long prot;                  /* protection value */
```

Fortran Subroutine Declaration:

```
    c                                   ! change memory page protection
            subroutine protme(adr, prot, status)
                integer*4 adr           ! address of memory page
                integer*4 prot          ! protection value
                integer*4 status        ! result of the operation
```

Pascal Procedure Declaration:

```
    procedure _protmem(                 {** change memory page protection}
            adr     : longint;          {** address of memory page}
            prot    : longint;          {** protection value}
        var status  : longint           {** result of the operation}
    ); external;
```

Read physical memory.

Description:

By default a process can access any memory that is part of its own logical address space ($000000 through $1FFFFF) To access memory above two megabytes, the process must either change to supervisor mode of operation or use this call asking MCS to read the memory for it.

Using _rdpmem to read physical memory has the additional property that when memory errors (e.g. attempt to access non-existant memory) occur, they are reported to the process through the status variable and are not considered fatal errors.

A process must have readphys privilege to read addresses in physical memory.

Related Privileges:

none	- Process not allowed to read physical memory
readphys	- Allows process to read physical memory

Parameters:

siteid	- A long word containing the system id of the system whose physical memory is to be read. A siteid of zero corresponds to the system on which the calling process is executing.
adr	- Address in physical memory to be read.
mode	- Specifies whether to use byte, word or long word transfers from the specified address. 0 indicates byte, 1 indicates word and 2 indicates long word transfers. All other values are reserved and should not be used.
buf	- Address of the buffer to receive the data read from the specified address.
nrec	- The number of units (bytes, words or long words) to be transferred.
trnsfr	- Address of a long word to receive the number of units actually transferred.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv (1) The process lacks the privileges required to

```
                           perform the operation.
   erroddbufaddr     (3)   The process's buffer does not begin on a word
                           boundary.
   errbustrap       (37)   The process has a bus error.
   errnonexmem      (39)   The process attempted to access nonexistent memory.
   errmemparity     (40)   The process has a memory parity-error.
```

See Also:

```
    _chsuper - Change to supervisor mode
    _wtpmem  - Write physical memory
```

Assembler Calling Sequence:

```
    push    siteid                      ;value - system id
    push    adr                         ;value - address to be read
    push    mode                        ;value - byte, word, long word moves
    push    buf                         ;address - user buffer
    push    nrec                        ;value - number of units to read
    push    trnsfr                      ;address - num of units transferred
    push    status                      ;address - result of the operation
    jsr     _rdpmem                     ;read physical memory
```

C function declaration:

```
                                        /* read physical memory */
    long                                /* returns result of the operation */
    _rdpmem(siteid, adr, mode, buf, nrec, trnsfr)
            long siteid;                /* system id */
            long adr;                   /* address to be read */
            long mode;                  /* byte, word, long word moves */
            char *buf;                  /* user buffer */
            long nrec;                  /* number of units to read */
            long *trnsfr;               /* num of units transferred */
```

Fortran Subroutine Declaration:

```
    c                                   ! read physical memory
            subroutine rdpmem(siteid, adr, mode, buf, nrec, trnsfr, status)
                integer*4 siteid        ! system id
                integer*4 adr           ! address to be read
                integer*4 mode          ! byte, word, long word moves
                character*(*) buf       ! user buffer
                integer*4 nrec          ! number of units to read
                integer*4 trnsfr        ! num of units transferred
                integer*4 status        ! result of the operation
```

Pascal Procedure Declaration:

```
    procedure _rdpmem(                  {** read physical memory}
            siteid  : longint;          {** system id}
```

```
          adr     : longint;        {** address to be read}
          mode    : longint;        {** byte, word, long word moves}
          buf     : ^array_of_char; {** user buffer}
          nrec    : longint;        {** number of units to read}
      var trnsfr  : longint;        {** num of units transferred}
      var status  : longint         {** result of the operation}
  ); external;
```

read - Read from an open file.

Description:

Given a valid logical unit number (lun) of a file open for read access, transfers one or more records from the file to the process's buffer.

On successful completion, returns the number of complete records actually transferred. This number may be less than the number requested if:

1) logical end of file was reached.
2) a timeout occurs.
3) a device error occurs.
4) the end of a line was reached while reading with edit mode 2

Related Privileges:

None.

Parameters:

lun	- logical unit number (lun) of the file to be read.
recnum	- The record number to be read. 0 means the first record in a file. A recnum of $FFFFFFFF (-1) represents the next sequential record. Note that recnum is an unsigned long word.
edmode	- The edit mode to use. This parameter is divided into two 16 bit fields. The least significant word represents which edit mode processor to use. The most significant word contains edit mode flags for the processor.

An input edit mode processor is used to filter the input stream before it is passed to the process. The the following transformations are defined:

Name	Edmode	Description
emvreadraw	0	Raw data. No alterations of data.
	1	Reserved.
emvreadln	2	For tty class devices, read line. Returns to the user process all characters from the specified position up to and including the line terminator. For tty, pipe, sync and nondev class

devices, all carriage returns (13) are converted to line feeds (10). For other device classes the line terminator is unmodified.

Control does not return to the calling process until a line terminator is entered or a timeout occurs. Valid line terminators are line feed (10), vertical tab (11), form feed (12), or carriage return (13). Processing delete characters (127) (prints a backspace, space, backspace and removes the last character from the buffer) is handled automatically.

If the length of the line actually read exceeds the nrecs parameter, (the number of characters to be read), then nrecs number of characters are returned and the status parameter returns a warning.

If the record length on the file being read is not one (1), an error is returned, and no data is transferred.

For other classes (disk, diskspc, tape, tapespc, pipe, pipespc, sync, syncspc, network, networkspc, ttyspc, tape$1, tape$1spc) this edit mode is undefined and functions the same as edit mode 0.

emvreadlnwchr 3 For tty class devices, read character. Returns to the user process one or more bytes from the specified position. All line terminators are mapped to a line feed (10).

Control does not return to the calling process until a line terminator is entered or a timeout occurs. Valid line terminators are line feed (10), vertical tab (11), form feed (12), or carriage return (13). Processing delete characters (127) (prints a backspace, space, backspace and removes the last character from the buffer) is handled automatically. When the line terminator is entered, the first

		character on the line is returned to the user process. Subsequent calls using edit mode 3 get subsequent characters on the line up to and including the mapped line terminator. The nrecs parameter specifies how many characters are to be returned. If the record length on the file being read is not one (1), an error is returned, and no data is transferred. For other classes (pipe, pipespc, sync, syncspc, network, networkspc, ttyspc, tape, tapespc, disk, diskspc, tape$1, tape$1spc) this edit mode is undefined and functions the same as edit mode 0.
emvreadlnall	4	Read line. Returns to the user process all characters from the specified position up to and including the line terminator. For tty, pipe, sync and nondev class devices, all carriage returns (13) are transformed to line feeds (10). For other device classes the line terminator is unmodified. For tty class devices, this edit mode works the same as edit mode 2. Control does not return to the calling process until a line terminator is entered or a timeout occurs. Valid line terminators are line feed (10), vertical tab (11), form feed (12), or carriage return (13). Processing delete characters (127) (prints a backspace, space, backspace and removes the last character from the buffer) is handled automatically. For disk, tape and pipe class devices, the line terminator is a line feed (10). If the length of the line actually read exceeds the nrecs parameter, (the number of characters to be read), then nrecs number of characters are returned and the status parameter returns a warning. If the record length on the file

being read is not one (1), an error is returned, and no data is transferred.

For other classes (diskspc, tapespc, pipespc, sync, syncspc, network, networkspc, ttyspc, tape$1, tape$1spc) this edit mode is undefined and functions the same as edit mode 0.

emvreadnwchall 5 Read character. Returns to the user process one or more bytes from the specified position. All line terminators are mapped to a line feed (10).

For tty class devices this edit mode is the same as edit mode 3. Control does not return to the calling process until a line terminator is entered or a timeout occurs. Valid line terminators are line feed (10), vertical tab (11), form feed (12), or carriage return (13). Processing delete characters (127) (prints a backspace, space, backspace and removes the last character from the buffer) is handled automatically.

For pipe, disk and tape class devices, the line terminator is line feed (10). The specified number of characters (up to the line terminator) are returned to the user process.

When the line terminator is entered, or encountered, the first character on the line is returned to the user process. Subsequent calls using edit mode 5 get subsequent characters on the line up to and including the mapped line terminator. The nrecs parameter specifies how many characters are to be returned.

If the record length on the file being read is not one (1), an error is returned, and no data is transferred.

For other classes (pipespc, sync, syncspc, network, networkspc, ttyspc, tapespc, diskspc, tape$1, tape$1spc) this edit mode is undefined and functions the same as edit mode 0.

The most significant word of the edmode parameter contains the following bit flags. If the bit is a one (1) the mode is as described here.

Bit Name	Bit #	Description
emnoecho	16	No echo - As characters come in on tty class devices, they are not echoed back.
emspcompact	17	Space decompaction - On sync class devices compacted spaces are automatically expanded.
emnofastread	18	No fast read - On disk class devices, this bit allows a process to do a 'normal' read on a file which was opened for fast reads.
emnoverifyrd	19	No verify read ok - Will return input data on sync class reads when input buffer becomes more than half full even though the data has not been verified. If the data turns out to be bad, an error is returned on the subsequent read.
	20	Reserved. Must be set to zero
emlockunlock	21	Read and lock - On disk class devices, this bit will cause all of the records requested to be read be locked.
	22-31	Reserved. Must be set to zero (0).

timout - The wait count is in 100'ths of a second and represents the amount of time to wait for the read to complete before returning to the user process. If a timeout occurs, the data received up to that point will be returned to the process.

buf - Address of the process's buffer into which the data will be read. May be on a word or a byte boundary unless the file was open for fast read in which case it must be on a word boundary. Also, if the file is to be read with a fast read, the entire buffer must exist on the same four kilobyte page of memory.

nrecs - Number of records to read. This parameter is an unsigned long word. If it is zero, no data is transferred.

trnsfr - Address of a long word to receive the number of records actually read.

status - Address of a long word to receive the result of the operation.

Diagnostics:

errtimeout	(128)	A request was not completed within the specified time.
errinvlfn	(132)	The logical unit number does not correspond to an open file.
errreqtolrg	(137)	The request is too large for the system to handle.
errreadleof	(140)	The process tried to read past the logical end of a file.
errnoreadacc	(141)	The process does not have read-access to the file.
errinvreadreq	(165)	The read request is invalid.
errpagebndry	(166)	The request crosses a physical page boundary in memory.
errinveditmd	(189)	The MCS does not recognize the specified edit mode.
errbadpos	(197)	The process tried to access a record (on a tape) out of sequence.
errdeadlock	(234)	The specified record cannot be locked without causing a deadlock.
errreclocked	(235)	The specified record(s) are locked by another process.
errlockint	(254)	(MCS error) A discrepency in the Record Locking code has been detected.
		Device integrity errors

See Also:

```
_bscbid - Bid or wait for bid
_bscpol - Multipoint bisync line control
_create - Create a file
_frdwait- Wait for fast read to complete
_getpos - Get the current file position
_lock   - Lock records within an open file
_open   - Open a file
_setpos - Set the current file position
_unlock - Unlock records within an open file
_write  - Write to an open file
```

Assembler Calling Sequence:

```
push    lun                     ;value - logical unit number
push    recnum                  ;value - record number
push    edmode                  ;value - edit mode
push    timout                  ;value - time out
push    buf                     ;address - user buffer
push    nrecs                   ;value - number of records to read
push    trnsfr                  ;address - number actually read
push    status                  ;address - result of the operation
jsr     _read                   ;read from an open file
```

C function declaration:

```
                                    /* read from an open file */
    long                            /* returns result of the operation */
    _read (lun,recnum,edmode,timout,buf,nrecs,trnsfr)
            long lun;               /* logical unit number */
            long recnum;            /* record number */
            long edmode;            /* edit mode */
            long timout;            /* time out */
            char *buf;              /* user buffer */
            long nrecs;             /* number of records to read */
            long *trnsfr;           /* number actually read */
```

Fortran Subroutine Declaration:

```
    c                               ! read from an open file
            subroutine read(lun, recnum, edmode, timout, buf,
         &          nrecs, trnsfr. status)
                integer*4 lun       ! logical unit number
                integer*4 recnum    ! record number
                integer*4 edmode    ! edit mode
                integer*4 timout    ! time out
                character*(*) buf   ! user buffer
                integer*4 nrecs     ! number of records to read
                integer*4 trnsfr    ! number actually read
                integer*4 status    ! result of the operation
```

Pascal Procedure Declaration:

```
    procedure  read(                {** read from an open file}
            lun     : longint;      {** logical unit number}
            recnum  : longint;      {** record number}
            edmode  : longint;      {** edit mode}
            timout  : longint;      {** time out value}
            buf     : ^array of_char; {** user buffer}
            nrecs   : longint;      {** number of records to read}
        var trnsfr  : longint;      {** number of records read}
        var status  : longint       {** result of the operation}
    ); external;
```

Rename a file.

Description:

Given a file name, locates the file and give it a new name. Both file names are logically translated before being used. Can be used to rename a file into another directory. Files cannot be renamed to other devices.

The new file is identical to the old file in every way (owner, protection, record length, etc) except for the new name.

A directory file on disk devices can be renamed to any place in the directory hierarchy except as a subdirectory of itself.

Unless the process has bypass privilege, it must have read and write privilege to the device containing the file, execute privilege to all directories in both the original and new paths leading to the file, read and write privilege to the directory containing the file, read and write privilege to the new directory to contain the file and read and write privilege to the file itself in order for the file to be successfully renamed.

If the original file name is specified by fcb.seq number the process must have read and write privilege to the device containing the file, execute privilege to all directories in the new path leading to the file, read and write privilege to the both the directory containing the file and the new directory to contain the file and read and write privilege to the file itself.

Related Privileges:

None	- Allows renaming if process has access to the file as described above.
altuic	- Allows renaming if the owner of image file for the current process has access to the file as described above.
bypass	- Allows the process to rename the file independent of the file protection.
system	- Allows renaming if the system has access to the file as described above.

Parameters:

fname	- Address of null terminated string containing the file name of an existing file to be renamed.

This string will be translated automatically by the MCS to its logical equivalent. This string may contain up to 93 valid characters followed by a null.

newnam - Address of null terminated string containing the new file name to be given to the file. This string will be translated automatically by the MCS to its logical equivalent. This string may contain up to 93 valid characters followed by a null.

status - Address of a long word to receive the result of the operation.

Diagnostics:

errnomemavail	(7)	All available memory has been allocated.
errinvvernum	(129)	A file's version number cannot be greater than 65535.
errinvdevnam	(130)	The specified devicename is syntactically incorrect.
errundevnam	(131)	The MCS does not recognize the devicename. Is the device mounted?
errfilnotfnd	(133)	The specified file could not be found.
errfilexists	(134)	The specified version of the file already exists
errnoexecpriv	(143)	The process does not have Execute Privilege for the file.
errnoreadpriv	(144)	The process does not have Read Privilege for the file.
errnowritepriv	(145)	The process does not have Write Privilege for the file.
errinvfnstr	(147)	The specified filename is syntactically incorrect.
errinvdirfle	(148)	The specified directory is not a directory.
errinvdirstr	(149)	The specified directory name is syntactically incorrect.
errrendiffdev	(167)	A file cannot be renamed to another device.
errinvcloper	(173)	The device class is inappropriate for the operation.
errdirnotfnd	(177)	The specified directory does not exist.
errinvdirren	(186)	The process tried to rename a directory as its own subdirectory.
		Device integrity errors

See Also:

none.

Assembler Calling Sequence:

```
push    fname                   ;address - original file name
push    newnam                  ;address - new file name
push    status                  ;address - result of the operation
```

```
    jsr     _rename                  ;rename a file
```

C function declaration:

```
                                     /* rename a file */
    long                             /* returns result of the operation */
    _rename(fname, newnam)
            char fname[94];          /* original file name */
            char newnam[94];         /* new file name */
```

Fortran Subroutine Declaration:

```
    c                                ! rename a file
            subroutine rename(fname, newnam, status)
                character*94 fname   ! original file name
                character*94 newnam  ! new file name
                integer*4 status     ! result of the operation
```

Pascal Procedure Declaration:

```
    procedure _rename(               {** rename a file}
            fname   : string[93];    {** original file name}
            newnam  : string[93];    {** new file name}
        var status  : longint        {** result of the operation}
    ); external;
```

Return a list of all known remote network ID numbers.

Description:

Return a list of network ID numbers and the total number of network ID numbers known in the network.

Related Privileges:

None.

Parameters:

rnidlst	- Address of buffer to receive the remote network IDs known about. This buffer must be word aligned.
len	- Maximum number of network IDs that can be contained in the rnidlst buffer.
retlen	- Address of a long word to receive the number of network IDs that were written into rnidlst.
total	- Address of a long word to receive the total number of network IDs known about in the system. This number may be greater than the number returned in retlen.

Diagnostics:

None.

See Also:

_getnnam - Get the name of a node
_getnsid - Get the site ID of a node
_rsidlst - Get list of site IDs from a remote network
_sidlst - Get list of site IDs on current network

Assembler Calling Sequence:

```
push    rnidlst                 ;address - rnid buffer
push    len                     ;value - length of rnid buffer
push    retlen                  ;address - number of rnids returned
push    total                   ;address - total number of rnids
jsr     _rnidlst                ;get list of remote network ids
```

C Function Declaration:

```
                                   /* get list of remote network ids */
void                               /* no result */
_rnidlst(rnidlst, len, retlen, total)
        long *rnidlst;             /* rnid buffer */
        long len;                  /* length of rnidlst */
        long *retlen;              /* number of rnids returned */
        long *total;               /* total number of rnids */
```

FORTRAN Subroutine Declaration:

```
c                                  ! get list of remote network ids
        subroutine rnidls(rnidlst, len, retlen, total)
            integer*4 rnidlst      ! rnid buffer
            integer*4 len          ! length of rnidlst buffer
            integer*4 retlen       ! number of rnids returned
            integer*4 total        ! total number of rnids
```

Pascal Procedure Declaration:

```
procedure _rnidlst(                {** get list of remote network ids}
        rnidlst : ^array_of_char; {** rnid buffer}
        len     : longint;         {** length of rnid buffer}
    var retlen  : longint;         {** number of rnids returned}
    var total   : longint          {** total number of rnids}
); external;
```

Return a list of all known site ID numbers for a remote network.

Description:

Return a list of site ID numbers and the total number of site ID numbers known for a given remote network.

Related Privileges:

None.

Parameters:

rnid	- The remote network ID for which the list of site IDs is being requested.
sidlst	- Address of buffer to receive the site IDs known about in the network. This buffer must be word aligned.
len	- Maximum number of site IDs that can be contained in the sidlst buffer
retlen	- Address of a long word to receive the number of site IDs that were written into sidlst.
total	- Address of a long word to receive the total number of site IDs known about in the system. This number may be greater that the number returned in retlen.

Diagnostics:

erroddbufaddr	(3)	The process's buffer does not begin on a word boundary.

See Also:

_getnnam - Get the name of a node
_getnsid - Get the site ID of a node
_rnidlst - Get list of remote network IDs
_sidlst - Get list of site IDs from current network

Assembler Calling Sequence:

```
push    rnid                      ;value - remote network id
push    sidlst                    ;address - siteid buffer
push    len                       ;value - length of sidlst
push    retlen                    ;address - number of siteids returned
push    total                     ;address - total number of siteidso
jsr     _rsidlst                   ;get list of site ids
```

C Function Declaration:

```
                                  /* get site ids from remote network */
long                              /* returns result of the operation */
_rsidlst(rnid, sidlst, len, retlen, total)
         long rnid;               /* remote network id */
         long *sidlst;            /* siteid buffer */
         long len;                /* length of sidlst */
         long *retlen;            /* number of siteids returned */
         long *total;             /* total number of site ids */
```

FORTRAN Subroutine Declaration:

```
c                                 ! get site ids from remote network
         subroutine _rsidls(rnid, sidlst, len, retlen, total)
             integer*4 rnid       ! remote network id
             integer*4 sidlst     ! siteid buffer
             integer*4 len        ! length of sidlst
             integer*4 retlen     ! number of siteids returned
             integer*4 total      ! total number of site ids
```

Pascal Procedure Declaration:

```
procedure _rsidlst(               {** get list of site ids from remote network}
        rnid    : longint;        {** remote network id}
        sidlst  : ^array_of_char; {** siteid buffer}
        len     : longint;        {** length of sidlst}
    var retlen  : longint;        {** number of siteids returned}
    var total   : longint         {** total number of site ids}
); external;
```

Set PCB attribute bits.

Description:

Call this routine to set the process attribute bits in the PCB for a particular process. To modify the process attributes of a process use _GETATTR first to get the current ones and set or reset the appropriate bits, then call this routine with the modified value.

Related Privileges:

none	- Allows affecting the attributes of any process with the same owner ID and group ID (UIC) as the calling process.
group	- Allows affecting the attributes of any process with the same group ID as the calling process.
world	- Allows affecting the attributes of any process.

Parameters:

pid	- A long word containing the process ID of the process whose attributes are to be changed. 0 represents the current process; -1 ($FFFFFFFF) represents the parent of the current process.
attr	- A long word containing the new attributes to set.

Process attribute bit definitions. Note that these offsets are defined for being in the high word of a longword. Because it is only a word in the PCB, if you access the PCB directly you will have to shift these numbers right by 16.

If all bits are set to zero, it signifies that the attribute bits of the process's parent are to be used. If the bits are non-zero, the attributes are to be taken as specified, unless only the high order bit (pcbattrforceset) is set. If only the pcbattrforceset bit is set, it signifies that all other bits are intended to be set to zero.

Bit Name	Bit Number	Description
pcbattrdesencrypt	16	If set, do network encryption with DES algorithm.
pcbattrfastencrypt	17	If set, do network encryption with fast algorithm.
pcbattruser1	23	If set, user attribute bit 1.
pcbattruser2	24	If set, user attribute bit 2.
pcbattruser3	25	If set, user
pcbattruser4	26	If set, user attribute bit 4.
pcbattrnowatchdog	27	If set, cannot be watchdogged.
pcbattrswappable	28	If set, the OS will not swap this process.
pcbattrprezeromem	29	If set, pages of memory are zeroed as they are allocated.
pcbattrpostzeromem	30	If set, pages of memory are zeroed as they are released.
pcbattrforceset	31	If set, then modify the bits. Must be set to cause other bits to take effect.

status - Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
errprcsnotfnd	(2)	The specified process is not in the system process table.

See Also:

_getattr - Get PCB attribute bits

Assembler Calling Sequence:

```
push    pid                     ;value - process id
push    attr                    ;value - new attribute bits
push    status                  ;address - result of the operation
jsr     _setattr                ;set the attributes
```

C Function Declaration:

```
                                /* change process attributes */
long                            /* returns result of the operation */
_setattr(pid, attr)
        long pid;               /* process id */
        long attr;              /* new attributes */
```

FORTRAN Subroutine Declaration:

```
c                               ! change process attributes
        subroutine _setatt(pid, attr, status)
            integer*4 pid       ! process id
            integer*4 attr      ! new attributes
            integer*4 status    ! result of the operation
```

Pascal Procedure Declaration:

```
procedure _setattr(             {** change process attributes}
        pid     : longint;      {** process id}
        attr    : longint;      {** new attributes}
    var status  : longint       {** result of the operation}
); external;
```

Set device protection.

Description:

Establishes the protection to be applied to a device. The protection grants access privileges to the device for classes of users.

This operation is valid for any mounted device.

To successfully change protection on a device the process must have operator privilege, bypass privilege or have the same owner id and group id as the device itself.

Related Privileges:

None	- Allows the owner of a device to modify the protection.
altuic	- Allows the process to change the protection if the owner of the process's image file is the same as the owner of the device.
bypass	- Allows the process to change the protection on any device.
operator	- Allows the process to change the protection on any device.

Parameters:

dname	- Address of a null terminated string containing the the name of the device whose protection is to be set. This string may contain up to 93 significant characters followed by a null. This string will be translated automatically by the MCS to its logical equivalent. If this string contains a file designation, the devicename portion of the file designation is used for this parameter.
prot	- File protection mask. The least significant 16 bit word of this parameter is divided into 4 nibbles. Each nibble corresponds to a class of users. The bits within each nibble represent the type of access that class of user is granted for this device. If the bit is set (1) the access is granted.

From the least to the most significant nibble the user classes are:

Ownr	- device owner
Grp	- processes with the same group id as the owner
Pub	- all other processes in the system

Sys - processes with SYSTEM privilege

```
 Sys  Pub  Grp  Ownr
|----|----|----|----|
|DWRE|DWRE|DWRE|DWRE|
|-------------------|
MSB                LSB
```

From the least to the most significant bits within the nibbles, the access privileges are:

E	- Execute access
R	- Read access
W	- Write access
D	- Delete access

A long word -1 ($FFFFFFFF) is a reserved value that means that the user's default protection mask is to be used.

status - Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
errinvdevnam	(130)	The specified devicename is syntactically incorrect.
errundevnam	(131)	The MCS does not recognize the devicename. Is the device mounted?

Device integrity errors

See Also:

_defprot - Set default protection mask
_getdprt - Get device protection
_getfprt - Get file protection
_setfprt - Set file protection

Assembler Calling Sequence:

```
push    dname                   ;address - device name
push    prot                    ;value - protection mask
push    status                  ;address - result of the operation
jsr     _setdprt                ;set device protection
```

C function declaration:

```
                                /* set device protection */
long                            /* returns result of the operation */
```

```
    _setdprt(dname, prot)
            char dname[94];             /* device name */
            long prot;                  /* protection mask */
```

Fortran Subroutine Declaration:

```
    c                                   ! set device protection
            subroutine setdpr(dname, prot, status)
                character*94 dname      ! device name
                integer*4 prot          ! protection mask
                integer*4 status        ! result of the operation
```

Pascal Procedure Declaration:

```
    procedure _setdprt(                 {** set device protection}
            dname   : string[93];       {** device name}
            prot    : longint;          {** protection mask}
        var status  : longint           {** result of the operation}
    ); external;
```

Set device status.

Description:

Allows a process to modify a device status table.

The device status is a device class dependent 128 byte table. It is maintained by the device driver for each device.

NOTE: The device status table may change with each release of the operating system. The current definition is included in each release in the file named: /SYSINCL.SYS/ DSTATDISP.*. The name of the record included in that file is "devicestatus," i.e., in your program you can declare a variable whose type is "devicestatus."

The device status table is divided into two parts. The first half is device independent and is composed of the following fields:

Name	Length (bytes)	Description
dsclassid	2	The device class. Valid classes are: (Note that these names are defined in the devtdisp.* files)

Class Name	Value	Description
dtclassttyspc	0	Character device (ttyspc)
dtclasstty	1	Character device (tty)
dtclasstapespc	2	Tape device (tapespc)
dtclasstape	3	Tape device (tape)
dtclassdiskspc	4	Disk device (diskspc)
dtclassdisk	5	Disk device (disk)
dtclassnetspc	6	Network dev. (networkspc)
dtclassnet	7	Network device (network)
dtclasspipespc	8	Pipe device (pipespc)
dtclasspipe	9	Pipe device (pipe)
dtclasssyncspc	10	BSC device (syncspc)
dtclasssync	11	BCS device (sync)
dtclassquespc	12	Queue device (quespc)
dtclassque	13	Queue device (que)

		dtclassnondevspc 14 Non-dev device(nondevspc)
		dtclassnondev 15 Non-dev device (nondev)
dsdriverid	2	The unique ID number for this device driver
dsblksz	2	The block size of the device (e.g. sector size)
dsharderr	2	The hard error count for the device
dssofterr	2	The soft error count for the device
dsreadoper	4	The number of read operations on this device
dswriteoper	4	The number of write operations on this device
dsmaxnumdev	2	Maximum # of devices this driver can handle
dscurnumdev	2	# of mounted devices using this driver
dsnumtoretry	2	# of times to retry before reporting an error
dserrorreason	4	Hardware error code for last error received on this device
dsreserved	32	Reserved
dsnexttableptr	4	Address of next device status table

The second half of the device status table is device class dependent. For TAPE class devices the second half is defined as follows:

Name	Length (bytes)	Description		
dstpstatus	2	Tape device status. A bit encoded word.		
		Bit name	bit #	Description
		dstpready	0	Set if device ready
		dstpintpend	1	Set if interrupt pending
		dstprewinding	2	Set if tape rewinding
		dstpbotdetect	3	Set if device is at physical BOT
		dstpeotdetect	4	Set if device is at physical EOT
		dstpwriteprot	5	Set if tape is write protected
dstpflagsl	2	Tape status information. A bit encoded word.		
		Bit name	bit #	Description
		dstpdoraw	0	0=Read after write disabled 1=Read after write enabled
		dstperrintenb	1	0=Error interrupts are enabled 1=Error interrupts are disabled

dstpspeed	1	Tape speed. Values are: 0 - Reserved dstpspeed12ips 1 - 12 ips dstpspeed25ips 2 - 25 ips dstpspeed30ips 3 - 30 ips dstpspeed50ips 4 - 50 ips dstpspeed90ips 5 - 90 ips dstpspeed100ips 6 - 100 ips dstpspeed125ips 7 - 125 ips
dstpdensity	1	Tape density. Values are: 0 - Reserved dstpdens800bpi 1 - 800 bpi dstpdens1600bpi 2 - 1600 bpi dstpdens3200bpi 3 - 3200 bpi dstpdens6250bpi 4 - 6250 bpi dstpdens6400bpi 5 - 6400 bpi
dstpiopbcnt	2	Number of IOPBs allocated to device
dstpcachesz	2	Number of cache elements allocated to device
dstpreserved	46	Reserved
dstpuserfield	8	User defined status

For DISK class devices the second half of the device status table is defined as follows:

Name	Length (bytes)	Description
dsdkintfac	2	Disk interleave factor
dsdkiopbcnt	2	Number of IOPB's allocated to the drive
dsdknumbsect	4	The number of sectors on the volume
dsdksectrack	2	The number of sectors on a track
dsdkheads	2	The number of heads on the device
dsdkcylinders	2	The number of cylinders on the volume
dsdkflags1	2	Disk status information. A bit encoded word.

Bit Name	Bit #	Description
dsdkdensity1	0	Device density
dsdkdensity2	1	
dsdkdenssignle		00 - Single density
dsdkdensdouble		01 - Double density
dsdkdensquad		10 - Quad density
dsdkdensreserve		11 - Reserved
dsdkdoraw	3	If set, do Read after write verify
dsdkwriteprot	4	If set, Device write protected

Bit Name	Bit #	Description
dsdkseekdir	15	Current seek direction
dsdkseekincr		0 - Increasing cylinder numbers
dsdkseekdecr		1 - Decreasing cylinder numbers

Name	Length (bytes)	Description
dsdkcurcyl	2	Current cylinder position
dsdkcachesz	2	Number of sectors in the disk cache
dsdkentryname	16	Null terminated string containing the name of this type of drive
dsdkreserved	20	Reserved
dsdkuserfield	8	User Defined status

For TTY class devices the second half of the device status table is defined as follows:

Name	Length (bytes)	Description
dstymoderegl	1	Uart mode register 1. This byte is bit encoded as follows:

Bit Name	Bit #	Description
dstymrlbaudfacl	0	Baud factor
dstymrlbaudfac2	1	
dstymrlsyncl		00 - sync 1 x clock rate
dstymrlasyncl		01 - async 1 x clock rate
dstymrlasync16		10 - async 16 x clock rate
dstymrlasync64		11 - async 64 x clock rate
dstymrlcharlenl	2	Character length definition
dstymrlcharlen2	3	
dstymrldw5bit		00 - 5 data bits
dstymrldw6bit		01 - 6 data bits
dstymrldw7bit		10 - 7 data bits
dstymrldw8bit		11 - 8 data bits
dstymrlparityctrl	4	Parity control
dstymrlpardis		0 - disable parity
dstymrlparenb		1 - enable parity
dstymrlparitytype	5	Parity type
dstymrlparodd		0 - odd parity
dstymrlparevn		1 - even parity
dstymrlstopbitsl	6	Async mode # of stop bits

		dstymrlstopbits2	7	Async mode # of stop bits
		dstymrlbinv		00 - invalid
		dstymrlsbl		01 - 1 stop bit
		dstymrlsbl5		10 - 1.5 stop bits
		dstymrlsb2		11 - 2 stop bits
		dstymrltransctrl	6	Sync mode transparent
		dstymrlnormal		0 - normal
		dstymrltrans		1 - transparent
		dstymrlnumsync	7	Sync mode # of syncs
		dstymrlsyncdouble		0 - double sync
		dstymrlsyncsingle		1 - single sync
dstymodereg2	1	Uart mode register 2. This byte is bit encoded as follows:		
		Bit Name	Bit #	Description
		dstymr2baudrt1	0	The baud rate
		dstymr2baudrt2	1	Baud rate continued
		dstymr2baudrt3	2	Baud rate continued
		dstymr2baudrt4	3	Baud rate continued
		dstymr2baud50		0000 - 50 baud
		dstymr2baud75		0001 - 75 baud
		dstymr2baud110		0010 - 110 baud
		dstymr2baud1345		0011 - 134.5 baud
		dstymr2baud150		0100 - 150 baud
		dstymr2baud300		0101 - 300 baud
		dstymr2baud600		0110 - 600 baud
		dstymr2baud1200		0111 - 1200 baud
		dstymr2baud1800		1000 - 1800 baud
		dstymr2baud2000		1001 - 2000 baud
		dstymr2baud2400		1010 - 2400 baud
		dstymr2baud3600		1011 - 3600 baud
		dstymr2baud4800		1100 - 4800 baud
		dstymr2baud7200		1101 - 7200 baud
		dstymr2baud9600		1110 - 9600 baud
		dstymr2baud19200		1111 - 19200 baud
		dstymr2recvclock	4	Receiver clock
		dstymr2recextclk		0 - External clock
		dstymr2recintclk		1 - Internal clock
		dstymr2transclock	5	Transmitter clock
		dstymr2trnextclk		0 - External clock
		dstymr2trnintclk		1 - Internal clock
			6-7	Reserved
dstycmdreg	1	Uart command register. Bit encoded.		
		Bit Name	Bit #	Description
		dstycrtransctrl	0	Transmitter control
		dstycrtcdis		0 - Disable transmitter

		dstycrtcenb		1 - Enable transmitter
		dstycrdtr	1	Data terminal ready
		dstycrdtrhigh		0 - DTR high
		dstycrdtrlow		1 - DTR low
		dstycrrecvcrtl	2	Receiver control
		dstycrrcdis		0 - Disable receiver
		dstycrrcenb		1 - Enable receiver
		dstycrforcebrk	3	Async force break
		dstycrbrknorm		0 - normal
		dstycrbrkforce		1 - force break
		dstycrsenddle	3	Sync send DLE
		dstycrdlenorm		0 - normal
		dstycrdlesend		1 - send DLE
		dstycrreseterror	4	Reset error
		dstycrnoreset		0 - normal
		dstycrreseterr		1 - reset error
		dstycrrts	5	Request to send
		dstycrrtshigh		0 - RTS high
		dstycrrtslow		1 - RTS low
		dstycropermodel	6	Operating mode
		dstycropermode2	7	Operating mode continued
		dstycromnormal		00 - Normal operation
		dstycromautoecho		01 - Async autoecho
		dstycromstripdle		01 - Sync strip DLE
		dstycromlocallp		10 - Local loop back
		dstycromremotelp		11 - Remote loop back
dstytermtype	1	Terminal type definition. This byte contains values for each type of terminal.		
		Value Name	Value	Description
			0-15	User defined types
			16-246	Reserved
		dstywit	247	WIT terminal
		dstyhydra	248	Hydra terminal
		dstyvt100	250	VT-100 terminal
		dstyvt52	251	VT-52 terminal
		dstyt7000	252	T-7000 terminal
		dstymg8000	253	MG-8000 terminal
		dstytvi912c	254	TVI 912 C terminal
		dstyvisual200	255	Visual 200 terminal
dstystatreg	1	Uart status register. Bit encoded.		
		Bit Name	Bit #	Description
		dstysrtransrdy	0	Transmitter buffer ready
		dstysrtranfull		0 - Transmitter full

		dstysrtranempty		1 - Transmitter empty
		dstysrrecvrdy	1	Receiver buffer ready
		dstysrrecvempty		0 - Receiver empty
		dstysrrecvfull		1 - Receiver full
		dstysrdschg	2	DSR or DCD change
		dstysrdsrnormal		0 - Normal
		dstysrdsrchange		1 - Change in DSR or DCD
		dstysrparityerr	3	Parity error
		dstysrparnormal		0 - Normal
		dstysrparerror		1 - Async parity error. Sync parity error or DLE received
		dstysroverrunerr	4	Overrun error
		dstysrovernormal		0 - Normal
		dstysrovererror		1 - Overrun error
		dstysrframingerr	5	Framing error
		dstysrframnormal		0 - Normal
		dstysrframerror		1 - Async framing error. Sync SYN char.
		dstysrdcddetect	6	DCD Detect
		dstysrdcdhigh		0 - DCD high
		dstysrdcdlow		1 - DCD low
		dstysrdsrdetect	7	DSR Detect
		dstysrdsrhigh		0 - DSR high
		dstysrdsrlow		1 - DSR low
dstypacketterm	1	Holds code for packet termination characters		
		Value Name	Value	Description
		dstyptnoterm	0	Do not terminate packet on any control characters
		dstyptallterm	1	Terminate packets on all control characters
		dstyptcrterm	2	Terminate packet on carriage return <CR> character
dstyflagsl	2	Terminal status information. Bit encoded.		
		Bit Name	bit #	Description
		dstycontrolc	0	Control C enable (0 = enabled)
		dstyxonxoff	1	xon xoff enable (0 = enabled)
		dstycontrolx	2	Control X enable (0 = enabled)
		dstycontrolz	3	Control Z enable (0 = enabled)

		dstycontrolo	4	Control O enable (0 = enabled)
		dstytabmap	5	Tab map enable (1 = enabled)
		dstymask8bit	6	Mask 8th bit enable (0 = enabled)
		dstycontrolu	7	Control U enable (0 = enabled)
		dstybroadcast	8	Broadcast enable (0 = enabled)
		dstyhandshake1	9	Handshaking type
		dstyhandshake2	10	
		dstyhsbell		00 - No handshake, send bell
		dstyhssoft		01 - Software handshake
		dstyhshard		10 - Hardware handshake
		dstyhsnone		11 - No handshake, no bell
		dstyduplex	11	Full/half duplex (0 = full duplex)
		dstymodemctrl	12	Modem control enable (1 = enabled)
		dstyautobaud	13	Auto baud enable (1 = enabled)
		dstyremote	14	Remote enable (1 = enabled)
dstyinputcnt	2	Count of characters in input interrupt buffer		
dstyoutptcnt	2	Count of characters in output interrupt buffer		
dstycolumnpos	2	Current column position		
dstyinbufsz	2	Input buffer size in bytes		
dstyoutbufsz	2	Output buffer size in bytes		
dstywidth	2	The width of the given terminal screen		
dstylength	2	The length of the given terminal screen		
dstysubreadoper	4	Number of sub-read operations		
dstysubwriteoper	4	Number of sub-write operations		
dstyreserved	26	Reserved		
dstyuserfield	8	User defined status		

For PIPE class devices the second half of the device status table is defined as follows:

Name	Length (bytes)	Description
dsppreaderpid	4	Process ID of pending reader
dsppwriterpid	4	Process ID of pending writer
dsppipeid	4	The pipe's ID
dsppbuffersz	2	The buffer size in bytes
dsppbuffercnt	2	Number of characters in the pipe buffer
dsppreserved	40	Reserved
dsppuserfield	8	User defined status

For SYNC class devices the second half of the device status table is defined as follows:

Name	Length (bytes)	Description
dssymodereg1	1	Mode register 1 of the uart (See DSTYMODEREG1 for bit definitions)
dssymodereg2	1	Mode register 2 of the uart (See DSTYMODEREG2 for bit definitions)
dssycmdreg	1	Command register of the uart (See DSTYCMDREG for bit definitions)
dssytermtype	1	Terminal type definition. A binary value.
dssystatreg	1	Status register of uart. (See DSTYSTATREG for bit definitions)
dssynumbsync	1	Number of sync characters to write
dssyflags1	2	Device Status flags. Bit encoded.

dssytermtype values:

Value Name	Value	Description
dssyibm3741	249	IBM 3741 terminal
dssyibm2968	250	IBM 2968 terminal
dssyibm2770	251	IBM 2770 terminal
dssyibm3276	252	IBM 3276 terminal
dssyibm3275	253	IBM 3275 terminal
dssyibm2780	254	IBM 2780 RJE
dssyibm3780	255	IBM 3780 RJE

dssyflags1 bits:

Bit Name	Bit #	Description
dssymultipnt	0	0=point to point 1=multipoint
dssyebcdic	1	0=ascii line 1=ebcdic line
dssycrcccitt	2	0=crc-16 1=crc-ccitt

dssylrc	3	0=crc (on above types) 1=lrc
dssyasctoebcw	4	0=no translate on write 1=translate ascii to ebcdic on write
dssyebctoascr	5	0=no translate on read 1=translate ebcdic to ascii on read
dssytranstbl2	6	0=use translate table 1 1=use translate table 2

dssyinputcnt	2	Number of characters in input interrupt buffer
dssyoutputcnt	2	Number of characters in output interrupt buffer
dssyinbufsz	2	Input buffer size in bytes
dssyoutbufsz	2	Output buffer size in bytes
dssyprevrderr	4	Error from previous un-verified read
dssyprevwrerr	4	Error from previous no-wait write
dssyprevrdtype	1	Type of previous read dssynontran - 0 Non-transparent read dssytran - 1 Transparent read
dssynumbtrpad	1	The number of trailing pads to write
dssyrecsize	2	Used in transparent mode with ITBs
dssyreserved	28	Reserved
dssyuserfield	8	User defined status

For NETWORK class devices the second half of the device status table is defined as follows:

Name	Length (bytes)	Description		
dsnkflags	2	Device status flags. Bit encoded.		
		Bit Name	Bit #	Description
		dsnkbyte	0	0=datagram mode 1=byte mode
		dsnkmodemctrl	1	0=not enabled 1=modem ctrl enabled
dsnkwindowsize	1	Window size the circuit will use		
dsnkreserved	53	Reserved		
dsnkuserfield	8	User defined status		

For NONDEV class devices the second half of the device status table is defined as follows:

Name	Length (bytes)	Description
dsnduserfield	64	User defined status

For QUEUE class devices the second half of the device status table is defined as follows:

Name	Length (bytes)	Description
dsquassocdev	9	A null terminated string containing the name of the physical printer device
dsqusenddev	9	A null terminated string containing the name of the physical device that control messages are to be sent to
dsquformname	10	A null terminated string containing the current form name
dsqunumexec	2	This is the maximum number of entries that can execute concurrently
dsqucurnumexec	2	This is the number of entries that are currently active
dsquflags	2	Device Status flags Bit encoded.
dsqulength	2	This holds the length of the forms of the printer associated with this queue
dsquwidth	2	This hold sthe width of the forms of the printer associated with this queue
dsqunextentry	4	This is the entry number of the next entry to be enqued
dsqutype	1	This contains the type of queue this is. The values are:

dsquflags bits:

Bit Name	Bit #	Description
dsquflupdating	0	Currently updating queue control file
dsquflqmstay	1	Queue manager process will remain running even when queue is empty
dsquflnorestart	2	When queue is mounted it does not restart jobs in queue

Value Name	Value	Description
dsqutpprint	1	Print type queue
dsqutpjob	2	Job entry type queue

dsqubaseprior	1	This contains the priority that entries will be queued at if they specify the default priority
dsqureserved	20	Reserved
dsquuserfield	8	User defined status

To perform a set status operation the process must have write privilege to the device and either be the owner of the device (matching UICs) or have writephys privilege.

Related Privileges:

none	- Allows access to the device only if the process has write privilege to the device and has the same owner ID and group ID (UIC) as the device.
altuic	- Allows the process to access the device if the owner of the image file for the current process has access to the device as described above.
bypass	- Allows the process to access the device without requiring write privilege. The process must still either be the owner of the device or have writephys privilege.
system	- Allows the process to access the device if the system has write privilege to the device as described above. (This does not obviate the need for device ownership or writephys privilege).
writephys	- Allows physical access to devices as described above. (This does not obviate the need for write privilege).

Parameters:

dname	- Address of a null terminated string containing the name of the device whose status table is to be written. This string will be translated automatically by the MCS to its logicl equivalent. This string may contain up to 93 valid characters followed by a null. If this string contains a file designation, the devicename portion of the file designation is used for this parameter.

dstat	- Address of the 128 byte device status table that is to be written. This buffer must be word aligned.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
errinvdevnam	(130)	The specified devicename is syntactically incorrect.
errundevnam	(131)	The MCS does not recognize the devicename. Is the device mounted?
errnowritepriv	(145)	The process does not have Write Privilege for the file.
errinvdrvnum	(311)	A value in at least one field of the devicename is disallowed.

See Also:

_getdnam	- Get devicename
_getdst	- Get device status
_giodst	- Get device status with LUN
_physop	- Perform physical device operation
_siodst	- Set device status with LUN

Assembler Calling Sequence:

```
%%sys$disk/sysincl.sys/dstatdisp.asm
push    dname                   ;address - devicename
push    dstat                   ;address - device status
push    status                  ;address - result of the operation
jsr     _setdst                 ;set device status
```

C Function Declaration:

```
#include "sys$disk/sysincl.sys/dstatdisp.h"
                                /* set device status */
long                            /* returns result of the operation */
_setdst(dname, dstat)
         char dname[94];        /* devicename */
         devicestatus dstat;    /* device status */
```

FORTRAN Subroutine Declaration:

```
     c                                 ! set device status
               subroutine _setdst(dname, dstat, status)
                  character*94 dname ! devicename
                  character*(*) dstat ! device status
                  integer*4 status   ! result of the operation
```

Pascal Procedure Declaration:

```
     %%sys$disk/sysincl.sys/dstatdisp.pas
     procedure _setdst(              {** set device status}
             dname   : string[93];   {** devicename}
             dstat   : ^array_of_char; {** device status}
         var status  : longint       {** result of the operation}
     ); external;
```

Set device UIC.

Description:

This system call allows a process to change the user identification code (UIC) of a device. By changing the UIC, the ownership of the device is changed.

To successfully change the UIC of a device, the calling process must have operator privilege, and either group privilege or world privilege.

If the calling process has group privilege, and the group ID of the device is the same as the group ID of the calling process, the process can modify the owner ID of the device.

If the calling process has world privilege and operator privilege, it can change the UIC of any device to be any other UIC except zero.

This system call is valid for any class of device.

Related Privileges:

none	- The process cannot change the UIC of the device.
group	- If the process also has operator privilege, it can modify the owner ID of any mounted device which has the same group ID as the calling process.
operator	- Allows setting the UIC if the process also has either group or world privilege.
world	- If the process also has operator privilege it can modify the UIC of any mounted device to any other UIC except zero.

Parameters:

dname	- Address of a null terminated string containing the name of the device whose UIC is to be changed. This string will be translated automatically by the WMCS to its logical equivalent. This string may contain up to 93 valid characters followed by a null byte. If this string contains a file designation, the devicename portion of the file designation is used for this parameter.
uic	- A long word containing the user identification code. This long word is divided into two fields. The most

significant 16 bits constitute the owner ID number. The least significant 16 bits constitute the group ID number (identifying the group to which the user belongs).

The value $FFFFFFFF (-1) is a reserved value that means to use the default UIC, i.e., the UIC of the calling process.

A value of zero is invalid.

status - Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
errinvdevnam	(130)	The specified devicename is syntactically incorrect.
errundevnam	(131)	The MCS does not recognize the devicename. Is the device mounted?

See Also:

_getduic - Get device UIC
_getfuic - Get file UIC
_getuic - Get process UIC
_setfuic - Set file UIC
_setuic - Set process UIC

Assembler Calling Sequence:

```
push    dname                  ;address - devicename
push    uic                    ;value - owner ID code
push    status                 ;address - result of the operation
jsr     _setduic               ;set device UIC
```

C Function Declaration:

```
                               /* set device UIC */
long                           /* returns result of the operation */
_setduic(dname, uic)
        char dname[94];        /* devicename */
        long uic;              /* owner ID code */
```

FORTRAN Subroutine Declaration:

```
c                               ! set device UIC
        subroutine setdui(dname, uic, status)
           character*94 dname  ! devicename
           integer*4 uic       ! owner ID code
           integer*4 status    ! result of the operation
```

Pascal Procedure Declaration:

```
procedure _setduic(          {** set device UIC}
        dname   : string[93];  {** devicename}
        uic     : longint;     {** owner ID code}
    var status  : longint      {** result of the operation}
); external;
```

Set event flags.

Description:

All the event flags corresponding to 1 bits in the mask provided will be set in the event flags of the specified process.

Related Privileges:

none	- Allows setting event flags in processes with the same owner ID and group ID (UIC) as the calling process.
group	- Allows setting event flags in processes with the same group ID as the calling process.
world	- Allows setting event flags in any process.

Parameters:

pid	- Process ID of the process whose event flags are to be set.
efmask	- Event flag mask. Contains the mask representing which event flags are to be set.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
errprcsnotfnd	(2)	The specified process is not in the system process table.

See Also:

_andevnt - Wait for AND of event flags
_clrevnt - Clear event flags
_getevnt - Read event flags
_orevnt - Wait for OR of event flags

Assembler Calling Sequence:

```
push    pid                 ;value - process ID
push    efmask              ;value - event flag mask
push    status              ;address - result of the operation
jsr     _setevnt            ;set event flags
```

C Function Declaration:

```
                                        /* set event flags */
    long                                /* returns result of the operation */
    _setevnt(pid, efmask)
            long pid;                   /* process ID */
            long efmask;                /* event flag mask */
```

FORTRAN Subroutine Declaration:

```
    c                                   ! set event flags
            subroutine setevn(pid, efmask, status)
                integer*4 pid           ! process id
                integer*4 efmask        ! event flag mask
                integer*4 status        ! result of the operation
```

Pascal Procedure Declaration:

```
    procedure _setevnt(                 {** set event flags}
           pid      : longint;          {** process ID}
           efmask   : longint;          {** event flag mask}
        var status  : longint           {** result of the operation}
    ); external;
```

setexit - Define exit handler.

Description:

The user may define an exit handler to be executed when the process is deleted. An exit handler can be used as a cleanup and restore routine or as a mechanism for "catching" otherwise fatal errors. Exit routines can determine the reason they are called by using the _errno system call to retrieve the process abort code. All return code values -65535 and +65535 are reserved to MCS. All numbers beyond this range are reserved for users to define as they wish. Exit routines cannot have have any call arguments.

The exit handler for a process is executed when a process exits regardless of the cause or circumstances of the exit. The exit handler is executed in the same processor mode (user of supervisor mode) as the mode from which the exit handler was defined or was executed, which ever is higher.

When control is passed to the exit handler the OS notes that the process is executing its exit handler. If a fatal process error occurs while the process is executing its exit handler, the process will be deleted without passing through the exit handler again. If the process wants an exit handler to be called again as the process exits, it must define a new exit handler while it is executing its exit handler. Since no further abort conditions will be honored until the next time the process is scheduled, a carefully written exit handler can determine the reason for being transferred to the exit handler and be able to define a new one if necessary.

To terminate the process normally once the exit handler has been called, issue a call to _exproc from within the exit handler.

When an exit handler is called, the registers contain the context of the process at the point it was interrupted. The return address and status register of the interrupted process are at bytes 2 and 0 respectively from the top of the stack. Return to the main process can be effected by executing an RTR or RTE instruction. Because an exit handler is capable of being called asynchronously in relation to the main process, changing global variables from within an exit handler may cause seemingly mysterious results when control is returned to the main body of a process which uses those same variables.

Related Privileges:

None.

Parameters:

adr - Address of the exit handler routine.

Diagnostics:

None.

See Also:

_errno	- Receive process abort reason
_exitrtn	- Define a returnable exit handler
_exproc	- Terminate the specified process

Assembler Calling Sequence:

```
push    adr                         ;value - address of exit handler
jsr     _setexit                    ;define exit handler
```

C function declaration:

```
                                    /* define exit handler */
void                                /* no result */
_setexit (adr)
        long adr;                   /* address of exit handler */
```

Fortran Subroutine Declaration:

```
c                                   ! define exit handler
        subroutine setexi(adr)
            external adr            ! address of exit handler
```

Pascal Procedure Declaration:

```
procedure  setexit(                 {** define exit handler}
        adr     : longint           {** address of exit handler}
); external;
```

Write file control block.

Description:

This SVC allows the calling process to update the file control block for an open file on any disk class device. Note that this requires that the calling process have writephys privileges and have write access to the file. For security reasons the file should have been opened with write locked access.

NOTE: The FCB file is the heart of the file system. Careless tampering with the FCB file can cause severe damage to the file system's integrity.

CAUTION: The format of the file control block my change with each release. The current definition is included in each release in the file /SYSINCL.SYS/FCBDISP.*. The name of the FCB record is "fcbtype," i.e., in your program you can declare a variable whose type is "fcbtype."

There are several variations on the format of file control blocks, depending on the class of device which contains the file. Disk files contain "primary" FCBs and "continuation" FCBs. Tape files have "tape" FCBs. All other files have "tty" FCBs. You can only set the FCB for disk class devices.

The format of the first 14 bytes of the FCB record is the same for all types of FCBs. The format of this common type is:

Name	Length (bytes)	Description
fcbnum	4	FCB number for this FCB. The record number of this record within the FCB file. For tty FCBs, the value of this field is zero. This field may not be changed.
fcbseqnum	2	FCB sequence number. This number is unique for each usage of this FCB. For tty FCBs, the value of this field is zero. This field may not be changed.

fcbcontfcbnum	4	FCB number of continuation FCB. The record number of the next FCB for this same file. For tape and tty FCBs, the value of this field is zero. This field may be zeroed (remove a continuation) but no other values may be set (add a continuation).
fcbcontfcbseq	2	Sequence number of the continuation FCB. For tape and tty FCBs, the value of this field is zero. This field may be zeroed (remove a continuation) but no other values may be set (add a continuation).
fcbusageid	1	Usage ID field. The type of FCB. Values are:
		fcbunalloc 0 This FCB is unused. The data in this record is invalid.
		fcballocroot 1 This record contains a root FCB.
		fcballoccont 2 This record contains a continuation FCB.
fcbextusecnt	1	Number of extent fields in use within this FCB.

The format of the last 242 bytes of the FCB is different for "primary" FCBs as opposed to "continuation" FCBs. For primary FCBs (disk, tape and tty) the format is as follows:

fcbfiletype	2	File type. For tty files, it is always set to zero (a data file). Valid file types are:

File Type	Value	Description
fcbftdata	0	data file
fcbftdir	1	directory file
fcbftimage	2	image file
fcbftksamdata	3	KSAM data file
fcbftksamkey	4	KSAM key file
fcbftllimage	5	LL image file
fcbftarchcont	6	archive file continuation
fcbftencrypt	7	encrypted file
fcbftsystem	8	system file
fcbftarchive	9	archive file
	20-255	reserved
	256-65535	user-defined file types

fcbfilename	9	Filename. For disk and tape files it contains the filename portion of the file designation. For tty files it contains the devicename.

fcbfileext	3	File extension. For tty FCBs this field is set to zero.
fcbfilevers	2	File version number. For tty FCBs this field is set to zero.
fcbdirfcbnum	4	Directory FCB number. The FCB number of the directory file containing this file. For tape and tty FCBs it contains zero.
fcbdirseqnum	2	Directory sequence number. The sequence number of the directory file containing this file. For tty FCBs this field contains zero.
fcbrecordsz	2	Default record size. For tty FCBs this field is set to 1.
fcbuserid	2	Owner ID of the file's owner.
fcbgroupid	2	Group ID of the file's owner.
fcbprotect	2	File protection field. For tty FCBs it contains the device protection.
fcbcreatemstim	4	The most significant 32 bits of the file creation date in system time format (year and day). For tty FCBs, it contains the year and day that the device was mounted.
fcbcreatelstim	4	The least significant 32 bits of the file creation date in system time format (hour, minute, ...). For tty FCBs, it contains the hour, minute, ... that the device was mounted.
fcbmodmstim	4	The most significant 32 bits of the date the file was last modified (year and day). For tty FCBs, it contains the year and day that the device was mounted.
fcbmodlstim	4	The least significant 32 bits of the date the file was last modified (hour, minute, second, tick). For tty FCBs, it contains the hour, minute, ... that the device was mounted.
fcbreserved	4	Reserved space.
fcbphysicalsz	4	The physical size of the file in bytes. For tty FCBs this field is set to zero.
fcblogicalsz	4	The logical size of the file in bytes. For tty FCBs this field is set to zero.
fcbfileid	2	File ID of the file. For tty FCBs this field is set to zero.
fcbrootextblk	180	File extent fields. There are 30 extent fields in a primary FCB. Each extent field is composed of 6 bytes. The first two bytes represent the number of sectors in that extent. The last four bytes are the logical sector number of the first sector in that extent.
fcbnotcksum	2	The FCB's notted checksum.

The format of the last 242 bytes of the FCB for "continuation" FCBs (disk only) is as follows:

fcbcontextblk	240	File extent fields in a continuation FCB. There are 40 extent fields in a continuation FCB. Each extent field is composed of 6 bytes. The first two bytes represent the number of sectors in that extent. The last four bytes are the logical sector number of the first sector in that extent.
fcbnotcksum	2	The FCB's notted checksum.

Related Privileges:

none	- Cannot write the FCB
writephys	- Allows the process to update the FCB if the process also has write access to the file.

Parameters:

lun	- Logical unit number of the file whose FCB is being updated.
cont	- Which part of the FCB for this file is to be updated. 0=root FCB, 1=first continuation FCB, etc.
fcbuff	- Address of a 256-byte buffer containing the FCB to be written. This buffer must be word aligned.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
erridxrange	(56)	The table ends before the specified occurrence.
errinvlfn	(132)	The logical unit number does not correspond to an open file.
errnowriteacc	(142)	The process does not have write-access to the specified file.

See Also:

_create	- Create a file
_getfcb	- Get file control block
_open	- Open a file
_setfprt	- Set file protection

Assembler Calling Sequence:

```
    push    lun                     ;value - logical unit number
    push    cont                    ;value - continuation FCB number
    push    fcbuff                  ;address - buffer containing FCB
    push    status                  ;address - result of the operation
    jsr     _setfcb                 ;write file control block
```

C Function Declaration:

```
    #include "sys$disk/sysincl.sys/fcbdisp.h"
                                    /* write file control block */
    long                            /* returns result of the operation */
    _setfcb(lun, cont, fcbuff)
            long lun;               /* logical unit number */
            long cont;              /* continuation FCB number */
            fcbtype fcbuff;         /* buffer containing FCB */
```

FORTRAN Subroutine Declaration:

```
    c                               ! write file control block

            subroutine setfcb(lun, cont, fcbuff, status)
                integer*4 lun       ! logical unit number
                integer*4 cont      ! continuation FCB number
                character*(*) fcbuff ! buffer containing FCB
                integer*4 status    ! result of the operation
```

Pascal Procedure Declaration:

```
    %%sys$disk/sysincl.sys/fcbdisp.pas
    procedure _setfcb(              {** write file control block}
            lun      : longint;     {** logical unit number}
            cont     : longint;     {** continuation FCB number}
            fcbuff   : ^array_of_char; {buffer containing FCB}
        var status   : longint      {** result of the operation}
    ); external;
```

Set file ID.

Description:

Allows a process to change the file indentification code (fid) on an open file. The file identification code is a 16 bit word which can have any value.

This operation is valid on any disk file.

To successfully change the fid, the process must have successfully opened the file for write access.

Related Privileges:

None.

Parameters:

lun	- Logical unit number of the file whose file id is to be changed.
fid	- The value to be assigned to the fid field for this file.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errnomemavail	(7)	All available memory has been allocated.
errnowriteacc	(142)	The process does not have write-access to the specified file.
errinvcloper	(173)	The device class is inappropriate for the operation.
		Device integrity errors

See Also:

_getfid - Get file id

Assembler Calling Sequence:

```
push    lun                      ;value - logical unit number
push    fid                      ;value - file id
push    status                   ;address - result of the operation
jsr     _setfid                  ;set file id
```

C Function Declaration:

```
                                 /* set file id */
```

```
    long                                /* returns result of the operation */
    _setfid(lun, fid)
            long lun;                   /* logical unit number */
            long fid;                   /* file id */
```

Fortran Subroutine Declaration:

```
    c                                   ! set file id
            subroutine setfid(lun, fid, status)
                integer*4 lun           ! logical unit number
                integer*4 fid           ! file id
                integer*4 status        ! result of the operation
```

Pascal Procedure Declaration:

```
    procedure _setfid(                  {** set file id}
            lun     : longint;          {** logical unit number}
            fid     : longint;          {** file id}
        var status  : longint           {** result of the operation}
    ); external;
```

Set file protection.

Description:

Establishes the protection to be applied to a file. The protection grants access privileges to the file for classes of users.

This operation is valid for files on any mounted device except tape class devices.

To successfully change protection on a file the process must have successfully opened the file. In addition, the process must have bypass privilege or operator privilege or have the same owner id and group id (uic) as the file itself.

Related Privileges:

None	- Allows the owner of a file to modify the protection.
altuic	- Allows the process to change the protection if the owner of the process's image file is the same as the owner of the file.
bypass	- Allows the process to change the protection on any file independent of file protection.
operator	- Allows the process to change the protection on any file independent of file protection.

Parameters:

lun	- The logical unit number of the file whose protection is to be set.
prot	- File protection mask. The least significant 16 bit word of this parameter is divided into 4 nibbles. Each nibble corresponds to a class of users. The bits within each nibble represent the type of access that class of user is granted for this file. If the bit is set (1) the access is granted.

From the least to the most significant nibble the user classes are:

Ownr	- file owner
Grp	- processes with the same group id as the owner
Pub	- all other processes in the system
Sys	- processes with SYSTEM privilege

```
Sys  Pub  Grp  Ownr
```

```
|----|----|----|----|
|DWRE|DWRE|DWRE|DWRE|
|-------------------|
MSB                LSB
```

From the least to the most significant bits within the nibbles, the access privileges are:

E	- Execute access
R	- Read access
W	- Write access
D	- Delete access

A long word -1 is a reserved value that means that the users default protection mask is to be used.

status - Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
errinvlfn	(132)	The logical unit number does not correspond to an open file.
errinvcloper	(173)	The device class is inappropriate for the operation.

See Also:

_create - Create a file
_getfcb - Get file control block
_setfcb - Write file control block

Assembler Calling Sequence:

```
push    lun                     ;value - logical unit number
push    prot                    ;value - protection mask
push    status                  ;address - result of the operation
jsr     _setfprt                ;set file protection
```

C function declaration:

```
                                /* set file protection */
long                            /* returns result of the operation */
_setfprt(lun, prot)
        long lun;               /* logical unit number */
        long prot;              /* protection mask */
```

Fortran Subroutine Declaration:

```
c                                    ! set file protection
        subroutine setfpr(lun, prot, status)
            integer*4 lun            ! logical unit number
            integer*4 prot           ! protection mask
            integer*4 status         ! result of the operation
```

Pascal Procedure Declaration:

```
    procedure _setfprt(              {** set file protection}
            lun     : longint;       {** logical unit number}
            prot    : longint;       {** protection mask}
        var status  : longint        {** result of the operation}
    ); external;
```

Set file record size.

Description:

Sets new current file record size on an open file. The file record size is the number of bytes returned when one record is requested from the operating system. All files have a default record size that was specified when the file was created or opened. This system call overrides the current record size.

Related Privileges:

None.

Parameters:

lun	- Logical unit number of the file whose record size is to be changed.
newrsz	- The new record size for this file. Only the low order 16 bits of the longword are used.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinvlfn	(132)	The logical unit number does not correspond to an open file.

See Also:

_getfrsz - Get file record size

Assembler Calling Sequence:

```
push    lun                   ;value - logical unit number
push    newrsz                ;value - new record size
push    status                ;address - result of the operation
jsr     _setfrsz              ;set file id
```

C Function Declaration:

```
                                    /* set file id */
    long                            /* returns result of the operation */
    _setfrsz(lun, newrsz)
            long lun;               /* logical unit number */
            long newrsz;            /* new record size */
```

FORTRAN Subroutine Declaration:

```
    c                               ! set file id
            subroutine _setfrsz(lun, newrsz, status)
                integer*4 lun       ! logical unit number
                integer*4 newrsz    ! new record size
                integer*4 status    ! result of the operation
```

Pascal Procedure Declaration:

```
    procedure _setfrsz(             {** set file id}
            lun     : longint;      {** logical unit number}
            newrsz  : longint;      {** new record size}
        var status  : longint       {** result of the operation}
    ); external;
```

Set file UIC.

Description:

This allows a process to change the user identification code (uic) on a given file. By changing the uic the ownership of the file is changed.

This operation is valid for any disk file.

To successfully change the uic of a file, the calling process must have successfully opened the file. In addition, the calling process must have operator privilege, and either group privilege or world privilege.

If the calling process has group privilege, and the group id of the file is the same as the group id of the calling process, the process can modify the owner id of the file.

If the calling process has world privilege and operator privilege it can change the uic of any file to be any other uic except zero.

Related Privileges:

none	- The process cannot change the uic of the file.
group	- If the process also has operator privilege, it can modify the owner id of any disk file which has the same group id as the calling process.
operator	- Allows setting the uic if the process also has either group or world privilege.
world	- If the process also has operator privilege it can modify the uic of any disk file to any other uic except zero.

Parameters:

lun	- A long word containint the logical unit number of the file whose uic is to be changed.
uic	- A long word containing the uic that the file will receive. This long word is divided into two fields. The most significant 16 bits constitute the owner id number. The least significant 16 bits constitute the group id number (identifying the group to which the user belongs. A long word -1 ($FFFFFFFF) is a reserved value that means to assign the default uic, i.e. the uic of the calling process.

status - Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
errinvlfn	(132)	The logical unit number does not correspond to an open file.
errinvcloper	(173)	The operation is inappropriate for the device class.

See Also:

_getdst	Get device status
_getduic	Get device uic
_getfcb	Get file control block
_getfuic	Get file uic
_getuic	Get process uic
_setduic	Set device uic
_setfcb	Write file control block
_setuic	Set process uic

Assembler Calling Sequence:

```
push    lun                         ;value - logical unit number
push    uic                         ;value - the new uic
push    status                      ;address - result of the operation
jsr     _setfuic                    ;Set file uic
```

C function declaration:

```
                                    /* set file uic */
long                                /* returns result of the operation */
_setfuic(lun, uic)
        long lun;                   /* logical unit number */
        long uic;                   /* the new uic */
```

Fortran Subroutine Declaration:

```
c                                   ! set file uic
        subroutine setfui(lun, uic, status)
            integer*4 lun           ! logical unit number
            integer*4 uic           ! the new uic
            integer*4 status        ! result of the operation
```

Pascal Procedure Declaration:

```
procedure _setfuic(                 {** set file uic}
        lun     : longint;          {** logical unit number}
        uic     : longint;          {** the new uic}
```

```
    var status  : longint          {** result of the operation}
); external;
```

setmprt - Change access protection of a named shared memory area.

Description:

_Setmprt is used to establish the protection of a named sharable memory area. The protection grants access privileges to the named memory area for classes of users.

To successfully change the protection on a named sharable memory area the process must have the same owner id and group id (uic) as the memory area, or have operator privilege, or have bypass privilege.

Related Privileges:

none	- Allows modifying the protection of a named shared memory area which has the same owner as the process.
altuic	- Allows modifying the protection of a named shared memory area if the owner of the process's image file is the same as the owner of the memory area.
bypass	- Allows the process to modify the protection of any named shared memory area.
operator	- Allows the process to modify the protection of any named shared memory area.

Parameters:

mname	- Address of a null terminated string identifying the specific memory area. This string will be translated automatically by WMCS into its logical equivalent. This string may contain up to 93 significant characters followed by a null.
prot	- Protection mask. The least significant 16 bit word of this parameter is divided into 4 nibbles. Each nibble corresponds to a class of users. The bits within each nibble represent the type of access that class of user is granted for this memory area. If the bit is set (1) the access is granted.

From the least to the most significant nibble the user classes are:

Ownr	- owner of the memory area
Grp	- processes with the same group id as the owner
Pub	- all other processes in the system

```
                    Sys  - processes with SYSTEM privilege

                     Sys  Pub  Grp  Ownr
                    |----|----|----|----|
                    |DWRE|DWRE|DWRE|DWRE|
                    |-------------------|
                   MSB                 LSB
```

From the least to the most significant bits within the nibbles, the access privileges are:

E	- Execute access
R	- Read access
W	- Write access
D	- Delete access

The value $FFFFFFFF (-1) is a reserved value that means that the users default protection mask is to be used.

status - Address of a long word to receive the result of the operation.

Diagnostics:

ermoname (82) The name specified does not exist.

See Also:

_defmem	- Define a named sharable memory area.
_udefmem	- Undefine a named sharable memory area.
_shrmem	- Share a named sharable memory area.
_ushrmem	- Unshare a named sharable memory area.
_getmlst	- Get a list of named sharable memory areas.
_setmuic	- Change owner of a named sharable memory area.

Assembler Calling Sequence:

```
push    mname            ; address - name of memory area
push    prot             ; value   - new protection
push    status           ; address - result of the operation
jsr     _setmprt         ; Change protection of a memory area.
```

C Function Declaration:

```
                                /* change protection of a memory area */
long                            /* returns result of the operation */
_setmprt(mname,prot)
        char    mname[94];      /* name of memory area */
```

```
          long    prot;          /* new protection */
```

FORTRAN Subroutine Declaration:

```
c                                ! change protection of a memory area
          subroutine setmpr(mname, prot, status)
              character*94 mname  ! name of memory area
              integer*4 prot      ! new protection
              integer*4 status    ! result of the operation
```

PASCAL Procedure Declaration:

```
    procedure _setmprt(          {** change protection of a memory area}
          mname   : string[93];  {** name of memory area}
          prot    : longint;     {** new protection }
       var status : longint      {** result of the operation}
    ); external;
```

setmuic - Set named memory area uic.

Description:

_setmuic is used to change the user identification code (uic) of a named sharable memory area.

To successfully change the uic of a named sharable memory area the calling process must have operator privilege, and either group privilege or world privilege.

If the calling process has group privilege and operator privilege, and the group id of the named sharable memory area is the same as the group id of the calling process, the process can modify the owner id of the named sharable memory area.

If the calling process has world privilege and operator privilege it can change the uic of any named sharable memory area to be any other uic except zero.

Related Privileges:

none	- Does not allow changing the UIC of a named shared memory area. Note: _setmuic will return successfully if the specified UIC is the same as the current UIC.
group	- If the process also has operator privilege, and the group id of the process is the same as the group id of the specified named sharable memory area, the process is allowed to modify the owner id portion of the uic.
operator	- Allows the process to change the UIC of any named shared memory area if the process also has either group or world privilege.
world	- If the process also has operator privilege, the process is allowed to modify the uic of the named sharable memory area to any uic except zero.

Parameters:

mname	- Address of a null terminated string identifying the specific memory area. This string will be translated automatically by WMCS into its logical equivalent. This string may contain up to 93 significant characters followed by a null.
uic	- A long word containing the UIC number of the new owner of the named shareable memory area.
status	- Address of a long word to receive the result of

the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
errnoname	(82)	The name specified does not exist.

See Also:

_defmem	-	Define a named sharable memory area.
_udefmem	-	Undefine a named sharable memory area.
_shrmem	-	Share a named sharable memory area.
_ushrmem	-	Unshare a named sharable memory area.
_getmlst	-	Get a list of named sharable memory areas.
_setmprt	-	Change protection of a named sharable memory area.

Assembler Calling Sequence:

```
            push    mname           ; address - name of memory area
            push    uic             ; value   - user identification code
            push    status          ; address - result of the operation
            jsr     _setmuic        ; Set named memory area uic.
```

C Function Declaration:

```
                                    /* set named memory area uic */
            long                    /* returns result of the operation */
            _setmuic(mname,uic)
                char mname[94];     /* name of memory area */
                long uic;           /* user identification code */
```

FORTRAN Subroutine Declaration:

```
    c                               ! set named memory area uic
            subroutine setmui(mname, uic, status)
                character*94 mname  ! name of memory area
                integer*4 uic       ! user identification code
                integer*4 status    ! result of the operation
```

PASCAL Procedure Declaration:

```
    procedure  setmuic(              {** set named memory area uic}
            mname   : string[93];    {** name of memory area}
            uic     : longint;       {** user identification code}
        var status  : longint        {** result of the operation}
    ); external;
```

Change process name.

Description:

Allows a process to set its own process name or give another process a new process name. The calling process must have operator privilege to change the process name.

Related Privileges:

None	- The calling process can not change the process name on any process.
operator	- The calling process can change the process name on any process with the same owner id and group id (uic).
group	- If the calling process has operator privilege it can change the process name of any process with the same group id.
world	- If the calling process has operator privilege it can change the process name of any process in the system.

Parameters:

pid	- Process ID of the process whose process name is to be changed. 0 refers to the calling process, -1 refers to the parent of the calling process.
pname	- Address of a 17 byte null terminated string containing the new process name to be given to the specified process. (up to 16 valid characters followed by a null)
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
errprcsnotfnd	(2)	The specified process is not in the system process table.

See Also:

_crproc	- Create a new process
_getpcb	- Get Process Control Block
_getpnam	- Get process name from pid

Assembler Calling Sequence:

```
push    pid                             ;value - process id
```

```
    push    pname                        ;address - process name
    push    status                       ;address - result of the operation
    jsr     _setpnam                     ;Change process name
```

C function declaration:

```
                                         /* change process name */
    long                                 /* returns result of the operation */
    _setpnam(pid, pname)
            long pid;                    /* process id */
            char pname[17];              /* process name */
```

Fortran Subroutine Declaration:

```
    c                                    ! change process name
            subroutine setpna(pid, pname, status)
                integer*4 pid            ! process id
                character*17 pname       ! process name
                integer*4 status         ! result of the operation
```

Pascal Procedure Declaration:

```
    procedure _setpnam(                  {** change process name}
            pid     : longint;           {** process id}
            pname   : string[16];        {** process name}
        var status  : longint            {** result of the operation}
    ); external;
```

t

Set the current file position.

Description:

Given a valid logical unit number (lun), sets the default position of the file pointer in an open file. If the next file access uses the default record number (-1) the transfer will begin at this file position.

This is the complementary operation to _getpos.

Note that this system call is not required for random file access on disk since all _read and _write system calls allow the process to specify a record number. It is, however, the only method of achieving pseudo random access on a tape. Setting the file position with this system call will position the tape correctly to the specified record within the file.

Related Privileges:

None.

Parameters:

lun	- A long word containing the logical unit number of the file whose position is to be set.
recnum	- The record number to which the file position is to be set. This is an unsigned long word.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinvlfn	(132)	The logical unit number does not correspond to an open file.
errreadleof	(140)	The process tried to read past the logical end of a file.
		Device integrity errors

See Also:

_getpos	- Get the current file position
_read	- Read from an open file
_write	- Write to an open file

Assembler Calling Sequence:

```
push    lun                          ;value - logical unit number
```

```
   push    recnum                       ;value - record number
   push    status                       ;address - result of the operation
   jsr     _setpos                      ;set the current file position
```

C function declaration:

```
                                        /* set the current file position */
   long                                 /* returns result of the operation */
   _setpos(lun, recnum)
           long lun;                    /* logical unit number */
           long recnum;                 /* record number */
```

Fortran Subroutine Declaration:

```
   c                                    ! set the current file position
           subroutine setpos(lun, recnum, status)
               integer*4 lun            ! logical unit number
               integer*4 recnum         ! record number
               integer*4 status         ! result of the operation
```

Pascal Procedure Declaration:

```
   procedure _setpos(                   {** set the current file position}
           lun     : longint;           {** logical unit number}
           recnum  : longint;           {** record number}
       var status  : longint            {** result of the operation}
   ); external;
```

Change a process's priority.

Description:

Allows a process to set its own scheduler priority or the priority of another process. There are 16 priority levels numbered 0..15. Priority level 0 is the highest.

A process may lower the priority of any process which it can affect, but it must have setprior privilege in order to increase the priority of any process.

Related Privileges:

none	- Allows the process to affect the priority of any process with the same owner id and group id (uic) as the calling process.
group	- Allows the process to affect the priority of any process with the same group id as the calling process.
world	- Allows the process to affect the priority of any process in the system.
setprior	- Allows the process to raise the priority of any process which it can affect.

Parameters:

pid	- A long word containing the process ID of the process whose priority is to be changed. 0 refers to the current process, -1 refers to the parent of the current process.
priort	- A long word containing the priority level (0..3) desired. Uses this value modulo 4 so that no out of range errors are detected. If the value of this parameter is -1 ($FFFFFFFF), it means to use the same priority of the calling process.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
errprcsnotfnd	(2)	The specified process is not in the system process table.

See Also:

_getpri	- Get process's priority
_prirat	- Set priority scheduling ratio
_settmsl	- Change scheduling time slice

Assembler Calling Sequence:

```
push    pid                     ;value - process id
push    priort                  ;value - new priority level
push    status                  ;address - result of the operation
jsr     _setpri                 ;change process's priority
```

C Function Declaration:

```
                                /* change process's priority */
long                            /* returns result of the operation */
_setpri (pid, priort)
        long pid;               /* process id */
        long priort;            /* new priority level */
```

FORTRAN Subroutine Declaration:

```
c                               ! change process's priority
        subroutine _setpri(pid, priort, status)
            integer*4 pid       ! process id
            integer*4 priort    ! new priority level
            integer*4 status    ! result of the operation
```

Pascal Procedure Declaration:

```
procedure _setpri(              {** change process's priority }
        pid     : longint;      {** process id}
        priort  : longint;      {** new priority level}
    var status  : longint       {** result of the operation}
); external;
```

Set process privilege.

Description:

Allows a process to acquire and relinquish various privileges as assigned by the process privilege word. A process must have setpriv privilege in order to assign privileges which it does not already have.

Related Privileges:

none	- Allows the process to modify privileges of processes with the same owner id and group id (uic) as the calling process.
group	- Allows the process to modify privileges of processes with the same group id as the calling process
setpriv	- Allows the process to assign new privileges to processes which it can affect
world	- Allows the process to modify privileges of any process in the system

Parameters:

pid - A long word containing the process id of the process whose privileges are to be changed. A pid of 0 represents the current process. A pid of -1 represents the parent of the current process.

priv - A long word containing the privilege mask, a bit mask of privileges to be given to the specified process. If the value of this parameter is -1, the specified process is given the same privileges as the calling process. If the value of this parameter is not -1, it represents privileges which are bit encoded as follows:

Bit Name	Bit	Description
pcbpvsetpriv	0	setpriv
pcbpvsystem	1	system
pcbpvreadphys	2	readphys
pcbpvwritephys	3	writephys
pcbpvsetprior	4	setprior
pcbpvchngsuper	5	chngsuper
pcbpvbypass	6	bypass
pcbpvoperator	7	operator

pcbpvaltuic	8	altuic
pcbpvworld	9	world
pcbpvgroup	10	group
pcbpvnetwork	11	network
pcbpvsetattr	12	setattr
	13-32	Reserved. Must be set to zero

status - Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
errprcsnotfnd	(2)	The specified process is not in the system process table.

See Also:

_getprv - Get process privilege
_settmsl - Change scheduling time slice

Assembler Calling Sequence:

```
push    pid                       ;value - process id
push    priv                      ;value - new privilege mask
push    status                    ;address - result of the operation
jsr     _setprv                   ;set process privilege
```

C Function Declaration:

```
                                  /* set process privilege */
long                              /* returns result of the operation */
_setprv(pid, priv)
         long pid;                /* process id */
         long priv;               /* new privilege mask */
```

FORTRAN Subroutine Declaration:

```
c                                 ! set process privilege
         subroutine _setprv(pid, priv, status)
             integer*4 pid        ! process id
             integer*4 priv       ! new privilege mask
             integer*4 status     ! result of the operation
```

Pascal Procedure Declaration:

```
    procedure _setprv(              {** set process privilege}
            pid     : longint;      {** process id}
            priv    : longint;      {** new privilege mask}
        var status  : longint       {** result of the operation}
    ); external;
```

Set/clear real time mode flag.

Description:

Allows a process with setprior privilege to set or clear the realtime mode flag in the process control block of the current process. If the real time bit is set, context switches to the next process will not occur until the process voluntarily relinquishes control. Note that doing an I/O operation that requires the process to wait until the I/O is complete will also cause the process to relinquish control. The time slice interrupt clock is ineffectual for a process in real-time mode.

Related Privileges:

none	- Allows the calling process to clear the realtime mode flag. Note that this is not especially useful unless the process can also set the realtime mode flag.
setprior	- Allows the calling process to set or clear the realtime mode flag.

Parameters:

mode	- A long word containing the realtime mode flag. A value of 0 will clear the realtime mode flag. Non-zero values set the flag.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.

See Also:

_prirat	- Set the priority scheduling ratio
_setpri	- Set process's priority
_settmsl	- Change scheduling time slice

Assembler Calling Sequence:

```
push    mode                        ;value - real time flag
push    status                      ;address - result of the operation
jsr     _setrtm                     ;set/clear real time mode flag
```

C function declaration:

```
                                        /* set/clear real time mode flag */
    long                                /* returns result of the operation */
    _setrtm(mode)
            long mode;                  /* real time flag */
```

Fortran Subroutine Declaration:

```
    c                                   ! set/clear real time mode flag
            subroutine setrtm(mode, status)
                integer*4 mode          ! real time flag
                integer*4 status        ! result of the operation
```

Pascal Procedure Declaration:

```
    procedure _setrtm(                  {** set/clear real time mode flag}
            mode    : longint;          {** real time flag}
        var status  : longint           {** result of the operation}
    ); external;
```

Assign devicenames to a rotor list.

Description:

This call is used to define a rotor list. A rotor list is a list of devices which share a set of generic characteristics. The term "rotor" is derived from the telephone industry, where a set of telephone lines is assigned to a customer. Although each line has a specific telephone number assignment, any available line may be used by dialing the number of the first line in the rotor group. Upon receipt of an incoming call to a number in the rotor group (which is actually a request to use a free line in the rotor group) the telephone company automatically searches for a free line and either assigns it to the incoming phone call or, should there be no free lines, returns an error signal to the caller (a busy signal).

Rotor lists are useful under WMCS when a group of similar devices is provided as a pool of resources, such as a set of modem lines, a set of identical printer lines, etc. An example may be a situation where several modem lines are available for outgoing calls on a system. Rather than writing device specific software or determining status on each modem line before attempting to use it, the system manager may wish to place all outgoing modem lines together in a rotor list. The software can then call the alloc system call using the rotor list name as its argument. If any modem line is free (and the specified process has appropriate access to it), the line will be reserved for the specified process and the name of the specific device will be returned to the calling process.

The first name provided in the input list is used as the rotor list name. This name may be up to 93 characters and will be logically translated before devices are assigned to it. Only the first 8 significant characters of the logical name translation will be retained by WMCS. The devicenames follow the rotor list name and are separated from the rotor list name and from each other by commas. The devicenames to be inserted into the rotor name list are logically translated before they are used. Imbedded spaces are illegal. If the first name in the list (the rotor list name) is found to already exist, the previous list is discarded and the new list takes its place. A rotor list may be deleted by setting the rotor list name to have no list elements.

Related Privileges:

none	- The process cannot assign a list of devicenames to a rotor list.
operator	- Allows assignment of a list of devicenames to a rotor list.

Parameters:

rtrnam	- Address of a null terminated string identifying the rotor list name.
rtrlst	- Address of a null terminated string identifying the devices which are to be assigned to the rotor list. Each name in the string is separated from the others by a comma. Each name in the string will be translated automatically by WMCS into its logical equivalent. Each element in this list may contain up to 93 significant characters but must translate to a name of not more than 8 characters.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
errnomemavail	(7)	All available memory has been allocated.
errnamenull	(80)	The specified name must not be null.
errnoname	(82)	The specified name does not exist.

See Also:

_alloc	- Allocate an available device
_dealloc	- Deallocate an allocated device
_getalc	- Get names of allocated devices
_getrel	- Get names of rotor list elements
_getrtr	- Get rotor list names

Assembler Calling Sequence:

```
push    rtrnam                  ;address - name of rotor
push    rtrlst                  ;address - name of rotor devices
push    status                  ;address - result of the operation
jsr     _setrtr                 ;assign devicenames to a rotor list
```

C Function Declaration:

```
                                    /* assign devicenames to rotor list */
    long                            /* returns result of the operation */
    _setrtr(rtrnam, rtrlst)
            char rtrnam[1024];      /* name of rotor */
            char rtrlst[1024];      /* name of rotor devices */
            long index;             /* index into table */
```

FORTRAN Subroutine Declaration:

```
    c                               ! assign devicenames to a rotor list
            subroutine _setrtr(rtrnam, rtrlst, status)
               character*1024 rtrnam ! name of rotor
               character*1024 rtrlst ! name of rotor devices
               integer*4 status     ! result of the operation
```

Pascal Procedure Declaration:

```
    procedure _setrtr(              {** assign devicenames to rotor list}
           rtrnam  : string[1024]; {** name of rotor}
           rtrlst  : string[1024]; {** name of rotor devices}
       var status  : longint        {** result of the operation}
    ); external;
```

Set system date and time.

Description:

Allows a process with OPERATOR privilege to set the system time-of-day clock. The time is specified in 8 bytes. Those fields of the time that exceed the maximum value for that field are truncated. The format of the date and time within these 8 bytes is as follows, where byte 0 is the most significant byte.

Bytes	Description
0,1	The current year (counted from A.D. 0). Example, 1983.
2,3	The day of the year (1..365 or 1..366)
4	The hour of the day (0..23)
5	The minute of the hour (0..59)
6	The second of the minute (0..59)
7	The fraction of a second (in 100ths of a second) (0..99)

Related Privileges:

none	- Process not allowed to set the date and time
operator	- Allows process to successfully set the system clock

Parameters:

siteid	- A long word containing the system id of the system whose clock is to be set. A siteid of zero corresponds to the system on which the calling process is executing.
mstime	- Most significant 32 bits of the clock in system time format. Contains the year and day portions of the clock.
lstime	- Least significant 32 bits of the clock in system time format. Contains the hour, minute, second and fraction of a second portion of the clock.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
errinvsiteid	(8)	The specified site id does not exist.

See Also:

_gettic - Get internal tick count
_gettim - Get the current date and time

Assembler Calling Sequence:

```
push    siteid                   ;value - system id
push    mstime                   ;value - day and year
push    lstime                   ;value - hour, minute, second, tick
push    status                   ;address - result of the operation
jsr     _settim                  ;set system date and time
```

C function declaration:

```
                                 /* set system date and time */
long                             /* returns result of the operation */
_settim(siteid, mstime, lstime)
        long siteid;             /* system id */
        long mstime;             /* day and year */
        long lstime;             /* hour, minute, second, tick */
```

Fortran Subroutine Declaration:

```
c                                ! set system date and time
        subroutine settim(siteid, mstime, lstime, status)
            integer*4 siteid     ! system id
            integer*4 mstime     ! day and year
            integer*4 lstime     ! hour, minute, second, tick
            integer*4 status     ! result of the operation
```

Pascal Procedure Declaration:

```
procedure _settim(               {** set system date and time}
        siteid  : longint;       {** system id}
        mstime  : longint;       {** day and year}
        lstime  : longint;       {** hour, minute, second and tick}
    var status  : longint        {** result of the operation}
); external;
```

Change scheduling time slice.

Description:

Change the scheduling time slice of a process. Time slice is the maximum amount of time the non-real time process will be allowed to execute each time it is scheduled. When the time slice is expired, other processes are allowed to execute according to the scheduling algorithm.

Each time slice increment is .01 milliseconds. A time slice value of 5000 allows the process to execute up to one twentieth of a second (50 milliseconds) each time it is scheduled. A time slice value less than 10 results in the process not running at all.

Note that processes will not always use their full time slice. When an I/O operation is performed, the process often relinquishes control and loses the rest of its time slice.

Any process can lower the time slice of all processes that it can affect. setprior privilege is required to increase the time slice value of any process.

Related Privileges:

None	- Allows affecting the time slice of any process with the same owner id and group id (uic) as the calling process.
group	- Allows affecting the time slice of any process with the same group id as the calling process.
world	- Allows affecting the time slice of any process whatsoever.
setprior	- Allows increasing the time slice of any process which the current process can affect.

Parameters:

pid	- A long word containing the process id of the process whose time slice is to be changed. 0 represents the current process; -1 ($FFFFFFFF) represents the parent of the current process.
tslice	- A long word containing the new time slice value (0..65535). A long word value of -1 ($FFFFFFFF) is a keyword value that means to use the same time slice as the calling process.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
errprcsnotfnd	(2)	The specified process is not in the system process table.

See Also:

_prirat - Set priority scheduling ratio
_setpri - change process's priority

Assembler Calling Sequence:

```
push    pid                         ;value - process id
push    tslice                      ;value - new time slice
push    status                      ;address - result of the operation
jsr     _settmsl                    ;change scheduling time slice
```

C function declaration:

```
                                    /* change scheduling time slice */
long                                /* returns result of the operation */
_settmsl(pid, tslice)
        long pid;                   /* process id */
        long tslice;                /* new time slice */
```

Fortran Subroutine Declaration:

```
c                                   ! change scheduling time slice
        subroutine settms(pid, tslice, status)
            integer*4 pid           ! process id
            integer*4 tslice        ! new time slice
            integer*4 status        ! result of the operation
```

Pascal Procedure Declaration:

```
procedure _settmsl(                 {** change scheduling time slice}
        pid     : longint;          {** process id}
        tslice  : longint;          {** new time slice}
    var status  : longint           {** result of the operation}
); external;
```

settrp - Initialize a user defined trap.

Description:

A "trap" is a software invoked interrupt. It can be fatal or non-fatal. Traps caused by attempting to execute privileged instructions in user mode, illegal instructions, address traps, and bus traps are fatal traps. When a fatal trap occurs, the OS deletes the process that caused it.

Non-fatal traps include the sixteen TRAP instructions, the CHK, TRAPV, and emulation instructions, and the divide by zero trap.

When a non-fatal trap occurs, the OS checks for a "user defined" trap handler. If there is one, control transfers to this trap handler where the process is allowed to handle the condition which caused the trap. The trap handler is execute in the same processor mode (user or supervisor mode) as the mode from which the trap handler was defined or was executed, which ever is higher. The return address and status register will be on the top of the stack when the trap handling routine is entered. Use the 'RTR' or 'RTE' instruction to return from a user defined trap.

If no trap handler has been defined, the OS treats it as a fatal error and terminates the process.

This system service routine allows a user process to define its own trap handling routines which can be used to handle non-fatal trap conditions.

Related Privileges:

None.

Parameters:

trap - The number of the trap for which a handler is being defined. Traps 0 and 1 are reserved for use by the OS. They may not be redefined. Traps 14 and 15 are reserved for the OS debugger. The 1010 emulation handler is used by some languages for floating point. Redefining the 1010 emulation disables the OS floating point support. The valid trap numbers are:

Trap #	Description

0	Reserved. (TRAP 0)
1	Reserved. (TRAP 1)
2	Corresponds to the "TRAP 2" instruction
3	Corresponds to the "TRAP 3" instruction
4	Corresponds to the "TRAP 4" instruction
5	Corresponds to the "TRAP 5" instruction
6	Corresponds to the "TRAP 6" instruction
7	Corresponds to the "TRAP 7" instruction
8	Corresponds to the "TRAP 8" instruction
9	Corresponds to the "TRAP 9" instruction
10	Corresponds to the "TRAP 10" instruction
11	Corresponds to the "TRAP 11" instruction
12	Corresponds to the "TRAP 12" instruction
13	Corresponds to the "TRAP 13" instruction
14	Reserved. (TRAP 14)
15	Reserved. (TRAP 15)
16	Divide by zero trap number
17	Bounds checking trap (CHK instruction)
18	Check overflow trap (TRAPV instruction)
19	Trace trap
20	1010 instruction emulation
21	1111 instruction emulation
22	Exit handler trap handler. A call to SETEXIT go to this handler instead of defining a new exit handler.
23	Floating point interrupt trap handler.

All other values reserved.

adr - The address of the trap handler routine. The entry point to which control should be transferred when the trap occurs. A zero in this parameter means that the trap is not to be handled by the user. That is, specifying zero for this parameter "undefines" the trap. Note that this address must be in the user process area ($000000 through $1FFFFF).

status - Address of a long word to receive the result of the operation.

Diagnostics:

errbadtrapnum (15) Trap number (during _SETTRAP) exceeds range of specifiable numbers.

See Also:

None.

Assembler Calling Sequence:

```
push    trap                          ;value - trap number
push    adr                           ;value - address of handler routine
```

```
    push    status                          ;address - result of the operation
    jsr     _settrp                         ;initialize a user defined trap
```

C function declaration:

```
                                            /* initialize a user defined trap */
    long                                    /* returns result of the operation */
    _settrp(trap, adr)
            long trap;                      /* trap number */
            long adr;                       /* address of handler routine */
```

Fortran Subroutine Declaration:

```
    c                                       ! initialize a user defined trap
            subroutine settrp(trap, adr, status)
                integer*4 trap              ! trap number
                external adr                ! address of handler routine
                integer*4 status            ! result of the operation
```

Pascal Procedure Declaration:

```
    procedure  settrp(                      {** initialize a user defined trap}
            trap    : longint;              {** trap number}
            adr     : longint;              {** address of handler routine}
        var status  : longint               {** result of the operation}
    ); external;
```

Set process UIC.

Description:

Allows a process to set its own user identification code (uic) or the uic of another process. The calling process must have operator privilege to affect the uic.

No check is made that the resulting uic belongs to a user with an account in the user authorization file.

Related Privileges:

None	- The calling process can not change the uic on any process.
operator	- If the calling process also has group or world privilege it can affect the uic of processes as described below.
group	- If the calling process has operator privilege it can change the owner id portion of the uic of any process with the same group id.
world	- If the calling process has operator privilege it can change the uic of any process in the system to any value whatsoever. (A value of zero is not allowed)

Parameters:

pid	- A long word containing the process ID (pid) of the process whose uic is to be changed. 0 refers to the calling process, -1 refers to the parent of the calling process.
uic	- A long word containing the uic that the specified process will receive. The most significant word (16 bits) of this parameter correspond to the owner id and the least significant word corresponds to the group id. A long word -1 ($FFFFFFFF) is a reserved value that means to use the default uic, i.e. the uic of the calling process. A value of zero for this parameter is not allowed.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
errprcsnotfnd	(2)	The specified process is not in the system

process table.

See Also:

_getpcb - Get Process Control Block

Assembler Calling Sequence:

```
push    pid                 ;value - process id
push    uic                 ;value - new uic
push    status              ;address - result of the operation
jsr     _setuic             ;Set process uic
```

C function declaration:

```
                            /* set process uic */
long                        /* returns result of the operation */
_setuic(pid, uic)
        long pid;           /* process id */
        long uic;           /* new uic */
```

Fortran Subroutine Declaration:

```
c                           ! set process uic
        subroutine setuic(pid, uic, status)
            integer*4 pid       ! process id
            integer*4 uic       ! new uic
            integer*4 status    ! result of the operation
```

Pascal Procedure Declaration:

```
procedure _setuic(          {** set process uic}
        pid     : longint;  {** process id}
        uic     : longint;  {** new uic}
    var status  : longint   {** result of the operation}
); external;
```

shrmem - Share a named shared memory area.

Description:

A process uses _shrmem to initiate memory sharing using named sharable memory areas. Named sharable memory areas are sections of system memory which have an associated name. Using this name, a process may request that this section of memory be mapped into its logical memory space which extends from address $00001000 through address $001fefff. The size of these memory areas will be some multiple of the hardware page size which is 4K bytes.

To successfully share a named memory area the process must have read and/or write privilege to the named memory area.

Related Privileges:

none	- Allows a process to share a named memory area if the process has read and/or write privilege to the named memory area.
altuic	- Allows the process to share a named memory area if the owner of the process's image file has read and/or write privilege to the named memory area.
bypass	- Allows the process to share a named memory area regardless of the memory protection mask.
system	- Allows the process to share a named memory area if the system has read/write privilege to the named memory area.

Parameters:

mname	- Address of a null terminated string identifying the specific memory area. This string will be translated automatically by WMCS into its logical equivalent. This string may contain up to 93 significant characters followed by a null.
adr	- A long word address within the user logical address space where the shared area is to appear. It is an error to give an address which does not begin on a hardware page boundary. It is an error if memory is already allocated at this location.
size	- A long word containing the number of bytes of memory to be shared. It is not an error if SIZE is not an even multiple of the hardware page size. The size parameter may also be less the defined size of the named memory area.

retlen	- Address of a long word to receive the number of bytes actually allocated. If SIZE is greater or equal to the size of the shared memory area, then the size of the the shared memory area is returned. Otherwise, if SIZE is sufficiently large that at least one hardware page can be shared then SIZE is returned and an warning is given. If nothing can be shared, an error is returned. It is an error to already have memory allocated in the logical space where the shared memory area is to reside.
mode	- A long word which specifies the desired useage of the area: read, write, or execute. There is no hardware facility for enforcing the distinction between read and execute privileges. Protection of the memory area is enforced as in the file system.

Bit Name	Bit #	Description
opreadacc	0	read access - requests permission for read access to the named shared memory
opwriteacc	1	write access - requests permission for write access to the named shared memory

timout	- A long word which specifies an amount of time the process can wait for the shared memory area to appear. If the memory area specified by mname is not defined before the expiration of timout, an error condition exists.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinvadr	(4)	The memory address is not on a 4K page boundary.
errmemalloc	(5)	The process requested a logical page that was was already allocated.
errsizovfl	(60)	The size passed to WMCS is out of range.
errnoname	(82)	The name specified does not exist.
errtimeout	(128)	A request was not completed within the specified time.
errnoreadpriv	(144)	The process does not have read privilege for the file.
errnowritepriv	(145)	The process does not have write privilege for the file.

See Also:

_defmem	- Define a named sharable memory area.
_udefmem	- Undefine a named sharable memory area.
_ushrmem	- Unshare a named sharable memory area.
_getmlst	- Get a list of named sharable memory areas.

```
_setmuic  -  Change owner of a named sharable memory area.
_setmprt  -  Change protection of a named sharable memory area.
```

Assembler Calling Sequence:

```
    push    mname                   ; address - name of memory area
    push    adr                     ; value   - address of memory area
    push    size                    ; value   - size of memory area
    push    retlen                  ; address - amount actually shared
    push    mode                    ; value   - access mode (read, write)
    push    timout                  ; value   - time out
    push    status                  ; address - result of the operation
    jsr     _shrmem                 ; Share a named shared memory area.
```

C Function Declaration:

```
                                    /* share a named shared memory area */
        long                        /* returns result of the operation */
        _shrmem(mname. adr, size, retlen, mode, timout)
            char mname[94];         /* name of memory area */
            long adr;               /* address of memory area */
            long size;              /* size of memory area */
            long *retlen;           /* amount actually shared */
            long mode;              /* access mode (read, write) */
            long timout;            /* time out */
```

FORTRAN Subroutine Declaration:

```
    c                               ! share a named shared memory area
        subroutine shrmem(mname. adr, size, retlen, mode, timout.
      &         status)
            character*94 mname      ! name of memory area
            integer*4   adr         ! address of memory area
            integer*4   size        ! size of memory area
            integer*4   retlen      ! amount actually shared
            integer*4   mode        ! access mode (read, write)
            integer*4   timout      ! time out
            integer*4   status      ! result of the operation
```

PASCAL Procedure Declaration:

```
    procedure  shrmem(              {** share a named shared memory area}
            mname   : string[93];   {** name of memory area }
            adr     : longint;      {** address of memory area }
            size    : longint;      {** size of memory area }
        var retlen  : longint;      {** amount actually shared }
            mode    : longint;      {** access mode (read, write) }
            timout  : longint;      {** time out }
        var status  : longint       {** result of the operation}
    ); external;
```

Return a list of all known site ID numbers.

Description:

Return a list of site ID numbers and the total number of site ID numbers known in the network.

Related Privileges:

None.

Parameters:

sidlst	- Address of buffer to receive the site IDs known about in the network. This buffer must be word aligned.
len	- Maximum number of site IDs that can be contained in the sidlst buffer
retlen	- Address of a long word to receive the number of site IDs that were written into sidlst.
total	- Address of a long word to receive the total number of site IDs known about in the system. This number may be greater that the number returned in retlen.

Diagnostics:

None.

See Also:

_getnnam - Get the name of a node
_getnsid - Get the site ID of a node
_rnidlst - Get list of remote network IDs
_rsidlst - Get list of site IDs from a remote network

Assembler Calling Sequence:

```
push    sidlst                  ;address - siteid buffer
push    len                     ;value - length of sidlst
push    retlen                  ;address - number of siteids returned
push    total                   ;address - total number of siteids
jsr     _sidlst                 ;get list of site ids
```

C Function Declaration:

```
                                   /* get list of known site ids */
void                               /* no result */
_sidlst(sidlst, len, retlen, total)
        long *sidlst;              /* siteid buffer */
        long len;                  /* length of sidlst */
        long *retlen;              /* number of siteids returned */
        long *total;               /* total number of site ids */
```

FORTRAN Subroutine Declaration:

```
c                                  ! get list of known site ids
      subroutine _sidlst(sidlst, len, retlen, total)
            integer*4 sidlst       ! siteid buffer
            integer*4 len          ! length of sidlst
            integer*4 retlen       ! number of siteids returned
            integer*4 total        ! total number of site ids
```

Pascal Procedure Declaration:

```
procedure _sidlst(                 {** get list of known site ids}
       sidlst  : ^array_of_char; {** siteid buffer}
       len     : longint;          {** length of sidlst}
   var retlen  : longint;          {** number of siteids returned}
   var total   : longint           {** total number of site ids}
); external;
```

Set device status with LUN.

Description:

Allows a process to modify a device status table.

The device status is a device class dependent 128 byte table. It is maintained by the device driver for each device.

NOTE: The device status table may change with each release of the operating system. The current definition is included in each release in the file named: /SYSINCL.SYS/DSTATDISP.*. The name of the record included in that file is "devicestatus," i.e., in your program you can declare a variable whose type is "devicestatus".

The device status table is divided into two parts. The first half is device independent and is composed of the following fields:

Name	Length (bytes)	Description
dsclassid	2	The device class. Valid classes are: (Note that these names are defined in the devtdisp.* files)

Class Name	Value	Description
dtclassttyspc	0	Character device (ttyspc)
dtclasstty	1	Character device (tty)
dtclasstapespc	2	Tape device (tapespc)
dtclasstape	3	Tape device (tape)
dtclassdiskspc	4	Disk device (diskspc)
dtclassdisk	5	Disk device (disk)
dtclassnetspc	6	Network dev. (networkspc)
dtclassnet	7	Network device (network)
dtclasspipespc	8	Pipe device (pipespc)
dtclasspipe	9	Pipe device (pipe)
dtclasssyncspc	10	BSC device (syncspc)
dtclasssync	11	BCS device (sync)
dtclassquespc	12	Queue device (quespc)

		dtclassque 13 Queue device (que)
		dtclassnondevspc 14 Non-dev device(nondevspc)
		dtclassnondev 15 Non-dev device (nondev)
dsdriverid	2	The unique ID number for this device driver
dsblksz	2	The block size of the device (e.g. sector size)
dsharderr	2	The hard error count for the device
dssofterr	2	The soft error count for the device
dsreadoper	4	The number of read operations on this device
dswriteoper	4	The number of write operations on this device
dsmaxnumdev	2	Maximum # of devices this driver can handle
dscurnumdev	2	# of mounted devices using this driver
dsnumtoretry	2	# of times to retry before reporting an error
dserrorreason	4	Hardware error code for last error received on this device
dsreserved	32	Reserved
dsnexttableptr	4	Address of next device status table

The second half of the device status table is device class dependent. For TAPE class devices the second half is defined as follows:

Name	Length (bytes)	Description		
dstpstatus	2	Tape device status. A bit encoded word.		
		Bit name	bit #	Description
		dstpready	0	Set if device ready
		dstpintpend	1	Set if interrupt pending
		dstprewinding	2	Set if tape rewinding
		dstpbotdetect	3	Set if device is at physical BOT
		dstpeotdetect	4	Set if device is at physical EOT
		dstpwriteprot	5	Set if tape is write protected
dstpflagsl	2	Tape status information. A bit encoded word.		
		Bit name	bit #	Description
		dstpdoraw	0	0=Read after write disabled 1=Read after write enabled
		dstperrintenb	1	0=Error interrupts are enabled 1=Error interrupts are disabled

dstpspeed	1	Tape speed. Values are: 0 - Reserved dstpspeed12ips 1 - 12 ips dstpspeed25ips 2 - 25 ips dstpspeed30ips 3 - 30 ips dstpspeed50ips 4 - 50 ips dstpspeed90ips 5 - 90 ips dstpspeed100ips 6 - 100 ips dstpspeed125ips 7 - 125 ips
dstpdensity	1	Tape density. Values are: 0 - Reserved dstpdens800bpi 1 - 800 bpi dstpdens1600bpi 2 - 1600 bpi dstpdens3200bpi 3 - 3200 bpi dstpdens6250bpi 4 - 6250 bpi dstpdens6400bpi 5 - 6400 bpi
dstpreserved	50	Reserved
dstpuserfield	8	User defined status

For DISK class devices the second half of the device status table is defined as follows:

Name	Length (bytes)	Description
dsdkintfac	2	Disk interleave factor
dsdkiopbcnt	2	Number of IOPB's allocated to the drive
dsdknumbsect	4	The number of sectors on the volume
dsdksectrack	2	The number of sectors on a track
dsdkheads	2	The number of heads on the device
dsdkcylinders	2	The number of cylinders on the volume
dsdkflags1	2	Disk status information. A bit encoded word.

Bit Name	Bit #	Description
dsdkdensity1	0	Device density
dsdkdensity2	1	
dsdkdenssignle		00 - Single density
dsdkdensdouble		01 - Double density
dsdkdensquad		10 - Quad density
dsdkdensreserve		11 - Reserved
dsdkdoraw	3	If set, do Read after write verify
dsdkwriteprot	4	If set, Device write protected

		dsdkseekdir 15 Current seek direction
		dsdkseekincr — 0 - Increasing cylinder numbers
		dsdkseekdecr — 1 - Decreasing cylinder numbers
dsdkcurcyl	2	Current cylinder position
dsdkcachesz	2	Number of sectors in the disk cache
dsdkentryname	16	Null terminated string containing the name of this type of drive
dsdkreserved	20	Reserved
dsdkuserfield	8	User Defined status

For TTY class devices the second half of the device status table is defined as follows:

Name	Length (bytes)	Description
dstymoderegl	1	Uart mode register 1. This byte is bit encoded as follows:

Bit Name	Bit #	Description
dstymrlbaudfacl	0	Baud factor
dstymrlbaudfac2	1	
dstymrlsyncl		00 - sync 1 x clock rate
dstymrlasyncl		01 - async 1 x clock rate
dstymrlasync16		10 - async 16 x clock rate
dstymrlasync64		11 - async 64 x clock rate
dstymrlcharlenl	2	Character length
dstymrlcharlen2	3	definition
dstymrldw5bit		00 - 5 data bits
dstymrldw6bit		01 - 6 data bits
dstymrldw7bit		10 - 7 data bits
dstymrldw8bit		11 - 8 data bits
dstymrlparityctrl	4	Parity control
dstymrlpardis		0 - disable parity
dstymrlparenb		1 - enable parity
dstymrlparitytype	5	Parity type
dstymrlparodd		0 - odd parity
dstymrlparevn		1 - even parity
dstymrlstopbitsl	6	Async mode # of stop bits

		dstymr1stopbits2	7	Async mode # of stop bits
		dstymr1binv		00 - invalid
		dstymr1sb1		01 - 1 stop bit
		dstymr1sb15		10 - 1.5 stop bits
		dstymr1sb2		11 - 2 stop bits
		dstymr1transctrl	6	Sync mode transparent
		dstymr1normal		0 - normal
		dstymr1trans		1 - transparent
		dstymr1numsync	7	Sync mode # of syncs
		dstymr1syncdouble		0 - double sync
		dstymr1syncsingle		1 - single sync
dstymodereg2	1	Uart mode register 2. This byte is bit encoded as follows:		
		Bit Name	Bit #	Description
		dstymr2baudrt1	0	The baud rate
		dstymr2baudrt2	1	Baud rate continued
		dstymr2baudrt3	2	Baud rate continued
		dstymr2baudrt4	3	Baud rate continued
		dstymr2baud50		0000 - 50 baud
		dstymr2baud75		0001 - 75 baud
		dstymr2baud110		0010 - 110 baud
		dstymr2baud1345		0011 - 134.5 baud
		dstymr2baud150		0100 - 150 baud
		dstymr2baud300		0101 - 300 baud
		dstymr2baud600		0110 - 600 baud
		dstymr2baud1200		0111 - 1200 baud
		dstymr2baud1800		1000 - 1800 baud
		dstymr2baud2000		1001 - 2000 baud
		dstymr2baud2400		1010 - 2400 baud
		dstymr2baud3600		1011 - 3600 baud
		dstymr2baud4800		1100 - 4800 baud
		dstymr2baud7200		1101 - 7200 baud
		dstymr2baud9600		1110 - 9600 baud
		dstymr2baud19200		1111 - 19200 baud
		dstymr2recvclock	4	Receiver clock
		dstymr2recextclk		0 - External clock
		dstymr2recintclk		1 - Internal clock
		dstymr2transclock	5	Transmitter clock
		dstymr2trnextclk		0 - External clock
		dstymr2trnintclk		1 - Internal clock
			6-7	Reserved
dstycmdreg	1	Uart command register. Bit encoded.		
		Bit Name	Bit #	Description
		dstycrtransctrl	0	Transmitter control
		dstycrtcdis		0 - Disable transmitter

dstycrtcenb		1 - Enable transmitter
dstycrdtr	1	Data terminal ready
dstycrdtrhigh		0 - DTR high
dstycrdtrlow		1 - DTR low
dstycrrecvcrtl	2	Receiver control
dstycrrcdis		0 - Disable receiver
dstycrrcenb		1 - Enable receiver
dstycrforcebrk	3	Async force break
dstycrbrknorm		0 - normal
dstycrbrkforce		1 - force break
dstycrsenddle	3	Sync send DLE
dstycrdlenorm		0 - normal
dstycrdlesend		1 - send DLE
dstycrreseterror	4	Reset error
dstycrnoreset		0 - normal
dstycrreseterr		1 - reset error
dstycrrts	5	Request to send
dstycrrtshigh		0 - RTS high
dstycrrtslow		1 - RTS low
dstycropermodel	6	Operating mode
dstycropermode2	7	Operating mode continued
dstycromnormal		00 - Normal operation
dstycromautoecho		01 - Async autoecho
dstycromstripdle		01 - Sync strip DLE
dstycromlocallp		10 - Local loop back
dstycromremotelp		11 - Remote loop back

dstytermtype 1 Terminal type definition. This byte contains values for each type of terminal.

Value Name	Value	Description
	0-15	User defined types
	16-246	Reserved
dstywit	247	WIT terminal
dstyhydra	248	Hydra terminal
dstyvt100	250	VT-100 terminal
dstyvt52	251	VT-52 terminal
dstyt7000	252	T-7000 terminal
dstymg8000	253	MG-8000 terminal
dstytvi912c	254	TVI 912 C terminal
dstyvisual200	255	Visual 200 terminal

dstystatreg 1 Uart status register. Bit encoded.

Bit Name	Bit #	Description
dstysrtransrdy	0	Transmitter buffer ready

		dstysrtranfull		0 - Transmitter full
		dstysrtranempty		1 - Transmitter empty
		dstysrrecvrdy	1	Receiver buffer ready
		dstysrrecvempty		0 - Receiver empty
		dstysrrecvfull		1 - Receiver full
		dstysrdschg	2	DSR or DCD change
		dstysrdsrnormal		0 - Normal
		dstysrdsrchange		1 - Change in DSR or DCD
		dstysrparityerr	3	Parity error
		dstysrparnormal		0 - Normal
		dstysrparerror		1 - Async parity error. Sync parity error or DLE received
		dstysroverrunerr	4	Overrun error
		dstysrovernormal		0 - Normal
		dstysrovererror		1 - Overrun error
		dstysrframingerr	5	Framing error
		dstysrframnormal		0 - Normal
		dstysrframerror		1 - Async framing error. Sync SYN char.
		dstysrdcddetect	6	DCD Detect
		dstysrdcdhigh		0 - DCD high
		dstysrdcdlow		1 - DCD low
		dstysrdsrdetect	7	DSR Detect
		dstysrdsrhigh		0 - DSR high
		dstysrdsrlow		1 - DSR low
dstypacketterm	1	Holds code for packet termination characters		
		Value Name	Value	Description
		dstyptnoterm	0	Do not terminate packet on any control characters
		dstyptallterm	1	Terminate packets on all control characters
		dstyptcrterm	2	Terminate packet on carriage return <CR> character
dstyflagsl	2	Terminal status information. Bit encoded.		
		Bit Name	bit #	Description
		dstycontrolc	0	Control C enable (0 = enabled)
		dstyxonxoff	1	xon xoff enable (0 = enabled)
		dstycontrolx	2	Control X enable (0 = enabled)
		dstycontrolz	3	Control Z enable (0 = enabled)

dstycontrolo	4	Control O enable (0 = enabled)
dstytabmap	5	Tab map enable (1 = enabled)
dstymask8bit	6	Mask 8th bit enable (0 = enabled)
dstycontrolu	7	Control U enable (0 = enabled)
dstybroadcast	8	Broadcast enable (0 = enabled)
dstyhandshake1	9	Handshaking type
dstyhandshake2	10	
dstyhsbell		00 - No handshake, send bell
dstyhssoft		01 - Software handshake
dstyhshard		10 - Hardware handshake
dstyhsnone		11 - No handshake, no bell
dstyduplex	11	Full/half duplex (0 = full duplex)
dstymodemctrl	12	Modem control enable (1 = enabled)
dstyautobaud	13	Auto baud enable (1 = enabled)
dstyremote	14	Remote enable (1 = enabled)

dstyinputcnt	2	Count of characters in input interrupt buffer
dstyoutptcnt	2	Count of characters in output interrupt buffer
dstycolumnpos	2	Current column position
dstyinbufsz	2	Input buffer size in bytes
dstyoutbufsz	2	Output buffer size in bytes
dstywidth	2	The width of the given terminal screen
dstylength	2	The length of the given terminal screen
dstysubreadoper	4	Number of sub-read operations
dstysubwriteoper	4	Number of sub-write operations
dstyreserved	26	Reserved
dstyuserfield	8	User defined status

For PIPE class devices the second half of the device status table is defined as follows:

Name	Length (bytes)	Description
dsppreaderpid	4	Process ID of pending reader
dsppwriterpid	4	Process ID of pending writer
dspppipeid	4	The pipe's ID
dsppbuffersz	2	The buffer size in bytes
dsppbuffercnt	2	Number of characters in the pipe buffer
dsppreserved	40	Reserved
dsppuserfield	8	User defined status

For SYNC class devices the second half of the device status table is defined as follows:

Name	Length (bytes)	Description
dssymodereg1	1	Mode register 1 of the uart (See DSTYMODEREG1 for bit definitions)
dssymodereg2	1	Mode register 2 of the uart (See DSTYMODEREG2 for bit definitions)
dssycmdreg	1	Command register of the uart (See DSTYCMDREG for bit definitions)
dssytermtype	1	Terminal type definition. A binary value.

Value Name	Value	Description
dssyibm3741	249	IBM 3741 terminal
dssyibm2968	250	IBM 2968 terminal
dssyibm2770	251	IBM 2770 terminal
dssyibm3276	252	IBM 3276 terminal
dssyibm3275	253	IBM 3275 terminal
dssyibm2780	254	IBM 2780 RJE
dssyibm3780	255	IBM 3780 RJE

Name	Length (bytes)	Description
dssystatreg	1	Status register of uart. (See DSTYSTATREG for bit definitions)
dssynumbsync	1	Number of sync characters to write
dssyflags1	2	Device Status flags. Bit encoded.

Bit Name	Bit #	Description
dssymultipnt	0	0=point to point 1=multipoint
dssyebcdic	1	0=ascii line 1=ebcdic line
dssycrcccitt	2	0=crc-16 1=crc-ccitt

		dssylrc	3	0=crc (on above types) 1=lrc
		dssyasctoebcw	4	0=no translate on write 1=translate ascii to ebcdic on write
		dssyebctoascr	5	0=no translate on read 1=translate ebcdic to ascii on read
		dssytranstbl2	6	0=use translate table 1 1=use translate table 2
dssyinputcnt	2	Number of characters in input interrupt buffer		
dssyoutputcnt	2	Number of characters in output interrupt buffer		
dssyinbufsz	2	Input buffer size in bytes		
dssyoutbufsz	2	Output buffer size in bytes		
dssyprevrderr	4	Error from previous un-verified read		
dssyprevwrerr	4	Error from previous no-wait write		
dssyprevrdtype	1	Type of previous read		
		dssynontran - 0 Non-transparent read		
		dssytran - 1 Transparent read		
dssynumbtrpad	1	The number of trailing pads to write		
dssyrecsize	2	Used in transparent mode with ITBs		
dssyreserved	28	Reserved		
dssyuserfield	8	User defined status		

For NETWORK class devices the second half of the device status table is defined as follows:

Name	Length (bytes)	Description		
dsnkflags	2	Device status flags. Bit encoded.		
		Bit Name	Bit #	Description
		dsnkbyte	0	0=datagram mode 1=byte mode
		dsnkmodemctrl	1	0=not enabled 1=modem ctrl enabled
dsnkwindowsize	1	Window size the circuit will use		
dsnkreserved	53	Reserved		
dsnkuserfield	8	User defined status		

For NONDEV class devices the second half of the device status table is defined as follows:

Name	Length (bytes)	Description
dsnduserfield	64	User defined status

For QUEUE class devices the second half of the device status table is defined as follows:

Name	Length (bytes)	Description
dsquassocdev	9	A null terminated string containing the name of the physical printer device
dsqusenddev	9	A null terminated string containing the name of the physical device that control messages are to be sent to
dsquformname	10	A null terminated string containing the current form name
dsqunumexec	2	This is the maximum number of entries that can execute concurrently
dsqucurnumexec	2	This is the number of entries that are currently active
dsquflags	2	Device Status flags Bit encoded.
dsqulength	2	This holds the length of the forms of the printer associated with this queue
dsquwidth	2	This hold sthe width of the forms of the printer associated with this queue
dsqunextentry	4	This is the entry number of the next entry to be enqued

dsquflags bits:

Bit Name	Bit #	Description
dsquflupdating	0	Currently updating queue control file
dsquflqmstay	1	Queue manager process will remain running even when queue is empty
dsquflnorestart	2	When queue is mounted it does not restart jobs in the queue

dsqutype	1	This contains the type of queue this is. The values are:

Value Name	Value	Description
dsqutpprint	1	Print type queue
dsqutpjob	2	Job entry type queue

dsqubaseprior	1	This contains the priority that entries will be queued at if they specify the default priority
dsqureserved	20	Reserved
dsquuserfield	8	User defined status

To perform a set status operation the process must have write privilege to the device and either be the owner of the device (matching UICs) or have writephys privilege.

Related Privileges:

none	- Allows access to the device only if the process has write privilege to the device and has the same owner id and group id (uic) as the device.
altuic	- Allows the process to access the device if the owner of the image file for the current process has access to the device as described above.
bypass	- Allows the process to access the device without requiring write privilege. The process must still either be the owner of the device or have writephys privilege.
system	- Allows the process to access the device if the system has write privilege to the device as described above. (This does not obviate the need for device ownership or writephys privilege).
writephys	- Allows physical access to devices as described above. (This does not obviate the need for write privilege).

Parameters:

lun	- Logical unit number (LUN) of a file on the device whose status we wish to change.
dstat	- Address of the 128 byte device status table that is to be written. This buffer must be word aligned.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
errinvlfn	(132)	The logical unit number does not correspond to an open file. incorrect.
errnowritepriv	(145)	The process does not have Write Privilege for the file.

See Also:

_getdnam	- Get device name
_getdst	- Get device status
_giodst	- Get device status with LUN
_physop	- Perform physical device operation
_setdst	- Set device status

Assembler Calling Sequence:

```
%%sys$disk/sysincl.sys/dstatdisp.asm
push    lun                     ;value - logical unit number
push    dstat                   ;address - device status
push    status                  ;address - result of the operation
jsr     _siodst                 ;set device status with LUN
```

C Function Declaration:

```
#include "sys$disk/sysincl.sys/dstatdisp.h"
                                /* set device status with LUN */
long                            /* returns result of the operation */
_siodst(lun, dstat)
         long lun;              /* logical unit number */
         devicestatus dstat;    /* device status */
```

FORTRAN Subroutine Declaration:

```
c                               ! set device status with LUN
         subroutine _siodst(lun, dstat, status)
             integer*4 lun      ! logical unit number
             character*(*) dstat ! device status
             integer*4 status   ! result of the operation
```

Pascal Procedure Declaration:

```
%%sys$disk/sysincl.sys/dstatdisp.pas
procedure _siodst(                {** set device status with LUN}
        lun     : longint;        {** logical unit number}
        dstat   : ^array_of_char; {** device status}
    var status  : longint         {** result of the operation}
); external;
```

skip - Position tape

Description:

Position a tape to the beginning or end of volume, or forward or backward a relative number of file marks.

To successfully position the tape there must be no open files on the tape. Unless the process has bypass privilege, it must have read privilege to the device.

Related Privileges:

None	- Allows positioning if process has access to the device as described above.
bypass	- Allows positioning independent of the file protection.
altuic	- Allows positioning if the owner of image file for the current process has access to the device as described above.
system	- Allows positioning if the system has access to the device as described above.

Parameters:

dname - Address of a null terminated string which contains the name of the device to be positioned. This string will be translated automatically to its logical equivalent by the MCS. This string may contain up to 93 valid characters followed by a null.
If this string contains a file designation, the devicename portion of the file designation is used for this parameter.

stype - The type of skip to be performed.

Name	Value	Description
skipfile	0	skip file marks
skipbot	1	skip to beginning of volume
skipeot	2	skip to end of volume

These names are defined in sysincl.sys/sysequ.asm, sysincl.sys/sysequ.h and sysincl.sys/sysequ.pas

units - The number of files to skip. Positive values skip toward the end of the tape. Negative values skip toward the beginning of the tape.

For the "skip to end of volume" option this parameter is ignored.

For the "skip to beginning of volume", if the units parameter is zero, the tape is positioned to the physical beginning of tape. If the units parameter is non-zero, the tape is positioned one block past the beginning of tape, i.e. the tape label is skipped, and the tape is positioned at the beginning of the first file on the tape.

nskip	- Address of a long word to receive the number of files successfully skipped. If the stype parameter was "skip to beginning of volume" or "skip to end of volume", the value assigned to this parameter will be one.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinvdevnam	(130)	The specified devicename is syntactically incorrect.
errundevnam	(131)	The MCS does not recognize the devicename. Is the device mounted?
errnoreadpriv	(144)	The process does not have Read Privilege for the file.
errinvcloper	(173)	The operation is inappropriate for the device class.
errfilopen	(202)	The operation cannot be performed because a tape file is open.
errinvskpcmd	(206)	The specified skip or erase tape-function is undefined.
		Device integrity errors

See Also:

_getpos - Get the current file position
_physop - Perform physical device operation
_setpos - Set the current file position

Assembler Calling Sequence:

```
push    dname                   ;address - device name
push    stype                   ;value - type of skip
push    units                   ;value - number to skip
push    nskip                   ;address - number actually skipped
push    status                  ;address - result of the operation
jsr     _skip                   ;position tape
```

C function declaration:

```
                                        /* position tape */
    long                                /* returns result of the operation */
    _skip(dname.stype,units,nskip)
            char dname[94];             /* device name */
            long stype;                 /* type of skip */
            long units;                 /* number to skip */
            long *nskip;                /* number actually skipped */
```

Fortran Subroutine Declaration:

```
    c                                    ! position tape
            subroutine skip(dname. stype, units, nskip, status)
                character*94 dname       ! device name
                integer*4 stype          ! type of skip
                integer*4 units          ! number to skip
                integer*4 nskip          ! number actually skipped
                integer*4 status         ! result of the operation
```

Pascal Procedure Declaration:

```
    procedure  skip(                      {** position tape}
            dname   : string[93];         {** device name}
            stype   : longint;            {** type of skip}
            units   : longint;            {** number to skip}
        var nskip   : longint;            {** number actually skipped}
        var status  : longint             {** result of the operation}
    ); external;
```

Send interprocess mail.

Description:

Send a message to another process. The message may be up to 3952 bytes long. The process to which the message is addressed must exist at the time of the transmission. The message may consist of any information whatsoever.

Related Privileges:

None	- Allows sending mail to any process with the same owner id and group id (uic) as the calling process.
group	- Allows sending mail to any process with the same group id as the calling process.
world	- Allows sending mail to any process.

Parameters:

pid	- Process id of the process which is to receive the message.
buf	- Address of the message to be sent.
buflen	- Length of the message expressed in bytes.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
errprcsnotfnd	(2)	The specified process is not in the system process table.
errnomemavail	(7)	All available memory has been allocated.

See Also:

_gmail - Receive interprocess mail

Assembler Calling Sequence:

```
push    pid                     ;value - process id
push    buf                     ;address - message
push    buflen                  ;value - message length
push    status                  ;address - result of the operation
jsr     _smail                  ;send interprocess mail
```

C function declaration:

```
                                        /* send interprocess mail */
        long                            /* returns result of the operation */
        _smail(pid, buf, buflen)
                long pid;               /* process id */
                char *buf;              /* message */
                long buflen;            /* message length */
```

Fortran Subroutine Declaration:

```
        c                               ! send interprocess mail
                subroutine smail(pid, buf, buflen, status)
                    integer*4 pid       ! process id
                    character*(*) buf   ! message
                    integer*4 buflen    ! message length
                    integer*4 status    ! result of the operation
```

Pascal Procedure Declaration:

```
        procedure _smail(               {** send interprocess mail}
                pid     : longint;      {** process id}
                buf     : ^array_of_char; {** message}
                buflen  : longint;      {** message length}
            var status  : longint       {** result of the operation}
        ); external;
```

tranpid - Translate another processes logical name.

Description:

Given a logical name. return the equivalent. If no translation can be found, the equivalent is a copy of the original.

When a translation for a name is found, the equivalent string will be translated again until one of the following occurs:
- The equivalent does not translate into anything else.
- The equivalent is defined in terms of itself, (a recursive definition is detected.
- The equivalent has been translated 16 times.

This feature allows logical names to be defined in terms of other logical names.

Given a pid, searches the logical name table of the specified process. If the name is not found, continues searching in the logical name table of the parent of the specified process, and so on with the grandparents until either the name is found or there are no other parents. If it is still not found, it will search the system logical name table.

Abbreviations are allowed in logical names. An asterisk (*) in the logical name is a marker that indicates the minimum string that is a recognized abbreviation of the logical name. For example, if the logical name is "PR*INT", a translation of any of the strings "PR", "PRI", "PRIN", or "PRINT" will return the equivalence.

If there is more than one occurrence of a name. the first one found is used. (Note that there can be only one instance of a given name in a process's logical name table)

Related Privileges:

none	- Allows the translation of logical names with the logical name table of any process with the same owner id and group id (uic) as the calling process.
group	- Allows the translation of logical names with the logical name table of any process with the same group id as the calling process.
world	- Allows the translation of logical names with the logical

name table of any process.

Parameters:

pid	- A long word containing the process ID of the process whose logical name tables are to be used. 0 refers to the current process. -1 refers to the parent of the current process.
lname	- Address of a null terminated string containing the logical name to be translated. The maximum length of this string is 93 significant characters followed by a null. If this string is longer than 93 characters, the string is truncated, and no attempt is made to translate it.
equiv	- Address of a 94 byte buffer to receive the equivalent of the logical name. The result will be null terminated. The string may contain up to 93 valid characters followed by a null.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
errprcsnotfnd	(2)	The specified process is not in the system process table.

See Also:

assign	- Assign a logical name
_gassign	- Assign a global logical name
_getglb	- Retreive a global logical name
_getlog	- Retrieve a logical name
_trans	- Translate a logical name

Assembler Calling Sequence:

```
push    pid                         ;value - process id
push    lname                       ;address - logical name
push    equiv                       ;address - equivalent string
push    status                      ;address - result of the operation
jsr     _tranpid                    ;translate another processes logical name
```

C function declaration:

```
                                    /* translate another processes logical name
long                                /* returns result of the operation */
_tranpid(pid, lname. equiv)
        long pid;                   /* process id */
        char lname[94];             /* logical name */
        char equiv[94];          /* equivalent string */
```

Fortran Subroutine Declaration:

```
    c                                ! translate another processes logical name
            subroutine tranpi(pid, lname, equiv, status)
                integer*4 pid        ! process id
                character*94 lname   ! logical name
                character*94 equiv   ! equivalent string
                integer*4 status     ! result of the operation
```

Pascal Procedure Declaration:

```
    procedure  tranpid(               {** translate another processes logical name
            pid     : longint;        {** process id}
            lname   : string[93];     {** logical name}
        var equiv   : string[93];     {** equivalent string}
        var status  : longint;        {** result of the operation}
    ); external;
```

trans - Translate a logical name.

Description:

Given a logical name, return the equivalent. If no translation can be found, the equivalent is a copy of the original.

When a translation for a name is found, the equivalent string will be translated again until one of the following occurs:
- The equivalent does not translate into anything else.
- The equivalent is defined in terms of itself, (a recursive definition is detected.
- The equivalent has been translated 16 times.

This feature allows logical names to be defined in terms of other logical names.

Searches the logical name table of the current process. If the name is not found, continues searching in the logical name table of the parent of the current process, and so on with the grandparents until either the name is found or there are no other parents. If it is still not found, it will search the global logical name table.

Abbreviations are allowed in logical names. An asterisk (*) in the logical name is a marker that indicates the minimum string that is a recognized abbreviation of the logical name. For example, if the logical name is "PR*INT", a translation of any of the strings "PR", "PRI", "PRIN", or "PRINT" will return the equivalence.

If there is more than one occurrence of a name, the first one found is used. (Note that there can be only one instance of a given name in a process's logical name table)

Related Privileges:

None.

Parameters:

lname	- Address of a null terminated string containing the logical name to be translated. The maximum length of this string is 93 significant characters followed

```
                by a null.  If this string is longer than 93 characters,
                the string is truncated, and no attempt is made to
                translate it.
    equiv     - Address of a 94 byte buffer to receive the equivalent
                of the logical name.  The result will be null
                terminated.  The string may contain up to 93 valid
                characters followed by a null.
```

Diagnostics:

None.

See Also:

```
    _assign  - Assign a logical name
    _gassign - Assign a global logical name
    _getglb  - Retreive a global logical name
    _getlog  - Retrieve a logical name
    _tranpid - Translate another processes logical name
```

Assembler Calling Sequence:

```
    push    lname                   ;address - logical name
    push    equiv                   ;address - equivalent string
    jsr     _trans                  ;translate a logical name
```

C function declaration:

```
                                    /* translate a logical name */
    void                            /* no result */
    _trans(lname, equiv)
            char lname[94];         /* logical name */
            char equiv[94];         /* equivalent string */
```

Fortran Subroutine Declaration:

```
    c                               ! translate a logical name
            subroutine trans(lname, equiv)
                character*94 lname  ! logical name
                character*94 equiv  ! equivalent string
```

Pascal Procedure Declaration:

```
    procedure  trans(               {** translate a logical name}
            lname   : string[93];   {** logical name}
        var equiv   : string[93]    {** equivalent string}
    ); external;
```

udefmem - Undefine a named shared memory area.

Description:

Normally, shared memory areas will go away when the last process using the memory area has terminated. If the linger bit is set at the time that the named shared memory area is defined, then the memory area must be explicitly removed using _udefmem. This will clear the linger bit and if no one is currently using the area will deallocate the memory. If someone is using the memory, a warning will be returned and the memory will go away when the last user finishes. The process executing this call must have delete privilege to the shared memory area or else have Bypass privilege.

Related Privileges:

none	- Allows deletion of a named shared memory area if the process has delete privilege to the named shared memory.
altuic	- Allows deletion of a named shared memory area if the owner of the process's image file has delete privilege to the named shared memory.
bypass	- Allows the process to delete any named shared memory area.
operator	- Allows the process to delete any named shared memory area.

Parameters:

mname	- Address of a null terminated string identifying the specific memory area. This string will be translated automatically by WMCS into its logical equivalent. This string may contain up to 93 significant characters followed by a null.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

ermoname	(82)	The name specified does not exist.
ermodelpriv	(146)	The process does not have delete privilege for the file.

See Also:

_defmem	- Define a named sharable memory area.
_shrmem	- Share a named sharable memory area.
_ushrmem	- Unshare a named sharable memory area.

```
 getmlst  -  Get a list of named sharable memory areas.
_setmuic  -  Change owner of a named sharable memory area.
_setmprt  -  Change protection of a named sharable memory area.
```

Assembler Calling Sequence:

```
        push    mname           ; address - name of memory area
        push    status          ; address - status
        jsr     _udefmem        ; Undefine a named shared memory area.
```

C Function Declaration:

```
                                /* undefine a named shared memory area */
long                            /* returns result of the operation */
_udefmem(mname)
    char mname[94];             /* name of memory area */
```

FORTRAN Subroutine Declaration:

```
c                               ! undefine a named shared memory area
        _udefmem(mname, status)
            character*94 mname  ! name of memory area
            integer*4 status    ! result of the operation
```

PASCAL Procedure Declaration:

```
procedure  udefmem(              {** undefine a named shared memory area}
        mname   : string[93];    {** name of memory area}
    var status  : longint        {** result of the operation}
); external;
```

Unlock a records in an open file.

Description:

_unlock is the complement of _lock. Given a valid logical unit number (lun) it unlocks the specified records so that they can be accessed by processes other than the calling process. Any subportion of a lock request can be unlocked.

When unlocking records the following rules will be applied:

a) If the entire unlock request is locked by this process then it will be successful and a normal status will be returned.
b) If some of the unlock request is locked by this process but not all, then those pieces will be unlocked but a warning will be returned saying it tried to unlock unlocked segments.
c) If the entire unlock request is not locked by this process then an error will be returned saying it tried to unlock unlocked segments.

Records may only be unlocked on disk class devices.

Related Privileges:

None.

Parameters:

lun	- The logical unit number of the file in which records are to be unlocked.
recnum	- A long word containing the starting position of the section of the file to be unlocked. The first record in the file is record 0. This is an unsigned value. A recnum of $FFFFFFFF (-1) is a reserved value and corresponds to the current file position.
nrecs	- A long word containing the number of records to be unlocked. If this parameter is zero, it means to unlock from the current position to the logical end of file. This also is an unsigned value.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinvlfn	(132)	The logical unit number does not correspond to an open file.
errinvcloper	(173)	The device class is inappropriate for the

		operation.
errksamnorec	(237)	The specified record(s) is not locked.
errrlocknotl	(253)	The process attempted to unlock a record(s) it had not locked.
errlockint	(254)	(MCS error) A discrepency in the Record Locking code has been detected.

See Also:

_lock	- Lock records within an open file
_read	- Read from an open file
_write	- Write to an open file

Assembler Calling Sequence:

```
push    lun                         ;value - logical unit number
push    recnum                      ;value - starting record number
push    nrecs                       ;value - number of records
push    status                      ;address - result of the operation
jsr     _unlock                     ;unlock records in an open file
```

C function declaration:

```
                                    /* unlock records in an open file *'
long                                /* returns result of the operation
_unlock(lun,recnum,nrecs)
        long lun;                   /* logical unit number */
        long recnum;                /* starting record number */
        long nrecs;                 /* number of records */
```

Fortran Subroutine Declaration:

```
c                                   ! unlock records in an open file
        subroutine unlock(lun, recnum, nrecs, status)
            integer*4 lun           ! logical unit number
            integer*4 recnum        ! starting record number
            ingeger*4 nrecs         ! number of records
            integer*4 status        ! result of the operation
```

Pascal Procedure Declaration:

```
procedure _unlock(                  {** unlock records in an open file}
        lun     : longint;          {** logical unit number}
        recnum  : longint;          {** starting record number}
        nrecs   : longint;          {** number of records}
    var status  : longint           {** result of the operation}
); external;
```

ushrmem - Unshare a named shared memory area.

Description:

Associated with each process is a list of the shared memory areas which are currently allocated to the process. When an area is no longer needed, _ushrmem is called to decrement the reference count of the specified memory area. If the reference count becomes zero and the linger bit is not set, then the shared memory area will be removed from the system. The process cleanup routines will reduce the reference counts for any shared memory areas still belonging to the process at the time it terminates.

Related Privileges:

none

Parameters:

mname	- Address of a null terminated string identifying the memory area to be deallocated. This string will be translated automatically by WMCS into its logical equivalent. This string may contain up to 93 significant characters followed by a null.
adr	- A long word containing the location in local user logical memory where the shared memory area starts. This needs to be specified since the process may have the same shared memory area mapped to several locations in its memory space.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinvadr	(4)	The memory address is not on a 4K page boundary.
errnoname	(82)	The name specified does not exist.

See Also:

_defmem	- Define a named sharable memory area.
_udefmem	- Undefine a named sharable memory area.
_shrmem	- Share a named sharable memory area.
_getmlst	- Get a list of named sharable memory areas.
_setmuic	- Change owner of a named sharable memory area.
_setmprt	- Change protection of a named sharable memory area.

Assembler Calling Sequence:

```
    push    mname                    ; address - name of memory area
    push    adr                      ; value   - address of memory area
    push    status                   ; address - result of the operation
    jsr     _ushrmem                 ; Unshare a named shared memory area.
```

C Function Declaration:

```
                                     /* unshare a named shared memory area */
    long                             /* returns result of the operation */
    _ushrmem(mname. adr)
        char mname[94];              /* name of memory area */
        long adr;                    /* address of memory area */
```

FORTRAN Subroutine Declaration:

```
    c                                ! unshare a named shared memory area
            _ushrmem(mname. adr, status)
                character*94 mname   ! name of memory area
                integer*4 adr        ! address of memory area
                integer*4 status     ! result of the operation
```

PASCAL Procedure Declaration:

```
    procedure  ushrmem(              {** unshare a named shared memory area}
            mname   : string[93];    {** name of memory area }
            adr     : longint;       {** address of memory area }
        var status  : longint        {** result of the operation}
    ); external;
```

Get the OS version banner.

Description:

Returns a null terminated ASCII string which contains the OS version number, release date and copyright notice.

Related Privileges:

None.

Parameters:

siteid	- A long word containing the system id of the system whose version banner is being requested. A siteid of zero corresponds to the system on which the calling process is executing.
buf	- The address of the user buffer to receive the version banner. This buffer must be at least 81 bytes long. The resulting string will be null terminated.
status	- The address of a long word to receive the result of the operation.

Diagnostics:

errinvsiteid	(8)	The specified site id does not exist.

See Also:

None.

Assembler Calling Sequence:

```
push    siteid                      ;value - system id
push    buf                         ;address - user buffer
push    status                      ;address - result of the operation
jsr     _version                    ;Get the OS version banner
```

C function declaration:

```
                                    /* get the OS version banner */
long                                /* returns result of the operation */
_version(siteid, buf)
        long siteid;                /* system id */
        char buf[81];               /* user buffer */
```

Fortran Subroutine Declaration:

```
c                                       ! get the OS version banner
          subroutine versio(siteid, buf, status)
               integer*4 siteid         ! system id
               character*81 buf         ! user buffer
               integer*4 status         ! result of the operation
```

Pascal Procedure Declaration:

```
procedure _version(                 {** get the OS version banner}
      siteid   : longint;           {** system id}
   var buf     : string[80];        {** User buffer}
   var status  : longint            {** result of the operation}
); external;
```

Pause for a period of time.

Description:

Relinquishes control to the operating system until the system clock time is greater than or equal to the time parameter passed to the routine. If the time parameter passed to the routine is negative, the absolute value of the parameter is interpreted as being the number of clock ticks to wait until waking the process. If the time parameter is positive, it is taken to be a clock time in system format, and control is returned to the calling routine when the system clock becomes greater or equal to that time. If the time parameter passed is zero, control is relinquished for one scheduling cycle.

System time format is expressed in 8 bytes. The format of these 8 bytes is as follows:

Bytes	Description
0,1	The year (counted from A.D. 0) Example: 1985
2,3	The day of the year (1..365 or 1..366)
4	The hour of the day (0..23)
5	The minute of the hour (0..59)
6	The second of the minute (0..59)
7	The tick (in 100ths of a second) (0..99)

Related Privileges:

None.

Parameters:

mstime	- The most significant 32 bits of the time parameter. (-1 is for relative waits, or day and year.) If the value of this parameter and the next parameter is zero, the process will relinquish control for one scheduling cycle.
lstime	- The least significant 32 bits of the time parameter (number of ticks, or hour, minute, second, and tick). If mstime is -1, then the value of this parameter represents -1 times the number of "ticks" to wait before rescheduling the process, i.e. the number that you enter should be negative.

NOTE: If you are specifying relative time, the two 32-bit numbers are treated as one 64-bit number.

Diagnostics:

None.

See Also:

_hibern - Hibernate a process

Assembler Calling Sequence:

```
push    mstime                   ;value - day and year
push    lstime                   ;value - hour, minute, second, tick
jsr     _wait                    ;pause for a period of time
```

C Function Declaration:

```
                                 /* pause for a period of time */
void                             /* no result */
_wait (mstime, lstime)
         long mstime;            /* day and year */
         long lstime;            /* hour, minute, second, tick */
```

FORTRAN Subroutine Declaration:

```
c                                ! pause for a period of time
         subroutine _wait(mstime, lstime)
             integer*4 mstime    ! day and year
             integer*4 lstime    ! hour, minute, second, tick
```

Pascal Procedure Declaration:

```
procedure _wait(                {** pause for a period of time}
        mstime  : longint;        {** most significant time}
        lstime  : longint;        {** least significant time}
); external;
```

wake - Wake a hibernated process.

Description:

Zeroes the hibernate count and clears the hibernate status bit in the process control block of the specified process. In other words the process will be awakened no matter how many times it has been hibernated. No error occurs if the process being awakened is not hibernating. Note that a process cannot wake itself since a hibernating process cannot make the call.

Related Privileges:

none	- Allows waking any process with the same owner id and group id as the calling process.
group	- Allows waking any process with the same group id as the calling process.
world	- Allows waking any process.

Parameters:

pid	- Process id of the process to wake up. A process id of -1 refers to the parent of the calling process.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
errprcsnotfnd	(2)	The specified process is not in the system process table.

See Also:

_hibern - Hibernate a process
_wakec - Wake a hibernated process with count

Assembler Calling Sequence:

```
push   pid                       ;value - process id
push   status                    ;address - result of the operation
jsr    _wake                     ;wake a hibernated process
```

C function declaration:

```
                                      /* wake a hibernated process */
    long                              /* returns result of the operation */
    _wake (pid)
            long pid;                 /* process id */
```

Fortran Subroutine Declaration:

```
    c                                 ! wake a hibernated process
            subroutine wake(pid, status)
                integer*4 pid         ! process id
                integer*4 status      ! result of the operation
```

Pascal Procedure Declaration:

```
    procedure  wake(                  {** wake a hibernated process}
            pid     : longint;        {** process id}
        var status  : longint         {** result of the operation}
    ); external;
```

wakec - Wake a hibernated process with count.

Description:

Decrements the hibernate count in the process control block of the specified process. When the count goes to zero the hibernate status bit of the specified process is then cleared. In other words the process does not resume execution until _wakec is called at least as many times as _hibern has been called. No error occurs if the process being awakened is not hibernating. Note that a process cannot wake itself since a hibernating process cannot make the call.

Related Privileges:

none	- Allows waking any process with the same owner id and group id as the calling process.
group	- Allows waking any process with the same group id as the calling process.
world	- Allows waking any process.

Parameters:

pid	- Process id of the process to wake up. A process id of -1 refers to the parent of the calling process.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
errprcsnotfnd	(2)	The specified process is not in the system process table.

See Also:

_hibern	- Hibernate a process
_wake	- Wake a hibernated process

Assembler Calling Sequence:

```
push    pid                     ;value - process id
push    status                  ;address - result of the operation
jsr     _wakec                  ;wake a hibernated process with count
```

C function declaration:

```
                                        /* wake a hibernated process with count */
    long                                /* returns result of the operation */
    _wakec (pid)
            long pid;                   /* process id */
```

Fortran Subroutine Declaration:

```
    c                                   ! wake a hibernated process with count
            subroutine wakec(pid, status)
                integer*4 pid           ! process id
                integer*4 status        ! result of the operation
```

Pascal Procedure Declaration:

```
    procedure  wakec(                    {** wake a hibernated process with count}
            pid     : longint;          {** process id}
        var status  : longint           {** result of the operation}
    ); external;
```

_write

Write to an open file.

Description:

Given a valid logical unit number (lun) of a file open for write access, transfers one or more records to the file from the process's buffer.

On successful completion, returns the number of complete records actually transferred. This number may be less than the number requested if:

1) the device becomes 'full'
2) a timeout occurs
3) a device error occurs

Related Privileges:

None.

Parameters:

lun	- Logical unit number (lun) of the file to be written.
recnum	- The record position in the file at which to write the data. The first record in a file is record Ø. A $FFFFFFFF (-1) in this parameter means to write the record at the current file position. Note that recnum is an unsigned long word. On devices other than disk, the only option is to read from the current file position. For instance on tape, if the value of the recnum parameter is not -1 (or the current record number), an error is returned.
edmode	- The edit mode to use. This parameter is divided into two 16 bit fields. The least significant word represents which edit mode processor to use. The most significant word contains edit mode flags for the processor.

An output edit mode processor is used to filter the output stream as it is written to the file. The following transformations are defined:

Name	Edmode	Description
emvwriteraw	Ø	Raw data. No alterations of data.
	1	Reserved. Must be set to zero (Ø).
	2	Reserved. Must be set to zero (Ø).
emvwriteln	3	For tty class devices, this edit mode will transform all line feeds (1Ø) found in the data to carriage return (13) line feed (1Ø) combinations. On all other device classes, this edit mode is the same as edmode Ø.

The most significant word of the edmode parameter contains the following bit flags. When the bit is a one (1) the following descriptions apply:

Bit name	Bit #	Description
	16	Reserved. Must be set to zero (Ø).
emspcompact	17	Space compaction - On sync class devices spaces are automatically compacted.
emforcedwrite	18	Forced write - The data will be written all the way to the device before control returns to the process. All device errors are returned. Note that without forced write, the data will be written only to the device cache and device errors will not be detected. Forced write results in lower performance but better error control.
emtransparent	19	Transparent mode - On bisync class devices causes the data to be written in transparent mode.

emnowaitwrite	20	No wait on write - Initiates the write and does not suspend the calling process waiting for an acknowledge. If an error occurs, it will be reported on a subsequent write.
emlockunlock	21	Write and unlock - On disk class devices, this bit will cause all of the records written to be unlocked.
emitbwrite	22	ITB write - On SYNC class devices, if set ITB record separators are to be used on this write.
emlinepause	22	Pause - On TTY class devices, if set the write is performed with a pause control. The length of each screen of output is determined by the length parameter for the device. This pause control works like the :pause switch on many CIP commands, i.e., an asterisk prompt is displayed after a screenful of data is written. Striking [SPACE BAR] writes another screen, striking [RETRN] writes one more line.
	23-31	Reserved. Must be set to zero (Ø).

timout	- The wait count is in 1ØØths of a second and represents the amount of time to wait for the transfer to complete before timing out.
buf	- The address of the buffer containing the data to be written. May be on a word or a byte boundary.
nrecs	- The number of records to write. This parameter is an unsigned long word. If it is zero, no data is transferred.
trnsfr	- Address of a long word to receive the number of records actually written.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errnomemavail	(7)	All available memory has been allocated.
errtimeout	(128)	A request was not completed within the specified time.
errinvlfn	(132)	The logical unit number does not correspond to an open file.
errreqtolrg	(137)	The request is too large for the system to handle.
errnowriteacc	(142)	The process does not have write-access to the specified file.
errinvsecreq	(151)	The WMCS cannot allocate more than 65535) sectors at a time.
errnospace	(154)	All available disk space has been allocated.
errinveditmd	(189)	The WMCS does not recognize the specified edit mode.
errbadpos	(197)	The process tried to access a record (on a tape) out of sequence.
errdeadlock	(234)	The specified record cannot be locked without causing a deadlock.
errreclocked	(235)	The specified record(s) are locked by another process.
errrlocknotl	(253)	The process attempted to unlock a record(s) it had not locked.
errlockint	(254)	(WMCS error) A discrepancy in the Record Locking code has been detected.
		Device integrity errors

See Also:

_create	Create a file
_getpos	Get the current file position
_lock	Lock records within an open file
_open	Open a file
_read	Read from an open file
_setpos	Set the current file position
_unlock	Unlock records in an open file

Assembler Calling Sequence:

```
    push    lun                  ;value - logical unit number
    push    recnum               ;value - record number
    push    edmode               ;value - edit mode
    push    timout               ;value - time out
    push    buf                  ;address - data to write
    push    nrecs                ;value - number of records to write
    push    trnsfr               ;address - number actually written
    push    status               ;address - result of the operation
    jsr     _write               ;write to an open file
```

C function declaration:

```
                                 /* write to an open file */
    long                         /* returns result of the operation */
    _write ( lun, recnum, edmode, timout, buf, nrecs, trnsfr)
            long lun;            /* logical unit number */
            long recnum;         /* record number */
            long edmode;         /* edit mode */
            long timout;         /* time out */
            char *buf;           /* data to write */
            long nrecs;          /* number of records to write */
            long *trnsfr;        /* number actually written */
```

Fortran Subroutine Declaration:

```
    c                            ! write to an open file
            subroutine write(lun, recnum, edmode, timout,
          &         buf, nrecs, trnsfr, status)
                integer*4 lun           ! logical unit number
                integer*4 recnum        ! record number
                integer*4 edmode        ! edit mode
                integer*4 timout        ! time out
                character*(*) buf       ! data to write
                integer*4 nrecs         ! number of records to write
                integer*4 trnsfr        ! number actually written
                integer*4 status        ! result of the operation
```

Pascal Procedure Declaration:

```
procedure _write(           {** write to an open file}
      lun     : longint;          {** logical unit number}
      recnum  : longint;          {** record number}
      edmode  : longint;          {** edit mode}
      timout  : longint;          {** time out}
      buf     : ^array_of_char; {** data to write}
      nrec    : longint;    {** number of records to write}
   var trnsfr : longint;          {** number actually written}
   var status : longint           {** result of the operation}
); external;
```

Write physical memory.

Description:

By default a process can access any memory that is part of its own logical address space ($000000 through $1FFFFF) To write memory above two megabytes, the process must either change to supervisor mode of operation or use this call asking MCS to write the memory for it.

Using _wtpmem to write physical memory has the additional property that when memory errors (e.g. attempt to access non-existant memory) occur, they are reported to the process through the status variable and are not considered fatal errors.

A process must have writephys privilege to write addresses in physical memory.

Related Privileges:

None	- Process not allowed to write physical memory
writephys	- Allows process to write physical memory

Parameters:

siteid	- A long word containing the system id of the system whose physical memory is to be written. A siteid of zero corresponds to the system on which the calling process is executing.
adr	- Address of the physical memory to be written.
mode	- Specifies whether to use byte, word or long word transfers from the specified address. 0 indicates byte, 1 indicates word and 2 indicates long word transfers. All other values are reserved and should not be used.
buf	- Address of the user buffer containing the data to be written.
nrec	- The number of units (bytes, words or long words) to be transferred.
trnsfr	- Address of a long word to receive the number of units actually transferred.
status	- Address of a long word to receive the result of the operation.

Diagnostics:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.

erroddbufaddr	(3)	The process's buffer does not begin on a word boundary.
errbustrap	(37)	The process has a bus error.
errnonexmem	(39)	The process attempted to access nonexistent memory.
errmemparity	(40)	The process has a memory parity-error.

See Also:

_chsuper - Change to supervisor mode
_rdpmem - Read physical memory

Assembler Calling Sequence:

```
push    siteid                      ;value - system id
push    adr                         ;value - address to write
push    mode                        ;value - byte, word, long word moves
push    buf                         ;address - data to write
push    nrec                        ;value - number of units to write
push    trnsfr                      ;address - num of units transferred
push    status                      ;address - result of the operation
jsr     _wtpmem                     ;write physical memory
```

C function declaration:

```
                                    /* write physical memory */
long                                /* returns result of the operation */
_wtpmem(siteid, adr, mode, buf, nrec, trnsfr)
        long siteid;                /* system id */
        long adr;                   /* address to write */
        long mode;                  /* byte, word, long word moves */
        char *buf;                  /* data to write */
        long nrec;                  /* number of units to write */
        long *trnsfr;               /* num of units transferred */
```

Fortran Subroutine Declaration:

```
c                                   ! write physical memory
        subroutine wtpmem(siteid, adr, mode, buf, nrec,
     &          trnsfr, status)
            integer*4 siteid        ! system id
            integer*4 adr           ! address to write
            integer*4 mode          ! byte, word or long word moves
            character*(*) buf       ! data to write
            integer*4 nrec          ! number of units to write
            integer*4 trnsfr        ! num of units transferred
            integer*4 status        ! result of the operation
```

Pascal Procedure Declaration:

```
procedure _wtpmem(                  {** write physical memory}
        siteid  : longint;          {** system id}
        adr     : longint;          {** address to write}
        mode    : longint;          {** byte, word, or long word moves}
        buf     : ^array_of_char;   {** data to write}
        nrec    : longint;          {** number of units to write}
    var trnsfr  : longint;          {** number of units transferred}
    var status  : longint           {** result of the operation}
); external;
```

Chapter 4

Keyed Sequential Access Method (KSAM)

Keyed Sequential Access Method (KSAM) is a file system that allows fixed length data records to be accessed rapidly based upon one or more indices or keys found within each data record.

Depending upon the exact nature and quantity of defined keys, a typical KSAM application program can randomly find and access a particular data record among many millions and do so with no more than three to five disk accesses. Processing data records sequentially (with respect to a particular key) typically takes no more than one disk access to load the appropriate portion of the key file.

Features of KSAM

KSAM has a number of special features:

1. KSAM is a multi-key, multi-segmented, multi-access file access method.

   Multi-key means that you can typically define as many as three hundred keys for each record of a file.

   Multi-segmented means that each key can comprise as many as fifteen segments. The segments need not be adjacent.

   Multi-access means that many different processes can read simultaneously information from a single file. A single process with sufficient privilege, however, can lock out other users and gain exclusive access to the file.

2. You can read a file randomly, based upon the value of a key or a portion of a key. You can also read the file sequentially forward or backward on any key or portion of a key.

3. KSAM is based upon the B-tree data structure that offers the following advantages over similar file access methods:

   KSAM runs on all Motorola MC68000-based WICAT computers. Previously, indexed sequential file access methods were available only on intermediate-sized mini-systems and larger computers.

   The B-tree data structure requires little operator maintenance. Algorithms on other indexed sequential file access systems often become inefficient as files are enlarged or deleted, occasionally requiring the operator to rebuild the key structure.

   KSAM is fast. For example, given a file containing 5000 150-byte records, each with 4-byte keys, records can be read randomly at 1000 records per minute, and can be written randomly at 500 records per minute.

   KSAM key files are rebuildable. Since data and key information reside in separate files, you can easily reconstruct the key file from the data file, if the key file is ever damaged.

Calling KSAM

The various KSAM operations are implemented by system supervisor calls (SVCs). For a general explanation of SVCs, see Chapter 3 of this manual. The KSAM SVCs can be called from assembly language and from most of the compiled high-level languages that WICAT supports. The calling sequences and parameters for each of these routines are described in Chapter 3.

KSAM as a Class Handler

KSAM consists of approximately 16 Kbytes of shareable, re-entrant code. You may elect to load this code as a class handler when you boot your system.

When WMCS encounters a call to a KSAM SVC and determines that KSAM is present, WMCS transfers control to the KSAM class handler. If KSAM is absent, WMCS returns an error to the calling process.

Memory Requirements

In addition to the 16 Kbytes for the class handler, KSAM requires four Kbytes of memory whenever one or more files are open for KSAM processing. Each additional file that is opened requires additional memory to be allocated for key file processing. The total amount of memory used depends upon the file definition and the options requested by the calling process. See the SVC descriptions for _KOPEN and _KCREAT for details.

KSAM File Structure

KSAM files are physically composed of a data file containing data records and a keys file containing the key information. This composition is for two reasons:

1. The keys may be treated uniformly (internally consistent) when data records are segregated.

2. The keys file may be recreated from the data file if a system failure ever damages the keys file.

With key information separated from data records, you must ensure that the data records agree with the key information. Neither WMCS nor KSAM knows that a key file has been erroneously paired with a data file or that either the key file or the data file has been independently changed in relation to its partner.

NOTE: Should independent changes occur in either file of a paired key file and data file, subsequent KSAM operations on the file pair may produce unpredictable results and irreparably damage the integrity of the data and/or key files.

Data File

The data file contains the actual data records, along with a flag indicating whether a record is deleted.

The creating process specifies the length of the data records. The data file comprises fixed length data records, and the data records must be between 4 and 65334 bytes inclusive.

Besides the user data, each record contains as its first byte a deletion flag. With the deletion flag KSAM can keep track of obsolete data record slots. These slots are reused as new data are written to the file.

This first byte also allows a recovery program to determine whether a record in a data file contains valid data.

A recovery program could use the deletion flag byte to read records from a data file, determine their validity, and then reconstruct a valid key file by writing only valid data records to a new KSAM file pair.

Keys File

The keys file consists of a number of 1-Kbyte key blocks containing information about the keys defined for the KSAM file, the data records and the B-trees of the keys. In the keys file, keys are stored and organized, and random searches and sequential reads are performed. The keys file contains pointers to the data records in the data file.

The key definition area in the keys file contains two kinds of information:

1. Information regarding the type, length, and components of each key.

2. Information regarding obsolete data and key records, the number of active and unused data and key records, the depth of the KSAM key tree structure, and the location of the root node of each key tree.

Key information used to locate particular records begins after the key definition area.

The structure used is a modified B-tree. Each defined key is represented by its own B-tree. The B-trees share the same file.

The leaf level of the B-trees points into the data file.

Because each node of the B-tree is 1 kbytes long, the B-tree can point to many children nodes. A four-byte key definition would allow a fan-out of 125 children at each level. (4-byte key + 4 bytes for pointer = 8 bytes; 1K/8 = ~125) Thus a four-level B-tree could provide indices into 125 * 125 * 125 * 125 or over 244 million data records. The top level of each B-tree is always kept within the KSAM file system.

You may specify an additional number of key nodes for permanent residency. Should only the root node of a four-level B-tree be kept in memory, any record of a 244-million record data file could be found and accessed with a maximum of four disk accesses.

The levels occupied by any B-tree are available via the _KINFO SVC. The number of levels provides a useful clue for estimating the access time necessary to refer to a data record using that key.

Pointers

Each file pair opened using KSAM is assigned three pointers. The user process cannot access these pointers. Nonetheless, the pointers tell how the key file is manipulated and how the data file is referenced.

CURRENT KEY POINTER. The current key pointer defines which of the B-trees will be used for key comparisons and data file references.

The current key is defined by default by calls to _KOPEN and _KCREAT and may be changed with calls to _KMOVFB and _KFIND.

CURRENT POSITION POINTER. The current position pointer may be thought of as always pointing between two data records.

Records may be read either sequentially forward or backward relative to the current position pointer.

The current position pointer is set by calls to _KCREAT and _KOPEN, and may be changed by _KMOVFB, _KFIND, _KWRITE, _KUPDAT, and _KDELET.

CURRENT RECORD POINTER. The current record pointer points to the last record read or written. Thus, it is defined by a successful call to _KREAD or _KWRITE.

Successful calls to _KOPEN, _KCREAT, _KDELET, _KMOVFB, and _KFIND leave the current record pointer undefined as do error returns from a KSAM function SVC.

Two SVCs, _KDELET and _KUPDAT, require that the current record be defined before they may be called. _KREAD is the typical choice for defining current record to make these calls.

Current Key

A current key and a current record exist whenever a KSAM file is being read or written. If a key is used in randomly finding a record, the key number must be given to KSAM.

This key number establishes which key becomes the current key.

If a sequential read is then executed, the next record (or previous record, in the case of a backward read) is that whose key alphabetically (numerically for numeric keys) follows (or precedes) the key of the current record.

All operations performed on the file will take place relative towards the current key.

The current key can be changed only by executing a random find of a record (_KFIND) or by placing the file at the beginning or end of the file (_KMOVFB).

The current key can become undefined if any of the above mentioned calls fails.

Current Record

A successful execution of a read (_KREAD), write (_KWRITE), or update (_KUPDAT) establishes the current record. The current record becomes undefined whenever one of the following five calls is performed, or if any of the other calls fail:

1. A random find (_KFIND)
2. A position to beginning or end of file (_KMOVFB)
3. A create (_KCREAT)
4. An open (_KOPEN)
5. A record delete (_KDELET)

Keys

Six distinct kinds of keys are supported.

1. signed byte	(8 bits)
2. unsigned byte	(character)
3. signed word	(16 bits)
4. unsigned word	(16 bits)
5. signed longword	(32 bits)
6. unsigned longword	(32 bits)

Size

Each key may be up to 255 bytes long. Word and longword keys and key segments must lie on word boundaries (even byte) within memory and within the data record. Word keys and key segments must be two-byte multiples, and longword keys and key segments must be four-byte multiples. Assigning either a byte value in a record definition may misalign word or longword key fields that follow. You may have to offset the other keys to align them on word or longword boundaries.

Key Definitions

No arbitrary limit exists for the number of keys that may be defined for a KSAM file pair.

The one limiting factor on key definitions is that the sum of key definitions may occupy no more than 3500 bytes.

An average key definition occupying eight bytes would allow for the definition of over 400 keys.

Although all keys must be constructed from data fields within the data record, these data fields need not be contiguous.

Up to 15 noncontiguous segments may be included in the definition of any key.

The same data field may be used in as many keys as desired.

KSAM allows a key definition to designate that duplicate key values be disallowed for a particular key. When duplicate key values are detected, KSAM returns an error condition.

Updating a Record

Any key may be updated. _KUPDAT allows a data record to be changed, and any changed key values are automatically updated. Updating eliminates the necessity of deleting an entire record and writing a new one when only one or a few of its many keys have changed values.

Searching for a Key

Searching is provided on partial as well as complete key values. Should less than the complete length of the key be specified for a search using _KFIND, the current position pointer moves to the record containing the first occurrence of the key that matches the partial length specified.

Should no match be possible, the current position pointer is left pointing before the record that would immediately follow the specified search key in alphabetical or numerical sequence.

The error return status of _KFIND indicates whether a match to a searched-for key was found or whether the current position pointer could be placed only immediately before where the key should be found were it present.

Locking Records

LOCKING: Records may be write locked. Limited only by available memory, a process may update, write, and delete any number of data records within a KSAM file. You need 32 bytes of memory to lock one record. The process attempting to update or delete a locked record may specify how long it will wait for the record to become unlocked before returning with an error.

UNLOCKING: All locked records are automatically unlocked when the file is closed by the process that locked the records. A process may unlock specific records or all records that it has previously locked. Because the locking information is retained in memory, a system crash does not leave a file on disk with records that cannot be unlocked.

KSAM files are automatically closed upon process deletion. Locked records are automatically unlocked on file closing.

Multiple Processes

KSAM allows multiple processes to read simultaneously a KSAM file pair that has not been read locked to other processes upon open.

Multiple processes can modify a file, but KSAM coordinates the procedure by allowing only one process at a time to modify (using _KWRITE, _KDELET, or _KUPDAT) the files. Thus, such modifications normally have no adverse effect on the pointers of other processes also using the files.

The only exception is that one process may delete a record currently pointed to by another process, thus causing the current record pointer of the other process to become undefined.

Information Facility

To design generic KSAM programs, use the information facility provided in KSAM. The _KINFO SVC provides information regarding the structure and current makeup of a previously created file.

Information available includes number of keys defined, their definitions, number of levels in each B-tree, number of currently active and inactive data records, number of currently active and inactive key blocks, data record size, etc.

Reading and Positioning File Pointer

The data file may be read sequentially in either direction on any key.

Calls exist that move the current position file pointer to the beginning or end of the file based on any key. By using all or part of any key, you can find records at random.

> NOTE: Partial keys for integer and longword keys must be multiples of two and four bytes respectively.

Hardware/Software Requirements

The KSAM routines run on all WICAT computers. The software supports the normal disk class handler and an additional class handler that may be loaded at boot time. The routines are not disk hardware specific, but rely upon other more primitive routines in the disk class handler and device driver(s) for actual device accesses. The keys file and data file can reside on different devices, but they must reside on the same machine.

KSAM Data File Description

The KSAM data file is divided into records of between 4 and 65535 user accessible bytes inclusive.

The first byte of every record is reserved for use by the KSAM to determine whether a record contains valid data.

If the first byte is 0, the record contains valid data.

If the first byte contains a 1, the record has been deleted.

Deleted records are linked using the first user accessible longword as a link field.

The link contains the relative byte address of the beginning of the next free record in the file.

The last deleted record in the file contains a forward pointer of -1.

If a KSAM data file contained six records, each with 15 user accessible bytes and only three of the records contained valid data, the file might look like this:

```
Byte address   Flag byte  Pointer/user data
------------------------------------------------------------------------

$00000000      $01        $00 00 00 30 xx xx xx xx xx xx xx xx xx xx xx
(This record is deleted, "xx" is leftover user data)

$00000010      $00        $uu uu uu uu uu uu uu uu uu uu uu uu uu uu uu
(Valid record; "uu" is user data)

$00000020      $00        $uu uu uu uu uu uu uu uu uu uu uu uu uu uu uu
(Valid record; "uu" is user data)

$00000030      $01        $00 00 00 40 xx xx xx xx xx xx xx xx xx xx xx
(This record is deleted; "xx" is leftover)

$00000040      $01        $FF FF FF FF xx xx xx xx xx xx xx xx xx xx xx
(This record is deleted; "xx" is leftover)

$00000050      $00        $uu uu uu uu uu uu uu uu uu uu uu uu uu uu uu
(Valid record; "uu" is user data)
```

KSAM Keys File Description

The KFCB comprises an in-core (memory only) section and a key file (disk) section. The following describes the disk section.

Key File Information

The key file contains the following information on the keys in the KSAM file. The number in the upper left-hand corner is the byte offset physically stored in the beginning of the keys file. The actual key definitions begin at offset 38.

0 - word

Length of the keys definition area of the file

2 - word

Data record length

4 - longword

Number of active records in the data file

8 - longword

Number of deleted records in data file

12 - longword

Number of active keyblocks in file

16 - longword

Number of deleted keyblocks in file

20 - longword

Byte address in the data file where the next unallocated data record starts (writing to this address causes the data file to be extended).

24 - longword

Byte address where last deleted data record starts; this last data record should contain a forward pointer to other existing deleted data records. A -1 is used as a null pointer.

28 - longword

Byte address where the next unallocated keyblock starts; writing to this address causes the key file to be extended.

32 - longword

Byte address where the last deleted keybock starts; this last block should contain a forward pointer to other existing deleted keyblocks. A -1 is used as a null pointer.

36 - word

Number of keys defined for this KSAM file

Definitions

38 - word

The first portion of the definitions contains a pointer (1 word/pointer) to the beginning of each key definition.

40 - word

If more than one key is defined, the offset of the key description for key 2 is contained here. Each key description offset occupies a successive word in memory; thus, the offset of key 3's description is contained at 42, key 4's at 44, etc.

The following repeats once for each key beginning at the offset for each key as specified above, minus hex 1A or decimal 26. The length of these key descriptions depends on the number of segments that constitute the key.

0 - byte

The high order byte for this word contains the number of levels in the B-tree for this key

1 - byte

Bit seven of this word is a flag bit; if set, the flag stipulates duplicate keys are disallowed. Bits four through six are the key type as described in _KCREAT call. The low order nibble contains the number of segments in this key.

2 - word

High byte is reserved. The low byte contains the length of this key in bytes.

4 - word

This word contains the maximum number of keys of this size that will fit into a 1024-byte keynode or leaf keyblock.

6 - longword

This longword is the byte pointer to the root node for this key within the key file.

10 -

The key segment definitions begin here. The segment definition consists of the byte offset within a record where a field begins and the length of the field in bytes. The pointer is a word, and the length is a word. The byte offset ignores the flag byte at the beginning of the record, i.e., the first data byte has an offset of zero.

Repeating Structure: The following two-word sequence repeats between 1 and 15 times for each key segment (i.e., there can be up to 15 segments for each key).

1. First word is the byte address within a data record where the segment starts.

2. Second word is the length of the segment in bytes.

Definition Area

The key definition area is null padded and rounded up to a 1024 byte boundary so that disk accesses on the key blocks are optimized during run time because they will fall on sector boundaries.

Immediately following the definition area, the node keyblocks and leaf keyblocks begin.

Keyblocks

The organization of each 1024 byte keyblock depends upon the size of the key it holds.

NODE KEYBLOCK. Each node keyblock contains the following:

A counter that indicates how many keys are kept in this node.

A pointer to the byte address in the key file where the parent block occurs (the parent of the root node contains a zero at this location).

A pointer to the byte address in the key file where the left brother block occurs.

A pointer to the byte address in the key file where the right brother block occurs.

NOTE: If no brother occurs, a zero is placed in those pointer locations.

Following the parent and brother pointers is a key structure that is repeated for each key value found in the node.

The first longword is a pointer to the byte address in the key file where the keyblock starts that contains keys less than or equal to the key value that follows the pointer.

Byte keys that have an odd length are right null padded.

Following the last key structure is a longword pointer containing the byte address in the key file where the block may be found containing keys higher that those found in the block pointed to by the last key-pointer structure in the present node.

LEAF KEYBLOCK. The leaf keyblock of the KSAM key file is similar to the node keyblock.

However, now a one-to-one correspondence exists between a pointer-key structure and a single record in the data file.

The key portion of the pointer-key structure in a leaf node is the exact data contained in the key-field area in the data record.

The pointer points to the byte address (the delete flag byte) of the record containing that key value.

As in the node keyblock, odd length keys are right null padded to an even boundary.

A leaf keyblock can be recognized by the set flag bit in bit 15 of the pointer-key structure count found in the first word of the block.

A leaf keyblock does not have a "greater than" pointer after the last pointer-key structure in the block.

KSAM Sample Program

The following KSAM program demonstrates how several SVC calls can be used to set up a phone list that can be accessed by name, address, city, state, zip code, or phone number.

```
        program record
        integer*4  lun,status
        integer*2  choice

        call getfil(lun)
100     continue
        call menu
110     continue
        call gotoxy(17,10)
        write (6,10) ' Enter choice:  '
        call getchr(choice)
        if ((choice .GE. 97).AND.(choice .LE. 122)) choice = choice - 32
        if (choice .EQ. 65) then
          call add(lun)
        else if (choice .EQ. 68) then
          call delete(lun)
        else if (choice .EQ. 70) then
          call find(lun)
        else if (choice .EQ. 83) then
          call show(lun)
        else if (choice .EQ. 81) then
          goto 1000
        else
          goto 110
        endif
        goto 100
1000    continue
        call kclose(lun,0,status)
        call clrscr
10      format(A)
30      format(1x,I5.1)
        end

        subroutine getfil(lun)
        character*94    fname,kfname
        integer*2       ktable(25)
        integer*4       mode,reclen,prot,numbuf,lun,status

        fname = 'phone.dat'
        kfname = 'phone.key'

        mode       = 3                          !allow read & write access
        numbuf     = 0                          ! use default buffer size

        call kopen(fname,kfname,mode,numbuf,lun,status)

        if (status .NE. 0) then
          reclen     = 81        !record length is 81 bytes
          prot       = -1        ! use default protection

          ktable(1)  = 6         !define 6 different keys

          ktable(2)  = 1         !char field, 1 segment,allow duplicates
          ktable(3)  = 20        !key is 20 bytes long
          ktable(4)  = 0         !begins in position 0 of record
          ktable(5)  = 20        !this segment runs for 20 bytes.

          ktable(6)  = 1         !char field, 1 segment,allow duplicates
          ktable(7)  = 24        !key is 24 bytes long
          ktable(8)  = 20        !begins in position 20 of record
          ktable(9)  = 24        !this segment runs for 24 bytes.

          ktable(10)  = 1        !char field, 1 segment,allow duplicates
          ktable(11)  = 20       !key is 20 bytes long
          ktable(12)  = 44       !begins in position 45 of record
          ktable(13)  = 20       !this segment runs for 20 bytes.
```

```
            ktable(14)  = 1       !char field, 1 segment,allow duplicates
            ktable(15)  = 2       !key is 2 bytes long
            ktable(16)  = 64      !begins in position 65 of record
            ktable(17)  = 2       !this segment runs for 2 bytes.

            ktable(18)  = 1       !char field, 1 segment,allow duplicates
            ktable(19)  = 5       !field is 5 bytes long
            ktable(20)  = 66      !begins in position 67 of record
            ktable(21)  = 5       !this segment runs for 5 bytes.

            ktable(22)  = 1       !char field, 1 segment,allow duplicates
            ktable(23)  = 10      !field is 10 bytes long
            ktable(24)  = 71      !begins in position 72 of record
            ktable(25)  = 10      !this segment runs for 10 bytes.

            reclen = 81           !record length is 81 bytes
            prot   = -1           ! use default protection
            numbuf = 0            ! use default buffer size

            call kcreat(fname,kfname,mode,reclen,prot,numbuf,
     +                  ktable,lun,status)
          endif
          return
          end

          subroutine add(lun)
          character       buf*81,addres*24,name*20,city*20,state*2
C         character*20    name,city
C         character*2     state
          character*5     zip
          character*10    phone
          integer*4       lun,status

          equivalence (buf(1:1),name)          ! Name is position 1-20,
          equivalence (buf(21:21),addres)      ! addres is position 21-44,
          equivalence (buf(45:45),city)        ! city is position 45-64,
          equivalence (buf(65:65),state)       ! state is position 65-66,
          equivalence (buf(67:67),zip)         ! zip is position 67-71,
          equivalence (buf(72:72),phone)       ! phone is position 72-81,
C                                              ! of the record 'buf'.

100       continue
          write (6,10) ' Enter client\'s name: '
          read (5,10) name

110       continue
          write (6,10) ' Enter client\'s address: '
          read (5,10) addres
```

```
120     continue
        write (6,10) ' Enter client\'s city: '
        read (5,10) city

130     continue
        write (6,10) ' Enter client\'s state: '
        read (5,10) state

140     continue
        write (6,10) ' Enter client\'s zip: '
        read (5,10) zip

150     continue
        write (6,10) ' Enter client\'s phone number (8012246882): '
        read (5,10) phone

        call kwrite(lun,-1,buf,status)
        if (status .ne. 0) write (6,20) status,' KWRITE'
1000    continue
        return
10      FORMAT(A)
20      format(1x,I5,A)
        end

        subroutine delete(lun)
        integer*4 lun,status
        integer*2 value
        call kdelet(lun,-1,status)

        if (status .EQ. 0) then
          write (6,10) ' Record deleted.'
        else
          write (6,10) ' Error in deleting record.  Error = ',status
        endif
        write (6,10) ' Hit any key to continue.'
        call getchr(value)
10      format(A,I5)
        return
        end

        subroutine find(lun)
        character*81    buf
        character*24    dummy,title(6)
        integer*4       keynum,lengt,lun,status,index
        integer*2       length(6),choice,value

        data length/20,24,20,2,5,10/
        data title /'name','address','city','state','ZIP','phone'/
100     continue
        call fdmenu
```

```
110     continue
        call gotoxy(13,10)
        write (6,10) ' Enter Choice: '
        call getchr(choice)

        if ((choice .GE. 97).AND.(choice .LE. 122)) choice = choice - 32
        if (choice .EQ. 78) then
          index = 1                               ! Search by name.
        else if (choice .EQ. 65) then
          index = 2                               ! Search by address.
        else if (choice .EQ. 67) then
          index = 3                               ! Search by city.
        else if (choice .EQ. 83) then
          index = 4                               ! Search by state.
        else if (choice .EQ. 90) then
          index = 5                               ! Search by ZIP.
        else if (choice .EQ. 80) then
          index = 6                               ! Search by phone.
        else if (choice .EQ. 70) then
          write (6,10) ' '
          goto 200                                ! Get next occurrence.
        else if (choice .EQ. 81) then
          goto 1000
        else
          goto 110
        endif
        keynum = index - 1
        call trim(title(index),title(index),lengt)
        write (6,10) ' Enter ',title(index)(1:lengt),' to search for: '
        read (5,10) dummy

        call trim(dummy,dummy,lengt)
        call kfind(lun,keynum,dummy(1:lengt),lengt,status)
200     continue
        call kread(lun,1,-1,buf,status)
        if (status .NE. 140) then
          call output(buf)
        else
          write (6,20) ' Error in reading record.  Error = ',status
        endif
          write (6,10) ' Touch any key to continue.'
          call getchr(value)
        goto 100
1000    continue
10      format(5A)
20      format(1x,A,5x,I5)
        return
        end
        subroutine show(lun)
        character*81    buf
        integer*4       keynum,lun,status,count
        integer*2       choice,value

100     continue
        call shmenu
110     continue
        call gotoxy(13,10)
        write (6,10) ' Enter Choice: '
        call getchr(choice)

        if ((choice .GE. 97).AND.(choice .LE. 122)) choice = choice - 32
        if (choice .EQ. 78) then
          keynum = 0                          ! Show by name.
        else if (choice .EQ. 65) then
          keynum = 1                          ! Show by address.
        else if (choice .EQ. 67) then
          keynum = 2                          ! Show by city.
        else if (choice .EQ. 83) then
          keynum = 3                          ! Show by state.
        else if (choice .EQ. 90) then
          keynum = 4                          ! Show by ZIP.
        else if (choice .EQ. 80) then
          keynum = 5                          ! Show by phone.
        else if (choice .EQ. 81) then
          goto 1000
        else
          goto 110
        endif
        call kmovfb(lun,keynum,0,status)
        call clrscr
        count = 0
200     continue
        call kread(lun,1,-1,buf,status)
        if (status .EQ. 0) then
          call output(buf)
          count = count + 1
          if (count .GE. 4) then
            write (6,10) ' *'
            call getchr(value)
            count = 0
          endif
          goto 200
        endif
        if (count .NE. 0) then
          write (6,10) ' *'
          call getchr(value)
        endif
1000    continue
```

```
10      format(A)
        return
        end

        subroutine menu
        call clrscr

        write (6,10) ' '
        write (6,10)
     +        '**************************************************'
        write (6,10)
     +        '*                                                *'
        write (6,10)
     +        '*            PHONE DIRECTORY MAIN MENU           *'
        write (6,10)
     +        '*                                                *'
        write (6,10)
     +        '* A = Add a new record.                          *'
        write (6,10)
     +        '* D = Delete the current record. (on screen).    *'
        write (6,10)
     +        '* F = Find a record.                             *'
        write (6,10)
     +        '* S = Show records by name, city, zip, or phone. *'
        write (6,10)
     +        '* Q = Quit.                                      *'
        write (6,10)
     +        '*                                                *'
        write (6,10)
     +        '**************************************************'
        return
10      format(13x,A)
        end

        subroutine fdmenu
        call clrscr

        write (6,10) ' '
        write (6,10) '              SEARCHING MENU'
        write (6,10) ' '
        write (6,10) ' N = Search by name.'
        write (6,10) ' A = Search by address.'
        write (6,10) ' C = Search by city.'
        write (6,10) ' S = Search by state.'
        write (6,10) ' Z = Search by ZIP.'
        write (6,10) ' P = Search by phone number.'
        write (6,10) ' F = Find next occurrence of record.'
        write (6,10) ' Q = Return to main menu.'
        return
10      format(13x,A)
        end
        subroutine shmenu
        call clrscr

        write (6,10) ' '
        write (6,10) '                SHOW MENU'
        write (6,10) ' '
        write (6,10) ' N = Show alphabetically by name.'
        write (6,10) ' A = Show alphabetically by address.'
        write (6,10) ' C = Show alphabetically by city.'
        write (6,10) ' S = Show alphabetically by state.'
        write (6,10) ' Z = Show numerically by ZIP.'
        write (6,10) ' P = Show numerically by phone number.'
        write (6,10) ' Q = Return to main menu.'
        return
10      format(13x,A)
        end

        subroutine output(buf)
        character*81    buf,bufl
        character*24    addres
        character*20    name,city
        character*2     state
        character*5     zip
        character*10    phone

        equivalence (bufl(1:1),name(1:1)),(bufl(21:21),addres(1:1))
        equivalence (bufl(45:45),city(1:1)),(bufl(65:65),state(1:1))
        equivalence (bufl(67:67),zip(1:1)),(bufl(72:72),phone(1:1))

        bufl = buf
        write (6,10) name,'\n'
        write (6,10) addres,'\n'
        write (6,10) city,state,'     ',ZIP,'\n'
        write (6,10) '(',phone(1:3),') ',phone(4:6),'-',phone(7:10),'\n'
        write (6,10) ' '
10      format(7A)
        return
        end

        subroutine getchr(valuel)

C ** This subroutine gets one character from the keyboard and returns
C    the ASCII value into the variable passed.  Control is returned
C    to the program immediately after the character is typed (No
C    Carriage Return <CR> needed.

        implicit undefined (a-z)
        integer*4 nrecs,status
        integer*2  valuel,value2
        character*1 value
        equivalence (value2,value)
        call read(1,-1,0,-1,value,1,nrecs,status)
        value1 = value2 / 256
        return
        end

        subroutine trim (input,output,length)
C
C ** This subroutine is used to get the actual length of the variable.
C ** It receives The variable in input and returns the variable in output.
C ** With the actual length returned in length.
C
        character*(*)   input,output
        integer         length

        length = len(input)
100     continue
        if (input(length:length) .EQ. ' ') then
          length = length - 1
          goto 100
        endif
        output = input(1:length)
        return
        end

        subroutine gotoxy(line,column)

C ** This routine puts the cursor at a location on the screen.  The screen has
C    25 lines and 80 columns.
C
C ** This routine is unique to the WICAT T7000, or MG8000 terminal.  You may
C    need to rewrite the subroutine for your specific terminal.

        integer*4 line,column
        character*1 esc
        data esc /z'1B'/
        write (6,10) esc,"[",line,";",column,"H"
10      format(2A,I2.2,A,I2.2,A)
        return
        end

        subroutine clrscr

C ** This routine clears the screen and puts the cursor at location 1,1 on the
C    screen.  The screen has 25 lines and 80 columns.
C
C ** This routine is unique to the WICAT T7000, or MG8000 terminal.  You may
C    need to rewrite the subroutine for your specific terminal.
        character*1 esc
        data esc /z'1B'/
        write (6,10) esc,'[2J'
10      format(2A)
        return
        end
```

Appendix A

Directory of System Calls

_ALARM	- Set alarm clock
_ALLOC	- Allocate a device
_ALLMEM	- Allocate dynamic memory
_ANDEVNT	- Wait for AND of event flags
_ASSIGN	- Assign a logical name
_CHDIR	- Set default device and directory
_CHSUPER	- Change to supervisor mode
_CHUSER	- Change processor mode to user
_CLONE	- Create a duplicate process
_CLOSE	- Close a file
_CLREVNT	- Clear event flags
_CONNECT	- Establish connection with remote machine
_CREATE	- Create a file
_CREATS	- Simplified file creation
_CRPRCS	- Simplified create process
_CRPROC	- Create a new process
_CRSHDP	- Enable/disable crash display
_CTRLC	- Set/clear [CTRL] c protection
_DCONALL	- Disconnect all remote process connections
_DCONIDLE	- Disconnect idle remote process connections
_DEALLOC	- Deallocate an allocated device
_DEFDPRT	- Set default device protection
_DEFDUIC	- Set default device UIC
_DEFMEM	- Define named shared memory area
_DEFPROT	- Set default protection mask
_DEINST	- Deinstall privileged file
_DELETE	- Delete a file
_DISCONN	- Break connection to remote machine
_DISMNT	- Dismount a logical device
_DUPLUN	- Duplicate Logical Unit number of a file
_ERRNO	- Receive process abort reason
_EXITRTN	- Define a returnable exit handler
_EXPROC	- Terminate the specified process
_FLUSH	- Flush I/O buffers to the device
_FRDWAIT	- Wait for fast read to complete

_FREMEM	- Deallocate a page of memory
_GASSIGN	- Assign a global logical name
_GENGY	- Get PID of ancestor process
_GETALC	- Get names of allocated devices
_GETATTR	- Get PCB attribute bits
_GETDIR	- Get default device and directory
_GETDNAM	- Get device name
_GETDPRT	- Get device protection
_GETDST	- Get device status
_GETDUIC	- Get device UIC
_GETEVNT	- Read event flags
_GETEXIT	- Get address of current exit handler
_GETFCB	- Get file control block
_GETFID	- Get file ID
_GETFNAM	- Given a lun, return the filename
_GETFPRT	- Get file protection
_GETFRE	- Get amount of available memory
_GETFRSZ	- Get file record size
_GETFUIC	- Get file UIC
_GETGLB	- Retrieve a global logical name
_GETINST	- Get installed privileged image
_GETLOG	- Retrieve a logical name
_GETMLST	- Get a entry from list of named shared memory areas
_GETNNAM	- Get network nodename given site ID
_GETNSID	- Get network site ID give nodename
_GETPCB	- Get process control block
_GETPID	- Get process ID (PID) from name
_GETPNAM	- Get process name from PID
_GETPOS	- Get the current file position
_GETPRI	- Get process's priority
_GETPROT	- Get default protection mask
_GETPRV	- Get process privilege
_GETREL	- Get names of rotor list elements
_GETRTR	- Get rotor list names
_GETTIC	- Get internal tick count
_GETTIM	- Get the current date and time
_GETTMSL	- Get scheduling time slice
_GETUIC	- Get process UIC
_GIODST	- Get device status with lun
_GMAIL	- Receive interprocess mail
_HIBERN	- Hibernate a process
_INSTALL	- Install privileged file
_KCLALL	- Close all KSAM files
_KCLOSE	- Close a KSAM file
_KCREAT	- Create a KSAM file
_KDELET	- Delete a KSAM record
_KFIND	- Locate a KSAM record
_KFLUSH	- Write modified KSAM buffers
_KINFO	- Retrieve KSAM file information
_KMOVFB	- Position to front or back of file

_KOPEN	- Open a KSAM file
_KREAD	- Read a KSAM record
_KUNLCK	- Unlock specified KSAM records
_KUPDAT	- Update an existing KSAM record
_KWRITE	- Write a new KSAM record
_LOCK	- Lock records within an open file
_MAPFP	- Maps physical address into logical address space (floating point)
_MAPPHYS	- Maps physical address into logical address space
_MEMMNT	- Mount a logical device from memory
_MOUNT	- Mount a logical device
_MULCRPS	- Multiple create process
_OPEN	- Open a file
_OREVNT	- Wait for OR of event flags
_ORIGPRV	- Get original process privilege
_PHYSIO	- Perform physical I/O operation
_PHYSOP	- Perform physical device operation
_PIDLST	- Get list of all known PIDs on the system
_PRCLST	- Get PID's on a priority level
_PRIRAT	- Set priority scheduling ratio
_PROTMEM	- Change memory page protection
_RDPMEM	- Read physical memory
_READ	- Read from an open file
_RENAME	- Rename a file
_RNIDLST	- Get list of all known remote IDs
_RSIDLST	- Get list of all known SIDs given remote network
_SETATTR	- Set PCB attribute bits
_SETDPRT	- Set device protection
_SETDST	- Set device status
_SETDUIC	- Set device UIC
_SETEVNT	- Set event flags
_SETEXIT	- Define exit handler
_SETFCB	- Write file control block
_SETFID	- Set file ID
_SETFPRT	- Set file protection
_SETFRSZ	- Set file record size
_SETFUIC	- Set file UIC
_SETMPRT	- Change access protection of a named shared memory area
_SETMUIC	- Set named memory area UIC
_SETPNAM	- Change process name
_SETPOS	- Set the current file position
_SETPRI	- Change process's priority
_SETPRV	- Set process privilege
_SETRTM	- Set/Clear real time mode flag
_SETRTR	- Assign device names to a rotor list
_SETTIM	- Set system date and time
_SETTMSL	- Change scheduling time slice
_SETTRP	- Initialize a user defined trap
_SETUIC	- Set process UIC
_SHRMEM	- Share a named shared memory area
_SIDLST	- Get list of all known Site IDs

_SIODST	- Set device status with lun
_SKIP	- Position tape
_SMAIL	- Send interprocess mail
_TRANPID	- Translate another processes logical name
_TRANS	- Translate a logical name
_UDEFMEM	- Undefine a named shared memory area
_UNLOCK	- Unlock records in an open file
_USHRMEM	- Unshare a named shared memory area
_VERSION	- Get the OS version banner
_WAIT	- Pause for a period of time
_WAKE	- Wake a hibernated process
_WAKEC	- Wake a hibernated process with count
_WRITE	- Write to an open file
_WTPMEM	- Write physical memory

Appendix B

Glossary of WMCS Diagnostic Messages

0 The specified operation was performed successfully (i.e., the request made of the WMCS was successfully completed).

Diagnostic name:

KERNEL Diagnostic Messages

1 The process lacks the privileges required to perform the operation.

Each process is assigned a set of privileges, i.e., rights or prerogatives within the WMCS. When a process asks the WMCS to do something, the process must have the right or prerogative to make such a request. Otherwise, the requested operation is not performed.

Diagnostic name: **errinsufpriv**

2 The specified process is not in the system process table.

The WMCS assigns a PID (process identification number) to each process. Some requests require that the PID of the target process be specified as part of the request. This status is assigned to a request when the specified PID does not match the PID of any of the current processes, i.e., the PID could not be found in the system's list, or table, of current processes.

Diagnostic name: **errprcsnotfnd**

3 The process's buffer does not begin on a word boundary.

Several of the system calls require that buffers provided by the calling process be word aligned. This diagnostic message is sent when the WMCS expects a word aligned buffer and the buffer provided by the calling process is not word aligned.

Diagnostic name: **erroddbufaddr**

4 The logical address, for the memory requested, is invalid.

The WMCS routines that deal with the allocation and deallocation of memory require that a page address be designated. The page address must be within the logical address space of the calling process and must begin on a 4 Kbyte boundary. Otherwise, this message is specified.

Diagnostic name: **errinvadr**

5 The process requested a logical page that was already allocated.

WICAT hardware allows processes to assign pages of physical memory to addresses within the logical address space of a process. This status is assigned when a process specifies a logical address to which memory is already assigned.

Diagnostic name: **errmemalloc**

6 The process tried to affect a page in memory it did not own.

A process may not own all the pages of memory in its logical address space. For example, the pure code for a process can be shared with another process (this sharing is effected when the latter process is created). When pages are shared, neither process owns the page, and attempts to deallocate or modify the protection assigned to these pages results in this status.

Diagnostic name: **errnonowned**

7 All available memory has been allocated.

Memory is allocated for processes (process creation, explicit calls for memory allocation, automatic stack growth), as well as for the system itself (process tables, disk caches, device drivers, open files, etc.). This status is assigned to a process whose request requires more memory than is available.

Diagnostic name: **errnomemavail**

8 The specified site ID does not exist.

Each WICAT system has a site identification number that identifies the system. Several system calls allow a process to specify the ID numbers of the sites that will be affected by a request. This status is assigned when the WMCS does not recognize the specified site ID number.

Diagnostic name: **errinvsiteid**

9 The process attempted to release memory that does not exist.

The logical address space assigned to the process does not include the address of the page that the process asked the WMCS to release, i.e., the process does not have a page of memory at the address specified.

Diagnostic name: **errmemdeall**

10 An arithmetic operation produced a number longer than 32 bits.

An arithmetic overflow occurred when the WMCS tried to calculate the position (within a file) of a particular record. In other words, the calculation resulted in a number that could not be contained in 32 bits.

Diagnostic name: **errartiovflw**

11 No number was found during a search or scan for a number.

The WMCS had to convert an ASCII string to a binary number, e.g., a file version number.

This status is never returned to a user process.

Diagnostic name: **errnonumber**

12 The file type is inappropriate for the given operation.

The specified operation requires that the target file be a particular kind of file, e.g., a process asks the WMCS to install a file that is neither an image file nor a system file.

Diagnostic name: **errinapft**

13 The specified process already exists.

This diagnostic is returned to the process that tries to create an additional copy of a system process (e.g., logflush).

Diagnostic name: **errprcsfnd**

14 A negative number is not allowed for this parameter.

A process specified a negative number for a parameter where a negative number is invalid.

Diagnostic name: **errnegnumber**

15 Trap number (during _SETTRP) exceeds range of specifiable numbers.

The process issued a _SETTRP system call and asked the WMCS to define a trap whose number does not correspond to a user-defineable trap.

Diagnostic name: **errbadtrapnum**

16 The specified device is not allocated.

This diagnostic is returned when a process requests that a device be deallocated, and it is not currently allocated.

Diagnostic name: **errnotalloc**

17 Insufficient memory to automatically extend the users stack.

The user process has exceeded its user stack space. The WMCS has attempted to extend the stack space on behalf of the process, but there is no unallocated memory available. The process is terminated.

Diagnostic name: **errautostackexp**

18 The specified rotor list is empty.

This diagnostic is returned when the process attempts to allocate a device using a rotor name as the name of the device, and there are no available devices in the list associated with that rotor.

Diagnostic name: **erremptyrtrlst**

19 The process was terminated because the remote connection was lost.

This diagnostic is generated if a process is terminated because the connection from a port to a remote port is lost, and the local port is configured to delete processes associated with that port. This diagnostic is reported to the process that is terminated, and, if the process does not have an exit handler defined, it is reported to the parent process in the ccode parameter of the _CRPROC system call.

Diagnostic name: **errremoteabort**

20 No interprocess mail, in system message table, for the process.

The process issued a _GMAIL system call when no interprocess mail had been sent to the process.

Diagnostic name: **errnomail**

21 The specified file is not an image file.

Some system functions (create, mount, allocate, etc.,) are defined only for image files. For example, a process might attempt to create a process by specifying a file that is not an image file.

Diagnostic name: **errnotimfle**

22 The queue control file is being deleted at the users request.

This diagnostic is returned as a warning when the user specifies the :norestart switch on the MNT command to mount a queue, and the queue manager deleted the queue control file. The queue control file contains the list of queue entries.

Diagnostic name: **errqctdeluser**

23 The queue control file is being deleted. It may be corrupted.

This diagnostic is returned as a warning when a process mounts a queue, and the queue manager discovers that the queue control file contains erroneous data. The queue control file is deleted, losing any entries that may have been there.

Diagnostic name: **errqctdelupdat**

24 The process has already allocated floating point hardware.

This diagnostic is returned when the process attempts to map floating point hardware into its address space after it has already done so. That is, a process cannot map floating point hardware more than once.

Diagnostic name: **errhavemath**

25 The process has an undefined trap: floating point.

A floating point exception has occurred. Note that if the process had set up a floating point trap handler (using _SETTRP), the floating point exception would have caused a transfer to the floating point trap handler rather than the termination of the process.

If the process was spawned, this status is sent to the ccode parameter of the _CRPROC system call to the parent process.

A process crash display, or stack dump, is also written to the SYS$ERROR file assigned to the terminated process.

Diagnostic name: **errflpointtrap**

26 The process abort status was forced to a normal exit status.

A process can specify this diagnostic code in the result parameter of the _exproc system call to force a normal exit status (0) overriding the current process abort reason code.

Diagnostic name: **errnormalexit**

27 The process was killed by another process.

This status results when a process is terminated by another process, e.g., when a process is terminated by the KILL Command.

If the terminated process was spawned (and is terminated by a third process that called _EXPROC) this status is sent to the ccode parameter of the _CRPROC system call to the parent process.

Diagnostic name: **errforcedexit**

28 The system clock reached the value specified for _ALARM.

A process set a duration period, i.e., an alarm, and the duration expired before the process was completed.

If the process was spawned, this status is sent to the ccode parameter of the _CRPROC system call to the parent process.

Diagnostic name: **erralarmexit**

29 The process has an undefined trap: divide-by-zero.

The process attempted to divide by zero. Note that if the process had set up a divide-by-zero trap handler (using _SETTRP), the attempt to divide by zero would have caused a transfer to the trap handler rather than the termination of the process.

If the process was spawned, this status is sent to the ccode parameter of the _CRPROC system call to the parent process.

A process crash display, or stack dump, is also written to the SYS$ERROR file assigned to the terminated process.

Diagnostic name: **errzerodivtrap**

30 The process has an undefined trap: CHK Instruction.

This status results when a process is terminated because it generated a CHK trap. Note that if the process had set up a CHK trap handler (using _SETTRP), the CHK trap would have caused a transfer to the trap handler rather than the termination of the process.

If the process was spawned, this status is sent to the ccode parameter of the _CRPROC system call to the parent process.

A process crash display, or stack dump, is also written to the SYS$ERROR file assigned to the terminated process.

Diagnostic name: **errchktrap**

31 The process has an undefined trap: TRAPV Instruction.

This status results when a process is terminated because it generated a TRAPV trap. Note that if the process had set up a TRAPV trap handler (using _SETTRP), the TRAPV would have caused a transfer to the trap handler rather than the termination of the process.

If the process was spawned, this status is sent to the ccode parameter of the _CRPROC system call to the parent process.

A process crash display, or stack dump, is also written to the SYS$ERROR file assigned to the terminated process.

Diagnostic name: **errtrapvtrap**

32 The process has an undefined trap: TRACE.

The process generated a TRACE trap and was terminated. Note that if the process had set up a TRACE trap handler (using _SETTRP), the TRACE trap would have caused a transfer to the trap handler rather than the termination of the process.

If the process was spawned, this status is sent to the ccode parameter of the _CRPROC system call to the parent process.

A process crash display, or stack dump, is also written to the SYS$ERROR file assigned to the terminated process.

Diagnostic name: **errtracetrap**

33 The process has an undefined trap: 1010 Instruction.

The process was terminated because it attempted to execute a 1010 instruction. Note that if the process had set up a 1010 trap handler (using _SETTRP), the 1010 instruction would have caused a transfer to the trap handler rather than the termination of the process.

If the process was spawned, this status is sent to the ccode parameter of the _CRPROC system call to the parent process.

A process crash display, or stack dump, is also written to the SYS$ERROR file assigned to the terminated process.

Diagnostic name: **err1010trap**

34 The process has an undefined trap: 1111 Instruction.

The process was terminated because it attempted to execute an 1111 instruction. Note that if the process had set up an 1111 trap handler (using _SETTRP), the 1111 instruction would have caused a transfer to the trap handler rather than the termination of the process.

If the process was spawned, this status is sent to the ccode parameter of the _CRPROC system call to the parent process.

A process crash display, or stack dump, is also written to the SYS$ERROR file assigned to the terminated process.

Diagnostic name: **err1111trap**

35 The process attempted to execute a privileged instruction.

The process attempted to execute a privileged instruction while the process was in user mode. These are the privileged instructions:

```
STOP
MOVE xx,SR
AND  xx,SR
OR   xx,SR
XOR  xx,SR
MOVE Ax,USP
MOVE USP,Ax
RESET
RTE
```

If the process was spawned, this status is sent to the ccode parameter of the _CRPROC system call to the parent process.

A process crash display, or stack dump, is also written to the SYS$ERROR file assigned to the terminated process.

Diagnostic name: **erprivintrap**

36 The process attempted to execute an illegal instruction.

The process attempted to execute an illegal instruction, i.e., an instruction that is not recognized by the MC68000 microprocessor.

If the process was spawned, this status is sent to the ccode parameter of the _CRPROC system call to the parent process.

A process crash display, or stack dump, is also written to the SYS$ERROR file assigned to the terminated process.

Diagnostic name: **errillintrap**

37 The process accessed nonexistent physical memory (bus error).

The process generated a bus error. A bus error pertains to the system's hardware, and is any access to an address at which there is no memory or memory mapped device.

The process is also terminated, unless the bus error results from a call to _WTPMEM or _RDPMEM.

If the process was terminated (and was a spawned process), this status is sent to the ccode parameter of the _CRPROC system call to the parent process.

A process crash display, or stack dump, is written to the SYS$ERROR file assigned to the process (if the process is terminated).

Diagnostic name: **errbustrap**

38 The process accessed a word on a byte boundary (address error).

The process caused an address error. An address error is generated by the MC68000 microprocessor when when the address of a word or long word operand is odd rather than even.

If the process was spawned, this status is sent to the ccode parameter of the _CRPROC system call to the parent process.

A process crash display, or stack dump, is also written to the SYS$ERROR file assigned to the terminated process.

Diagnostic name: **erradrtrap**

39 The process accessed nonexistent logical memory (memory violation).

The process caused a memory violation. A memory violation occurs when a process attempts to access an address in logical memory ($000000 - $1FFFFF) to which no memory has been assigned, or when a process in user mode accesses any address greater than $1FFFFF.

If the process was spawned, this status is sent to the ccode parameter of the _CRPROC system call to the parent process.

A process crash display, or stack dump, is also written to the SYS$ERROR file assigned to the terminated process.

Diagnostic name: **errnonexmem**

40 The process has a memory parity-error.

The process was terminated because it attempted to access a location in memory whose contents had been altered because some aspect of the system's hardware malfunctioned.

If the process was spawned, this status is sent to the ccode parameter of the _CRPROC system call to the parent process.

A process crash display, or stack dump, is also written to the SYS$ERROR file assigned to the terminated process.

Diagnostic name: **errmemparity**

41 The process attempted to write to a write-protected page in memory.

The process attempted to write to an address on a write-protected page of logical memory. Write-protected portions of logical memory include the following:

Pure code shared by processes.

Pages write-protected by the process that owns them.

If the process was spawned, this status is sent to the ccode parameter of the _CRPROC system call to the parent process.

A process crash display, or stack dump, is also written to the SYS$ERROR file assigned to the terminated process.

Diagnostic name: **errwriteprot**

42 A handler was not defined before a TRAP instruction was executed.

The process attempted to execute a TRAP instruction for which it had not used the _SETTRP system call to define a trap handler.

If the process was spawned, this status is sent to the ccode parameter of the _CRPROC system call to the parent process.

A process crash display, or stack dump, is also written to the SYS$ERROR file assigned to the terminated process.

Diagnostic name: **errundeftrap**

43 The WMCS does not recognize the SVC number used by the process.

The process specified an undefined SVC number in an attempt to execute a system call.

If the process was spawned, this status is sent to the ccode parameter of the _CRPROC system call to the parent process.

A process crash display, or stack dump, is also written to the SYS$ERROR file assigned to the terminated process.

Diagnostic name: **errundefsvc**

44 The process has lost Data Set Ready on a tty line it controlled.

This diagnostic is generated if a process is terminated because the connection from a port to a remote port is lost, and the local port is configured to delete processes associated with that port. This diagnostic is reported to the process that is terminated, and, if the process does not have an exit handler defined, it is reported to the parent process in the ccode parameter of the _CRPROC system call.

Diagnostic name: **errdsrloss**

45 This item is not implemented yet.

This function has not yet been implemented, but will be implemented in a future release of WMCS.

Diagnostic name: **errnotimp**

46 The spawned child has terminated.

A child process created by a surrogate process has terminated. Note that when a process is spawned by a surrogate process, the surrogate process does not go to sleep in a child wait, but continues execution. When the spawned child process terminates, rather that waking up the parent process, the O.S. kills the surrogate process with this error code, and the surrogate process's exit handler then makes a reply packet to the remote parent process.

Diagnostic name: **errspawndone**

47 The process was not allowed to log on to the remote system.

The user ID from the given site ID of the process is not allowed to log on to the desired remote system.

Diagnostic name: **errremotelogon**

48 (WMCS error) Nondelete, or critical, count is too large (overflow).

This status represents an error within the WMCS, i.e., the user process is not at fault.

Diagnostic name: **errsetdelovfl**

49 (WMCS error) Nondelete, or critical, count is less than 0 (underflow).

This status represents an error within the WMCS, i.e., the user process is not at fault.

Diagnostic name: **errclrdelovfl**

50 Supervisor's stack does not contain enough parameters (underflow).

Two stacks are associated with each process:

user - used while the process is in user mode

system - used when the process calls the WMCS

This status results when the process expects data from the system stack and the system stack is empty.

This occurs when a process is running in supervisor mode and calls the WMCS without having pushed the requisite number of parameters.

Diagnostic name: **errprevalloc**

51 User's stack does not contain enough parameters (underflow).

Two stacks are associated with each process:

user - used while the process is in user mode system - used when the process calls the WMCS

This status results when the process is running in user mode and calls the WMCS without having pushed the requisite number of parameters.

Diagnostic name: **errustckunfl**

52 No network virtual circuits are available for this operation.

All of the network virtual circuits have already been allocated, leaving insufficient network virtual circuits available to make a connection to a remote machine in order to complete this operation.

Diagnostic name: **errnodevavail**

53 The specified node could not be found.

The system network tables do not contain an entry corresponding to the specified node name.

Diagnostic name: **errnonodefnd**

54 The originator process has been aborted.

The remote originator of a surrogate process has been terminated. (This error will never be received by a user process. It will only be sent by the OS to a surrogate process.)

Diagnostic name: **errorigterm**

55 Remote process creation is not allowed by the remote system.

The process does not have the privilege required on the remote system in order to create a process on the remote system.

Diagnostic name: **errnoremcrproc**

56 The table ends before the specified occurrence.

The following system calls enable a process to request an item from a system table:

_GETDNAM, _GETLOG, _GETINST

When a process uses one of these system calls, the process must specify the position in the list, or table, of the item.

This status results when the list is not long enough to include the ordinal number specified, e.g., a process requests the name of the fifth device in a particular list, and the list contains only four devicenames.

Diagnostic name: **erridxrange**

57 The siteid verification failed for the specified network node.

The site ID verification process for the specified network node failed.

Diagnostic name: **errsiteinvalid**

58 The priority ratio for the scheduler is less than or equal to zero.

The priority ratio is the number of processes, at any level of priority, that will be scheduled (for processing) for each process at the next lower level of priority.

This status results when the ratio assigned (using _PRIRAT) to the scheduler is less than or equal to zero.

Diagnostic name: **errpriorratio**

59 The address, sent to an SVC, exceeds user's logical address space.

The WMCS requires that all addresses that it receives from a user process be within the logical address space assigned to the user process ($000000 through $1FFFFF). This keeps processes from affecting the WMCS and other processes.

This status results when an address exceeds the logical address space for the user process.

Note that a process operating in supervisor mode can access addresses grater than $1FFFFF.

Diagnostic name: **erraddovfl**

60 The size, sent to an SVC, is out of range.

The size parameter associated with this system call contains an unacceptable value. For instance, this diagnostic is returned to the process attempting to share a named shared memory segment, and the size specified is zero or, the size specified is not large enough to accommodate the entire memory segment.

Diagnostic name: **errsizovfl**

61 An invalid value was specified.

The value specified for the parameter is not valid.

Diagnostic name: **errinvvalue**

62 The process was killed because of a SWAPPER I/O error.

The swapper was unable to read the process from the swap file to restore it to memory and allow it to run.

Diagnostic name: **errswapio**

63 (Floating point diagnostic) Illegal instruction given to FFP board.

An illegal operation code was sent to the FFP hardware floating point board.

Diagnostic name: **errffpillinst**

64 An invalid character appears in a decimal string.

This status results when a process asks the WMCS to convert a decimal ASCII string to a binary string (for example, a file version number) and the WMCS encounters a nonnumeric character in the string.

Diagnostic name: **errinvnumchar**

65 (Floating point diagnostic) Device does not respond.

This status indicates a malfunction in the floating point hardware.

A timer is set whenever the floating point processor is assigned a task. This status is assigned when the processor does not complete the task in the allotted time, and the process is terminated. All processes in the middle of a floating point operation when this status is assigned are also terminated.

If the process was spawned, this status is sent to the ccode parameter of the _CRPROC system call to the parent process.

Diagnostic name: **errmathbrdfail**

66 (Floating point diagnostic) Divide-by-zero error.

The process was terminated because it attempted to use the floating point software or hardware to divide by zero.

If the process was spawned, this status is sent to the ccode parameter of the _CRPROC system call to the parent process.

A process crash display, or stack dump, is also written to the SYS$ERROR file assigned to the terminated process.

Diagnostic name: **errdiv0**

67 (Floating point diagnostic) Number is too small.

The process was terminated because the result of the floating point operation was too small to be represented in the floating point format. In other words, the exponent that results from the normalization of a floating point operation does not fit in the exponent field of the floating point format.

If the process was spawned, this status is sent to the ccode parameter of the _CRPROC system call to the parent process.

A process crash display, or stack dump, is also written to the SYS$ERROR file assigned to the terminated process.

Diagnostic name: **errunderflo**

68 (Floating point diagnostic) Number is too large.

The process was terminated because the result of the floating point operation was too large to be represented in the floating point format. In other words, the exponent that results from the normalization of a floating point operation does not fit in the exponent field of the floating point format.

If the process was spawned, this status is sent to the ccode parameter of the _CRPROC system call to the parent process.

A process crash display, or stack dump, is also written to the SYS$ERROR file assigned to the terminated process.

Diagnostic name: **erroverflo**

69 (Floating point diagnostic) Illegal operation.

The process was terminated because it asked the floating point hardware to perform an undefined operation, e.g., the square root of a negative number, the log of a negative number, or raising a negative number to a power.

If the process was spawned, this status is sent to the ccode parameter of the _CRPROC system call to the parent process.

A process crash display, or stack dump, is also written to the SYS$ERROR file assigned to the terminated process.

Diagnostic name: **errillegalop**

70 (Floating point diagnostic) Denormalized operand.

The process was terminated because it performed a floating point operation that produced a denormalized result.

If the process was spawned, this status is sent to the ccode parameter of the _CRPROC system call to the parent process.

A process crash display, or stack dump, is also written to the SYS$ERROR file assigned to the terminated process.

Diagnostic name: **errdenormop**

71 This operation is not allowed on a SURROGATE process.

The operation attempted cannot be done to a surrogate process.

Diagnostic name: **errsurrogate**

72 A connect packet was received after the connection was made.

After a valid connection has been made by the network, a connect packet was received. (This error should not be received by a user process.)

Diagnostic name: **errdupconnect**

73 An SVC packet was received before the connect packet was received.

A connection must be established with a remote node via a connect packet before any SVC packets or special packets may be sent.

Diagnostic name: **errnoconnect**

74 The disconnect packet was not from the originator process.

Only the process that initiated a connection may terminate the connection via a disconnect packet.

Diagnostic name: **errwrngorigpid**

75 A packet was received for a local-execution-only SVC.

Certain SVCs may not be executed remotely, but only on the local machine. If a local-execution-only SVC packet type is received by a surrogate process, it will generate a reply packet with this error.

Diagnostic name: **errnotremotesvc**

76 The actual packet size is not the same as the size in the header.

The number of data bytes read by the network software for the current packet does not agree with the number of bytes that the packet header indicated were to follow.

Diagnostic name: **errbadpktsize**

77 The reply packet SVC is not the same as the request packet SVC.

The SVC number in the reply packet is not the SVC number that was sent to the remote system.

Diagnostic name: **errbadpktsvcno**

78 All available memory has been allocated on the remote system.

There is no memory available on the remote system to perform the requested operation.

Diagnostic name: **errnoremotemem**

79 The process is incompatible with the current operating system version.

Certain utility programs contain intimate knowledge about a particular version of WMCS. If a program that requires a specific version of WMCS to operate correctly detects that the version of WMCS is different than the required version, this error message is returned.

Diagnostic name: **errwrongos**

80 The specified name must not be null.

This diagnostic is returned to the process attempting to define a named shared memory segment with a null name. Null names are not allowed for named shared memory segments.

Diagnostic name: **errnamenull**

81 The specified name already exists.

This diagnostic is returned to the process attempting to define a named shared memory segment using a name that is already defined as a named shared memory segment.

Diagnostic name: **errnameexists**

82 The specified name does not exist.

This diagnostic is returned to the process attempting to share a named shared memory segment, and a segment with the specified name cannot be found.

Diagnostic name: **errnoname**

83 Process killed because of a queue restart request.

The Queue Manager was requested to restart a process, so it killed the currently running version of the process. This error is to enable special restart handling or cleanup of jobs run in a queue.

Diagnostic name: **errquerestart**

84-127 No error assigned.

Class Handler Diagnostic Messages

128 A request was not completed within the specified time.

The process specified a timeout value, i.e., a duration, for the completion of a request, and the duration expired before the requested operation was completed.

For example, a _READ request was not completed in the specified time.

Diagnostic name: **errtimeout**

129 A file's version number cannot be greater than 65535.

The process attempted to use the _CREATE or _CREATS system call to perform one or the other of these operations:

Create a file with a version number greater than 65535.

Create a file with a version number of zero in a directory that contains a file whose filename and extension match that of the file to be created, and whose version number is 65535.

Diagnostic name: **errinvvernum**

130 The specified devicename is syntactically incorrect.

At least one element in the devicename field of the specified file designation is disallowed.

Diagnostic name: **errinvdevnam**

131 The WMCS does not recognize the devicename. Is the device mounted?

The system device table, i.e., the list of mounted devices, does not contain the specified devicename.

Diagnostic name: **errundevnam**

132 The logical unit number does not correspond to an open file.

No open file has a logical unit number matching the unit number specified.

Diagnostic name: **errinvlfn**

133 The specified file could not be found.

The specified directory does not contain a file whose filename, file extension, etc., match the file designation specified.

Diagnostic name: **errfilnotfnd**

134 The specified version of the file already exists.

A file already exists whose file designation matches that of the file specified.

Diagnostic name: **errfilexists**

135 The specified file is read-locked.

The specified file is open and read-locked, i.e., it cannot be opened for read access.

Diagnostic name: **errreadlock**

136 The specified file is write-locked.

The specified file is open and write-locked, i.e., it cannot be opened for write access.

Diagnostic name: **errwritelock**

137 The specified queue does not have a default definition.

This diagnostic is returned when a process attempts to open (using the _open or the _create system call) a queue class device, and there is no default create process definition for the specified queue. (see the :printtype switch of the DSTAT command, and/or the _physop system call).

Diagnostic name: **errquenodef**

138 This edit mode requires that the record length be set to one.

The process opened, or created, a file with a record length other than one, and then (in an edit mode requiring that the file have a record length of one) attempted to read (_READ) or write to (_WRITE) the file.

Diagnostic name: **errinvreclen**

139 The specified file type is reserved for the WMCS.

The process attempted to create a kind of file whose file type is reserved for the WMCS.

Diagnostic name: **errinvfiletype**

140 The process tried to read past the logical end of a file.

The process attempted to use the _READ system call to read records that are beyond the end of the file, i.e., they do not exist.

Note that if some of the specified records exist they are read and no diagnostic message is reported, e.g., a process requests 10 records and the file contains 5; the five are read and no diagnostics message is reported.

Diagnostic name: **errreadleof**

141 The process does not have read-access to the specified file.

The process opened a file, did not request read access to that file, and then issued a system call that requires read access, e.g., _READ.

Diagnostic name: **errnoreadacc**

142 The process does not have write-access to the specified file.

The process opened a file, did not request write access, and then issued a system call that requires write access, e.g., _WRITE.

Diagnostic name: **errnowriteacc**

143 The process does not have Execute Privilege for the file.

The process cannot execute the specified file because the process does not have execute privilege for the file. Note that the owner of a file grants execute privilege to the various classes of users. The privileges granted to the class of users to which the process belongs is indicated in the file's protection mask.

This status also results when a process attempts to access a file, e.g., use the _OPEN system call, when the process does not have execute privilege for one or more of the directories along the directory path to the file. Thus, a process must have the appropriate privilege for the file, and execute privilege for all directories belonging to the file's designation.

Diagnostic name: **errnoexecpriv**

144 The process does not have Read Privilege for the file.

The process cannot open the file for read access because the process does not have read privilege for the file. Note that the owner of a file grants read privilege to the various classes of users. The privileges granted to the class of users to which the process belongs is indicated in the file's protection mask.

This status also results when a process attempts to access a file, e.g., use the _OPEN system call, when the process does not have read privilege for the directory or device containing the file.

Diagnostic name: **errnoreadpriv**

145 The process does not have Write Privilege for the file.

The process cannot open the file for write access because the process does not have write privilege for the file. Note that the owner of a file grants write privilege to the various classes of users. The privileges granted to the class of users to which the process belongs is indicated in the file's protection mask.

This status also results when a process attempts to access a file, e.g., use the _DELETE system call, when the process does not have write privilege for the directory or the device containing the file.

Diagnostic name: **errnowritepriv**

146 The process does not have Delete Privilege for the file.

The process does not have delete privilege for the specified file, and attempted one of the following operations:

Delete the file.

Open the file (with the delete-upon-closing mode-bit set).

Dismount the device.

Close the file (with the delete mode-bit set).

Note that the owner of a file grants delete privilege to the various classes of users. The privileges granted to the class of users to which the process belongs is indicated in the file's protection mask.

Diagnostic name: **errnodelpriv**

147 The specified filename is syntactically incorrect.

The filename field of the specified file designation is syntactically incorrect. For example, it may contain characters disallowed in the filename field, or it may contain too many characters, etc.

Diagnostic name: **errinvfnstr**

148 The specified directory is not a directory-type file.

While searching the directory path (the directories specified in the directory name filed of the specified file designation), the WMCS encountered a file that is not a directory.

Diagnostic name: **errinvdirfle**

149 The specified directory name is syntactically incorrect.

The directory name field of the specified file designation violates the syntax for that portion of the file designation. For example, it mnay contain characters disallowed in file designations.

Diagnostic name: **errinvdirstr**

150 The specified entry is already active.

Once an entry in the queue has begun execution, it cannot be modified, put on hold, or awakened. This diagnostic is returned to the process that attempts to perform one of those operations on an active queue entry. (see the _physop system call)

Diagnostic name: **errentryactive**

151 The WMCS cannot allocate more than 65535 sectors at a time.

The process asked the WMCS to allocate more than 65535 sectors (a very large _WRITE request).

Diagnostic name: **errinvsecreq**

152 The FCB (or the TFCB) does not correspond to its checksum.

Each File Control Block (FCB), or Tape File Control Block (TFCB) has a checksum associated with it, i.e., a sum of each of the values appearing in each field of the control block.

This status results when a process refers the WMCS to an FCB or TFCB and the WMCS finds that the checksum and the data in the control block do not correspond.

This helps the WMCS maintain the integrity of the file system on the device.

Diagnostic name: **errinvfcbcksum**

153 The specified file is open, has been marked for deletion.

The process asked the WMCS to delete an open file, e.g., a file that another process has open.

This status results as a WARNING, and when the process that has the file open closes the file, the file will be deleted.

Diagnostic name: **erropendel**

154 All available disk space has been allocated.

The process asked the WMCS to allocate space on a volume on which all available space has been allocated.

Diagnostic name: **errnospace**

155 The specified queue is closed.

This diagnostic is returned to a process which attempts to insert a new entry into a queue that is closed. A closed queue will not accept any new entries, but will continue to process all existing entries.

Diagnostic name: **errqueclosed**

156 The specified sector/block size is not supported on this device.

This status results when the sector size on a disk neither 512 nor 1024, or when the block size on a tape is greater than 4096.

Diagnostic name: **errinvsectorsz**

157 The specified entry was not found.

This diagnostic is returned to the process which attempts to access or modify an entry in a queue, and the specified entry does not exist.

Diagnostic name: **errentrynotfnd**

158 System files cannot be deleted.

The process attempted to delete one of the following system files:

/ROOTDIR/BITMAP.SYS /ROOTDIR/FCB.SYS /ROOTDIR/FCBBITMAP.SYS / ROOTDIR/ROOTDIR.DIR

Diagnostic name: **errdelfile**

159 System files cannot be renamed.

The process attempted to rename one of the following system files:

/ROOTDIR/BITMAP.SYS /ROOTDIR/FCB.SYS /ROOTDIR/FCBBITMAP.SYS / ROOTDIR/ROOTDIR.DIR

Diagnostic name: **errrenfile**

160 The device cannot be dismounted because files are still open on it.

The process attempted to dismount a device on which files are still open.

Diagnostic name: **errfilesopen**

161 The usage field in the file's FCB contains an unexpected value.

The usage field of a File Control Block (FCB) indicates the use begin made of that FCB, i.e., whether the FCB is available (unassigned), a primary FCB, or a continuation FCB.

This status results when the WMCS refers to an FCB and finds that the usage field does not indicate the usage the WMCS expected, e.g., the WMCS anticipates that the specified FCB is a primary FCB and finds that the FCB's usage field indicates that the FCB is either available or that it is a continuation FCB.

Diagnostic name: **errinvusagid**

162 The specified device was not properly configured.

This diagnostic is returned when a process attempts to mount a disk whose boot block does not contain a media description block. For disks mounted special, this diagnostic is returned as a warning. This diagnostic is not returned to the user process as an error.

Diagnostic name: **errdrnotconfig**

163 The request cannot cross machine boundaries.

The operation requested is not able to be done remotely. It must be performed on a local machine.

Diagnostic name: **errdiffmachine**

164 This device was improperly dismounted.

While mounting the specified device, the WMCS found that the device had not been properly dismounted. This can happen when the SHUTDOWN Command is not used to reboot the system.

Diagnostic name: **errimprdmnt**

165 The read request is invalid.

The process asked the WMCS to perform a task either too large or too small for the system to handle.

Diagnostic name: **errinvreadreq**

166 The request crosses a physical page boundary in memory.

Some file system operations, e.g., a fast read, require that the data buffer exist entirely within a single page of physical memory. This status results when such an operation is specified and the data buffer crosses a physical page boundary.

Diagnostic name: **errpagebndry**

167 A file cannot be renamed to another device.

The REN Command cannot be used to move a file to a device other than the device on which the file is located.

Diagnostic name: **errrendiffdev**

168 The boot block has changed since the device was mounted.

This status indicates that when the specified disk was dismounted, the data in the disk's boot block do not match the data read into memory when the disk was mounted.

Diagnostic name: **errdiffbtblk**

169 A sector(s) in the disk cache could not be written to the disk.

The WMCS has made several unsuccessful attempts to write a sector from the disk cache to the disk.

Diagnostic name: **errnowritesec**

170 Operator privilege is required in order to change a network window size.

The process must have operator privilege in order to change a network window size.

Diagnostic name: **errwindwszpriv**

171 The operation is inappropriate for physical devices in the network class.

The operation requested may only be performed by logical devices (virtual circuits), and not by physical network devices.

Diagnostic name: **errinvnetoper**

172 An error occurred in doing Huffman decompression on the network data.

Network data were compressed using the Huffman compression algorithms, and during the decompression phase an error was detected, invalidating the data that was transferred.

Diagnostic name: **errdecompress**

173 The operation is inappropriate for the device class.

Some operations can be performed only on devices in a particular class(es). For example, a process can rename a disk file, but not a TTY-class file or a tape file (the rename operation is not defined for TTY-class files and tape files).

Diagnostic name: **errinvcloper**

174 Directories do not exist on the specified device.

The target device for the specified operation does not contain directories, i.e., it is not a directory oriented device, and the specified operation requires that the target device have directories.

Diagnostic name: **errinvdirdev**

175 The specified device driver function code is disallowed.

The specified function code (_PHYSOP) is not recognized by the device driver.

Diagnostic name: **errunknowncmd**

176 The process buffer is too small for the specified operation.

The process used edit mode 2 or 4 to request a _READ, and the line that was read does not fit in the specified buffer for the user process.

Diagnostic name: **errbuftosmall**

177 The specified directory does not exist.

At least one of the directories in the directory name field of the file designation could not be found.

Diagnostic name: **errdirnotfnd**

178 The FCB.SEQ number for the file does not match the specified FCB.

The sequence number appearing on the specified File Control Block (FCB) does not correspond with one of the following (depending upon the manner in which the FCB was requested):

1. The FCB.SEQ number used to open the file, e.g., the process requests that the WMCS open the file by FCB.SEQ number.

2. The FCB.SEQ number in the directory file that contains information on the files in that directory does not match the FCB (in FCB.SYS) for the specified file. In other words, each directory file (a file with a .DIR file extension) contains a list of the files in that directory. Part of the information on that list is the FCB.SEQ number for each file in the directory. When a file designation is used to specify a file, the WMCS searches the list in the directory file to obtain the FCB.SEQ number for the specified file and then searches FCB.SYS to find the FCB

itself so that the WMCS can then find the data constituting the file on the disk. This status indicates that the FCB.SEQ number in the list in the directory file, and the FCB.SEQ number on the specified FCB in FCB.SYS, do not match.

Diagnostic name: **errinvseqnum**

179 The specified device is already mounted.

The process attempted to mount a device that is already mounted.

Diagnostic name: **errdevnamexs**

180 The WMCS does not recognize the specified device class.

The device class, specified in the request that a device be mounted, is not a recognized device class.

Diagnostic name: **errinvclass**

181 The specified volume has no valid boot block.

When the WMCS attempts to mount a disk, the WMCS reads sectors 0 and 32 on the disk (each of these sectors contains a boot block).

The WMCS determines the validity of a disk's boot block by comparing the checksum (that is part of the boot block) with the sum of the values that appear in each field of the boot block. This status is assigned when the checksum and the data in the boot block do not correspond.

When the WMCS attempts to mount a tape, the WMCS reads the first block from the tape and determines whether the data in that block correspond to the checksum. This status indicates that the data and the checksum do not correspond.

Diagnostic name: **errnobbfound**

182 The user's write request is too large to fit in the system buffers.

There is insufficient system buffer space to hold the user's write request.

Diagnostic name: **errwritetoobig**

183 The process requested more than 3964 bytes of dynamic memory.

One of the system tables is too large. This diagnostic is internal to the WMCS and is never assigned to a user process.

Diagnostic name: **errinvdmreq**

184 Not enough network buffers are available for a remote connection.

There are insufficient network buffers available to complete a remote connection.

Diagnostic name: **errnonetbufs**

185 The device class handler was not loaded when the system was booted.

The process asked the WMCS to mount a device, and the class handler for the class to which the device belongs, has not been loaded.

This status is also assigned when a process attempts to use the KSAM routines, and those routines have not been loaded.

Diagnostic name: **errnoclass**

186 The process tried to rename a directory as its own subdirectory.

The process attempted to rename a directory as a subdirectory of itself.

Diagnostic name: **errinvdirren**

187 The WMCS cannot extend the FCB file.

There is not enough free disk space to be able to extend the FCB.SYS file.

Diagnostic name: **errnoextfcbfil**

188 The specified device is already mounted, and has another name.

The process asked the WMCS to mount a device that is already mounted and has been assigned another name.

Diagnostic name: **errprevinit**

189 The WMCS does not recognize the specified edit mode.

The WMCS does not recognize the edit mode specified by the process.

Diagnostic name: **errinveditmd**

190 The specified device has already been mounted for synchronous use.

The process asked that a device be mounted as an asynchronous communication line, and the WMCS found that the device is already mounted as a synchronous communication device.

Diagnostic name: **errmntasync**

191 The specified device has already been mounted for asynchronous use.

The process asked that a device be mounted as a synchronous communication device, and the WMCS found that the device is already mounted as an asynchronous communication device.

Diagnostic name: **errmntsync**

192 The specified tape speed is not 12, 25, 30, 50, 90, 100, or 125 ips.

The tape speed parameter of the device status block is not one of the recognized speeds.

Diagnostic name: **errinvtpspeed**

193 The specified tape density is not 800, 1600, 3200, 6250, or 6400 bpi.

The only valid values for tape density are 800, 1600, 3200, 6250 or 6400 bpi. None of the values is valid for all types of tapes. Use the value that is appropriate for the type of tape being used.

Diagnostic name: **errinvtpdensity**

194 The network site ID on this machine is uninitialized.

The system variable containing the site ID must be initialized before any network operations can be performed.

Diagnostic name: **errnositeid**

195 The network nodename on this machine is uninitialized.

The system variable containing the nodename of the machine must be initialized before any network operations can be performed.

Diagnostic name: **errnonodename**

196 No error assigned.

197 The process tried to access a record (on a tape) out of sequence.

Random access is not possible on tape devices. Therefore, this status indicates that the process requested a record that is not the next record (relative to the position of the tape head).

Diagnostic name: **errbadpos**

198-199 No error assigned.

200 A directory file cannot have a version number greater than one.

The process asked the WMCS to perform an operation that would have resulted in a directory file (a file with a .DIR file extension) with a version number greater than 1. The WMCS requires that all directory files have a version number of 1.

Diagnostic name: **errdirinvver**

201 No error assigned.

202 The operation cannot be performed because a tape file is open.

The process made a request that would have resulted in more than one file being open on a tape simultaneously; this is disallowed because random file access is not supported on tapes.

Diagnostic name: **errfilopen**

203-205 No error assigned.

206 The specified skip or erase tape-function is undefined.

The process missed an unrecognized skip tape function (see _SKIP and _PHYSOP/_PHYSIO).

Diagnostic name: **errinvskpcmd**

207-209 No error assigned.

210 The specified directory cannot be deleted; it contains files.

The DEL Program assigns this status when a process attempts to delete a directory that contains files.

Diagnostic name: **errdelnempdir**

211-214 No error assigned.

215 The specified device driver is unsuitable for this device class.

An attempt was made to mount a device and assign it a device driver that is incompatible with the class to which the device belongs, e.g., mounting a disk drive and assigning it a TTY-class device driver.

Diagnostic name: **errinvdrvclass**

216 The specified file does not contain a device driver.

The file designated as the device driver does not contain a device driver, or is not a system file (the WMCS checks several fields in a file to determine whether the file contains a device driver).

Diagnostic name: **errinvdriver**

217 The value specified for a KSAM key type is undefined.

KSAM allows for multiples of signed and unsigned bytes, words, and long words as key types. This status is assigned when the specified type is not one of the types allowed (see _KCREAT).

Diagnostic name: **errbadkeytype**

218-220 No error assigned.

221 One or more of the KSAM keys is not contained in the record.

When KSAM reads a record, it checks to make sure that all key fields within that record contain the values identified in the key file. This status is assigned when one or more of those key fields contains incorrect values.

Diagnostic name: **errkeynotinrec**

222 The KSAM key definition table is larger than 3500 bytes.

The KSAM key definition table (used in _KCREAT) cannot be larger than 3500 bytes.

Diagnostic name: **errkeytablelen**

223 The specified file is not a KSAM data file.

The file, specified in a call to _KOPEN, is not a KSAM data file. Note that a file's type is determined when the file is created, and that a file created with _KCREAT is of the correct type.

Diagnostic name: **errnodatafile**

224 The specified file is not a KSAM key file.

The file, specified in a call to _KOPEN, is not a KSAM key file. Note that a file's type is determined when the file is created, and that a file created with _KCREAT is of the correct type.

Diagnostic name: **errnokeyfile**

225 The specified number of keys is less than or equal to zero.

The first word in the ktable parameter of the _KCREAT system call specifies how many keys are defined in this KSAM file. This status indicates that the value assigned to the ktable parameter is either zero or a negative number.

Diagnostic name: **errnumofkeys**

226 The specified number of segments is less than or equal to zero.

Each key in a KSAM file can consist of from 1 to 15 segments. This status indicates that, during the creation of a KSAM file) by means of _KCREAT), the number of segments (for one or more keys) is less than or equal to zero.

Diagnostic name: **errnumofsegs**

227 The record size is less than 4 bytes or greater than 65534 bytes.

KSAM allows record sizes of from 4 - 64434 bytes. This status indicates that the record size, specified in the reclen parameter of the _KCREAT system call, is not in the foregoing range.

Diagnostic name: **errrecsz**

228 A KSAM key for a word or longword key type is not word aligned.

All signed or unsigned word or long word keys in a KSAM file must be word aligned within a record. This status indicates one or more of the word or long word keys specified in a call to _KCREAT is not properly aligned.

Diagnostic name: **errsegalign**

229 The specified key length is not a multiple of the key-type length.

The _KCREAT system call assigns this status to indicate that the length of a word key-field is not a multiple of two, or that the length of a longword key-field is not a multiple of four.

Diagnostic name: **errseglen**

230 Key number is greater than or equal to the number of defined keys.

When a process creates a KSAM file, the process specifies the number of keys that each record will contain. This status indicates that, subsequent to the file's creation, a process refers to a key whose number is greater than or equal to the number of keys in each record in the file, e.g., each record in a file contains five keys (numbered 0 - 4) and a process attempts to refer to a key whose number is 5 or more (the 5 is equal to the total number of keys in the file, but does not correspond to a key number).

Diagnostic name: **errkeynotfnd**

231 This operation requires that the current key be defined.

This status indicates that the required definition was not given.

Diagnostic name: **errkeynotdef**

232 Duplicate key was attempted in a field disallowing duplicate keys.

One attribute of each key field in a KSAM file is whether or not duplicate values are allowed for the key. This status indicates that, for a key disallowing duplicate values, a process attempted to write to that key a value that is not unique (within the file) to that key.

Diagnostic name: **errnodupkey**

233 (WMCS error) A discrepancy in the KSAM code has been detected.

This status indicates that an error occurred within KSAM. The process is not directly responsible for the error.

Diagnostic name: **errksamint**

234 The specified record cannot be locked without causing a deadlock.

The WMCS cannot lock the specified records without causing a deadlock, i.e., a situation wherein each of two processes has a record the other process wants to lock of access.

Diagnostic name: **errdeadlock**

235 The specified record(s) are locked by another process.

The process attempted to access a record locked by another process.

Diagnostic name: **errreclocked**

236 This operation requires that the current record be defined.

Some KSAM operations require that the current record be defined. This status indicates that the required definition was not provided.

Diagnostic name: **errrecnotdef**

237 The process attempted to unlock a record(s) it had not locked.

The process attempted to unlock on or more records that are not locked by the calling process.

Diagnostic name: **errrlocknotl**

238 (WMCS error) A discrepancy in the KPFD linkage has been detected.

An error occurred within KSAM, an error for which the process is not directly responsible.

Diagnostic name: **errkpfdlink**

239 The key does not point to the beginning of an active data record.

A KSAM key pointer points to a record that has been deleted.

Diagnostic name: **errkeylink**

240 (WMCS error) A KSAM data-structure linkage error has been detected.

An error occurred within KSAM, an error for which the process is not directly responsible.

Diagnostic name: **errnoprocess**

241 An exact match for the specified key value was not found.

The specified key value (in a call to _KFIND) was not found in the file.

Diagnostic name: **errsrchnotfnd**

242 (WMCS error) A KSAM buffer flushing error was detected.

An error occurred within KSAM, an error for which the process is not directly responsible.

Diagnostic name: **errksamflush**

243 Key- and data-file values for a record's key do not agree.

The value of key field in the data record, and the value of the key in the key file do not match.

Diagnostic name: **errbadkeycmp**

244 (WMCS error) An error was detected during deletion of a leaf key.

An error occurred within KSAM, an error for which the process is not directly responsible.

Diagnostic name: **errdellink**

245 No error assigned.

246 One of the parameters specifies an unrecognized option.

Several KSAM system calls have an option parameter. This status indicates that the value assigned to an option parameter does not correspond with a recognized function.

Diagnostic name: **errknooption**

247 (WMCS error) A discrepancy in the KFCB linkage has been detected.

An error occurred within KSAM, an error for which the process is not directly responsible.

Diagnostic name: **errkfcblink**

248-253 No error assigned.

254 (WMCS error) A discrepancy in the Record Locking code has been detected.

An internal recording-locking error occurred. The process is not directly responsible for the error.

Diagnostic name: **errrlockint**

255 [CTRL] c terminated the process.

The process was terminated because it was the last process to access a serial port on which a [CTRL] c was received.

If the process was spawned, this status is sent to the ccode parameter of the _CRPROC system call to the parent process.

Diagnostic name: **errcontccode**

Device Driver Diagnostic Messages

256 The sector header on the disk cannot be read.

The device was unable to read the header for a disk sector.

Diagnostic name: **errbadheader**

257 The seek or rewind took too long.

This error is returned when the device reports an error during a seek or a rewind operation.

Diagnostic name: **errseekto**

258 The device cannot perform a seek.

A device check occurred, and a disk seek cannot be performed.

Diagnostic name: **errseekflt**

259 A seek did not reach the proper cylinder.

This error is reported when a disk device fails during a search operation and does not reach the specified cylinder.

Diagnostic name: **errseekwrng**

260 The data in a sector header do not match the CRC or ECC.

This error is returned when a the data in a sector header on a disk does not match the check word (CRC or ECC) that is used to verify that data.

Diagnostic name: **errheadcrc**

261 The device cannot perform a recalibration.

This error is returned when a device fault has occurred during a recalibration operation.

Diagnostic name: **errreorgflt**

262 A recalibration took too long.

This error is returned when a device fault has occurred during a recalibration operation.

Diagnostic name: **errreorgto**

263 The specified device is either off-line, or is not responding.

The device driver attempted to access the specified device and that device did not respond.

Diagnostic name: **erroffline**

264 A device error occurred during a write to the volume write fault).

This error is returned when a write fault occurs on a device.

Diagnostic name: **errwriteflt**

265 The specified device is format-protected, and cannot be formatted.

The device driver's attempt to format the device failed because the device is format-protected (this is not the same as write protection).

Diagnostic name: **errformatprot**

266 A device error occurred during a read from the volume (read fault).

This error is returned when a read fault occurs on a device.

Diagnostic name: **errreaddataflt**

267 The data on the volume do not match the CRC, ECC, or checksum.

Each sector or block on a disk or tape contains a check word (CRC, ECC or checksum) used to guarantee the validity of the data. If a sector or block is read, and the contents of the sector or block do not match the check word, this error is returned.

Diagnostic name: **errdatacrcerr**

268 The specified sector was not found on the current track.

The controller could not find the specified sector.

Diagnostic name: **errseclocate**

269 The specified device is write-protected.

The process attempted to write to a write-protected device.

This status is also assigned as a warning when a write-protected device is mounted.

Diagnostic name: **errdevwrtprot**

270 The specified sector number is too large.

If the specified sector existed, it would be beyond the end of the volume.

Diagnostic name: **errsectolarge**

271 The device received a command the device did not recognize.

This error is returned when the device controller is sent a command that it did not recognize.

Diagnostic name: **errbadcmd**

272 The device is not functioning properly (device check).

A general device check has occurred; the cause is unknown.

Diagnostic name: **errdevcheck**

273 Data were lost; the driver could not read them quickly enough.

This error is returned when a data overrun error occurs.

Diagnostic name: **errportovrun**

274 Sector headers could not be found. Is the volume formatted?

This error is returned when the disk controller did not find sector headers on the volume.

Diagnostic name: **errnotformat**

275 The specified device did not respond in the allotted time.

This error is returned when the device did not respond in the expected time. A command was sent to the device, and was not completed in a reasonable amount of time.

Diagnostic name: **errdevtimeout**

276 A read-after-write shows a discrepancy in the data.

A verification of a read-after-write was performed and the data read do not match the data written.

Diagnostic name: **errrawfailure**

277 The tape is positioned at the end of the data on the tape.

The process attempted to read data that, if they existed, would be located beyond the end of the data that do exist on the tape.

Diagnostic name: **errlogicaleot**

278 The tape is positioned at the physical end of the volume.

The process attempted to write data beyond the physical end of the tape.

Diagnostic name: **errphysicaleot**

279 The tape is positioned at the physical beginning of the volume.

The process attempted to position the tape beyond the physical beginning of the tape. For example, the process may have attempted to skip five files toward the beginning of the tape when only four files were located between the tape's position and the beginning of the tape.,

Diagnostic name: **errphysicalbot**

280 The size of the block read from the tape is larger than requested.

The process asked that a block be read from a tape, and the number of bytes in the block exceeds the number of bytes requested by the process.

Diagnostic name: **errtpoverflow**

281 A parity error was detected in the data on the tape.

This error is returned when a parity error was detected in the data being read from the tape.

Diagnostic name: **errtpparityl**

282 The device wasn't granted access to the bus in the allotted time.

DMA-type devices request access to the system bus. This status indicates that access was not granted within a reasonable time.

Diagnostic name: **errbusto**

283 A parity error was detected in the tape controller.

A parity error was detected in the dual port memory on the tape controller board. Note that this is not a parity error in the data on the tape.

Diagnostic name: **errtpparity2**

284 The specified device was improperly set up.

The device driver received a function request for a device that has not been initialized, e.g., a device driver receives a read function request before receiving a startup request.

Diagnostic name: **errnotinit**

285 The device being read was written at a different density.

The density with which the data were written on the tape does not match the data density expected by the controller.

Diagnostic name: **errinvdensity**

286 Connection to a remote computer has not been established.

This diagnostic is returned to a process which attempts to read or write to a modem control port (e.g. X.25) which has no connection established to a remote port.

Diagnostic name: **errnocallestb**

287 Connection to a remote computer has already been established.

This diagnostic is returned to a process which attempts to establish a connection on a modem control port which already has a connection.

Diagnostic name: **errcallestb**

288 The specified device was improperly set up.

This diagnostic is returned to a process which attempts to mount a port using hardware which is being used by a different driver.

Diagnostic name: **errdevinuse**

289 A deadlock error has been detected on the device.

This diagnostic is returned when a process attempts to write to a port to which no writes are possible because there are no more output buffers available, and the input buffers are full. That is, "nobody can write until somebody reads".

Diagnostic name: **erriodeadlock**

290 The X.25 channel has been reset by the network, possible data loss.

This diagnostic is returned to a process attempting to read or write to an X.25 channel which has been reset by the network. This is a warning that data may have been lost.

Diagnostic name: **errlinereset**

291 The dial request failed.

This diagnostic is returned to a process attempting to do a dial _phsop system call when the dial fails. The process can use _getdst to obtain the device status block which contains the reason code why the dial failed.

Diagnostic name: **errdialfailed**

292 The state of the BSC line disallows the specified function.

The process specified a function that, given the state of the BSC line, cannot be performed.

Diagnostic name: **errsyinvlnst**

293 The modem is not ready for communication.

The modem, attached to a synchronous line, is not in a ready state.

Diagnostic name: **errmodemnotrdy**

294 A bid was received in response to a BSC bid.

The process issued a bid request to the BSC driver and received a different bid on the response line.

Diagnostic name: **errsylnbidrcvd**

295 A NAK was received in response to a BSC bid, poll, or select.

A NAK was received on the synchronous line in response to a bid, poll, or select.

Diagnostic name: **errsynakrcvd**

296 An EOT was received on a BSC line.

An unexpected EOT was received on the synchronous line.

Diagnostic name: **errsyeotrcvd**

297 An RVI was received in response to a write on a BSC line.

The process sent a write on a BSC line and received an RVI in response.

Diagnostic name: **errsyrvircvd**

298 A disconnect sequence was received on a BSC line.

Diagnostic name: **errsydiscrcvd**

299 None of the devices, on a BSC polling list, responded.

The BSC driver polled each device specified on the polling list and none responded.

Diagnostic name: **errsypollstemp**

300 _BSCLOG's Transfer Log was invoked before Begin Logging.

The process asked the BSC driver to transfer the contents of the log buffer, but did not first use the the Begin Logging Function to initiate the log buffer.

Diagnostic name: **errsynodblog**

301 The driver transferred unverified data to the process.

The process requested data and the BSC driver sent unverified data to the process. Data are unverified when the driver has not compared the data to the check word.

Diagnostic name: **errsynoverread**

302 A conversational reply was recevied in response to a BSC write.

The BSC driver received an unexpected conversational reply, e.g., bid, in response to a write.

Diagnostic name: **errsyconvreply**

303 The last (no-verify) read did not succeed.

The BSC driver transferred unverified data to a process and then found the data to be erroneous.

Diagnostic name: **errsyprevread**

304 The last (no-wait) write did not succeed.

The calling process did not wait for the requested transmission from the BSC driver.

Diagnostic name: **errsyprevwrite**

305 Only part of the driver's transmission block was transferred.

The BSC driver assigns this status as a WARNING.

Diagnostic name: **errsypartread**

306 The BSC transmission block is larger than the driver's buffer.

The transmission block is too large for the BSC driver.

Diagnostic name: **errsybufovflw**

307 A WAK was received in response to a BSC bid, poll, or select.

The BSC driver received an unexpected WAK.

Diagnostic name: **errsywackrcvd**

308 The size of the device driver does not match its expected size.

At the front of each device driver is a length. When a device is mounted the device driver for that device is loaded into the system. If the OS gets an error reading the driver, this error is returned.

Diagnostic name: **errcantreaddsr**

309 A BSC line is no longer synchronized.

The BSC driver assigns this status when a synchronous line drops out of synchronization.

Diagnostic name: **errsydropsync**

310 _BSCPOL's parameter block is incorrect.

A syntactical error was discovered in the parameter block.

Diagnostic name: **errsyinvprmblk**

311 A value in at least one field of the devicename is disallowed.

The system's hardware does not include anything corresponding to the value appearing in the drive number field of the specified devicename, e.g., an attempt to mount _TT99 when a 99th serial port does not exist.

Diagnostic name: **errinvdrvnum**

312 The PC board for the specified device is not installed.

The driver received a bus error while attempting to access the specified device.

Diagnostic name: **errnohardware**

313 The hangup cannot take place, files are still open on the device.

No longer used.

Diagnostic name: **errdiscfilsopn**

314 The device driver does not contain the code to be downloaded.

This diagnostic is returned when a process attempts to mount a port that requires download code, and the download code is not present in the driver file.

Diagnostic name: **errnodwnldcode**

315 The WICOM board has been restarted and all calls were cleared.

This diagnostic is returned to each process that attempts to access a port without first closing and reopening the port following a hardware failure on the Wicom board which clears all calls.

Diagnostic name: **errboardreset**

316 The contents of the dial buffer are missing or invalid.

This diagnostic is returned to the process making a dial _physop call if the length of the dial buffer is zero or if the contents of the buffer are not appropriate for the line type.

Diagnostic name: **errinvdialbuf**

317 The driver cannot use this version of the drive type table.

There is a drive type table stored in the boot block of each disk device. One field of the drive type table is a version number. The disk device drivers are keyed to specific versions of this drive type table. If the driver reads the boot block and discovers that the version number is not a version that the driver recognizes, this diagnostic is returned.

Diagnostic name: **errbaddrtptbl**

318 The SCSI port is already busy on select.

When the hardware device port was sent a select command, it was already busy.

Diagnostic name: **errscsibusy**

319 No SCSI request after select.

After issuing a select command to the hardware device, it did not respond with a request.

Diagnostic name: **errnoscsireq**

320 The SCSI controller is in the wrong phase.

The device controller is in the wrong phase to receive the type of command received.

Diagnostic name: **errscsiphase**

321 Error detected while requesting SCSI error status.

An error occurred during the process of requesting extended error status from the device.

Diagnostic name: **errreqscsistat**

322 SCSI port hardware error.

An error occurred in the hardware port of the device.

Diagnostic name: **errscsiporthd**

323 SCSI error detected with no error status.

An error was detected in the device controller, but no error status followed to define the error.

Diagnostic name: **errscsinostat**

324 No index signal.

No index signal was detected on the device to synchronize with.

Diagnostic name: **errscsinostat**

325 No track zero.

The disk controller was unable to detect where track zero should begin.

Diagnostic name: **errnotrkzero**

326 Multiple Winchester drives selected.

More than one Winchester driver has been selected at the same time.

Diagnostic name: **errscsimanydev**

327-383 No error assigned.

Utility Diagnostic Messages

384 A character in the specified accept sequence is disallowed.

The specified accept sequence (a character string beginning with a backslash, _\) is syntactically incorrect.

Diagnostic name: **errinvacceptsq**

385 No more file designations match the specified wildcard pattern.

The list of files, matching the specified wildcard pattern, has been exhausted.

Diagnostic name: **errnomorefiles**

386 No file designations match the specified wildcard pattern.

The specified wildcard pattern does not match any files in the directories searched.

Diagnostic name: **errnofilesfnd**

387 One or more parameter value(s) is longer than 255 characters.

Neither the command line parser nor the WMCS utilities can accommodate a parameter containing more than 255 characters.

Diagnostic name: **errclparamlong**

388 There are more than nine parameters to the parameter file.

No more than nine parameters can be assigned to a parameter file. Note that this restriction does not pertain to the number of parameters to the WMCS utilities.

Diagnostic name: **errtomanyprm**

389 Too many parameter values were specified.

The number of values specified for the required and optional parameters to a particular WMCS utility does not correspond to the number of parameters expected. This status indicates that too many parameter values were specified for a particular CIP command (on the command line, in a command file, or in a parameter file).

Diagnostic name: **errtomanyreq**

390 The specified switch is not recognized.

At least one of the specified switches is not recognized, e.g., the spelling or abbreviation is incorrect.

Diagnostic name: **errclunknownsw**

391 An unacceptable value was specified for this switch.

The value assigned to a value switch is either unacceptable or unrecognizable.

Diagnostic name: **errclinvswval**

392 The abbreviation of the specified switch is ambiguous.

The command line parser could not determine which of two or more switches was intended, e.g., if a utility has :ABCD and :ABCE as switches and you type :ABC on the CIP command line, the command line parser cannot determine which of the two switches is intended.

Diagnostic name: **errclnonunqsw**

393 This switch was specified twice; the first occurrence is used.

The same switch is specified more than once on a single command line, or in a single parameter file.

Diagnostic name: **errclmultsw**

394 A required parameter was not specified in the parameter file.

There are fewer parameters in the parameter file than there are required parameters to the command.

Diagnostic name: **errclmissprm**

395 An error occurred when the process attempted to create SYS$ERROR.

The process received an error while attempting to create SYS$ERROR.

Diagnostic name: **erropensyserr**

396 The operation cannot be performed on a file of this type.

Some WMCS utilities operate on certain kinds of files, e.g., image files, directory files, data files, etc. The specified file is not of the kind required by the utility.

Diagnostic name: **errinvftype**

397 The specified directory cannot be deleted; it contains files.

The specified file is a directory file that has files in it; only empty directories can be deleted.

Diagnostic name: **errnodeldir**

398 Multiple command lines are not allowed for this operation.

This status results when the command line character string or the parameter file contains multiple command lines, and the utility does not support multiple command lines.

Diagnostic name: **errmultcmdln**

399 No such command is defined for this operation.

Diagnostic name: **errinvcmd**

400 The specified switch is not of the expected type.

Either a value switch was specified and no value was assigned, or a boolean switch has a value assigned to it.

Diagnostic name: **errinvswtype**

401 The syntax of the specified date and time is incorrect.

Either the date, time, or both the date and the time were incorrectly specified, e.g., a keyword is unrecognizable, an unacceptable delimiter was used, etc.

Diagnostic name: **errinvdate**

402 Conflicting function switches were specified.

For utilities that have function switches, at most one of the switches may be specified in any given command. This diagnostic is reported when more than one function switch was specified.

Diagnostic name: **errinvsetsw**

403 There is not enough space on the volume to accommodate the request.

This diagnostic is reported by the dinit utility when it cannot write the file system to the volume with the parameters specified. For instance, if the operator specifies a value for the :fcbsize= switch which will not fit on the volume this diagnostic is reported.

Diagnostic name: **errinsufspace**

404 The :edit= switch syntax did not match str1:str2,str3:str4, etc.

This diagnostic is reported when the value of the :edit= switch is syntactically incorrect.

Diagnostic name: **errinvedit**

405 The :protection= switch syntax did not match S:DWRE,P:DWRE, etc.

This diagnostic is reported when the value of the :protection= switch is syntactically incorrect.

Diagnostic name: **errinvprotect**

406 The UIC syntax did not match [xxxx,xxxx].

This diagnostic is reported when the specified UIC is syntactically incorrect.

Diagnostic name: **errinvuic**

407 The range specification syntax did not match n or n-m or n- .

This diagnostic is reported when the syntax of a range of numbers is syntactically incorrect.

Diagnostic name: **errinvrange**

408 The data received do not match the original data transmitted.

The USSCOPY utility reports this diagnostic when it cannot successfully send or receive a packet after the specified number of retries.

Diagnostic name: **errtransmit**

409 The remote station's response does not relate to the transmitted data.

The USSCOPY utility reports this diagnostic when it receives a packet that does not correspond to the current context. The two USSCOPY processes are not synchronized.

Diagnostic name: **errsynchronize**

410 The remote station did not respond in a reasonable amount of time.

The USSCOPY utility will report this diagnostic when it does not receive a packet from the opposite station within the specified timeout. It waits for a packet with the specified timeout for the specified number of retries before this error is reported.

Diagnostic name: **errnoresponse**

411 The specified switch is disallowed in this context.

Some utilities have combinations of switches and parameters that are mutually exclusive. When an invalid combination of switches is specified, this diagnostic is reported.

Diagnostic name: **errinapprsw**

412 The specified username does not exist.

The specified username was not found in the SHORTUAF.DAT file. Use the WHO command to obtain a list of authorized usernames.

Diagnostic name: **errinvuser**

413 Fixed-length records cannot be converted to a different length.

This diagnostic is displayed by the TCOPY utility when there is a numeric value assigned to both the :srceform= switch and the :destform= switch, and the two values are not equal. TCOPY is not capable of "reblocking" the file. That is, you cannot use TCOPY to convert fixed-length records of one record size to fixed-length records of another record size.

Diagnostic name: **errinvconvert**

414 The record size must divide evenly into the block size.

The block size specified in the TCOPY utility must be an even multiple of the specified record size. If it is not, this diagnostic appears. For instance, if the record size is 132, the block size can be any multiple of that record size (132, 264, 396, 528, 660, ...).

Diagnostic name: **errinvrecsize**

415 The :privilege= switch syntax did not match SYSTEM,SETPRIV, etc.

The :privilege= switch may be followed by any combination of the following keywords (or unique abbreviations thereof) separated by commas. If any of these syntax rules are violated, this error is reported.

setpriv	nosetpriv	system	nosystem
readphys	noreadphys	writephys	nowritephys
setprior	nosetprior	chngsuper	nochngsuper
bypass	nobypass	operator	nooperator
altuic	noaltuic	world	noworld
group	nogroup	all	none
network	nonetwork		

Diagnostic name: **errinvpriv**

416 A parameter contains a wildcard character where they are not allowed.

A wildcard character was found in a parameter where wildcards are disallowed. Possible wildcards are asterisk, *, equal sign, =, square brackets, [], and parentheses, ().

Diagnostic name: **errinvwild**

417 The specified pipe command is invalid.

This diagnostic is displayed when the command line contains an invalid usage of the pipe character, e.g., > dir |

Diagnostic name: **errinvpipecmd**

418 The syntax of the specified pattern is incorrect.

The pattern specified as a parameter to the SCAN utility is syntactically incorrect.

Diagnostic name: **errinvpattern**

419 There is not enough space in the file to accommodate the request.

This diagnostic is reported when the user requests that a file be inserted or replaced within an existing archive file and if either the specified archive file cannot contain any more files (the archive directory is full) or there is not enough space within the archive file to accommodate the new file.

Diagnostic name: **errarchfull**

420 The values in the setup file are invalid or out of range.

This diagnostic is reported when a process reads a setup file and discovers that the contents are inappropriate.

Diagnostic name: **errsetupfileinv**

421 The specified drive type was not found in the drive type file.

The utility searched the DISK.CFG file for a record corresponding to the specified drive type, and did not find one.

Diagnostic name: **errdrtypnotfnd**

422 The specified device had no drive type listed for it.

The utility checked the record in the DEVCONFIG file for the specified device. The record was found, but the drive type field was empty (unspecified).

Diagnostic name: **errnodrtype**

423 The process was terminated with an error.

This abort status code is returned by a process to its parent process to signal that the child was not successful. For example, the COMPILE utility creates a child process to perform the syntax check. If errors are found by the child process in the syntax check, then the child process returns this error to the COMPILE utility.

Diagnostic name: **errprcsfailure**

424 The lower bound of the range is greater than the upper bound.

In specifying a range, the lower bound may not be greater than the upper bound.

Diagnostic name: **errbadrange**

425 The specified range falls outside the allowable range.

The user-specified range is not within the allowable range for this operation.

Diagnostic name: **erroutrange**

426 The keys are not consecutive; a :keyN= switch has been skipped.

When specifying multiple keys for sorting, the key numbers must be consecutive.

Diagnostic name: **errskipkeysw**

427 The FIELD= modifier cannot be used with binary type fields.

The FIELD= key modifier for sorting cannot be used with binary type fields.

Diagnostic name: **errsrtfldbin**

428 The IGNORELEADING= modifier cannot be used with binary type fields.

The IGNORELEADING= key modifier for sorting cannot be used with binary type fields.

Diagnostic name: **errsrtignbin**

429 The STARTAT= modifier must be on a byte boundary.

The STARTAT= key modifier for sorting must begin on a byte boundary.

Diagnostic name: **errsrtsatbyte**

430 The ENDAT= modifier must be on a byte boundary.

The ENDAT= key modifier for sorting must begin on a byte boundary, rather than within a byte (bit field).

Diagnostic name: **errsrteatbyte**

431 The OFFSET= modifier must be on a byte boundary.

The OFFSET= key modifier for sorting must begin on a byte boundary, rather than within a byte (bit field).

Diagnostic name: **errsrtoffbyte**

432 The sort key requires the field to start on a byte boundary.

The specified sort key requires the field to begin on a byte boundary, rather than within a byte (bit field).

Diagnostic name: **errsrtposbyte**

433 The sort key requires the length to be a multiple of bytes.

The specified sort key requires the field to be a multiple of bytes, rather than bits (bit field).

Diagnostic name: **errsrtlenbyte**

434 The sum of STARTAT= + OFFSET= modifiers must be positive.

The sum of the STARTAT= and OFFSET= key modifiers for sorting must be positive.

Diagnostic name: **errsrtsatoffng**

435 The STARTAT= modifier must be a positive integer.

The value specified for the STARTAT= key modifier for sorting must be a positive integer.

Diagnostic name: **errsrtsatpos**

436 The OFFSET= modifier must be a positive integer.

The value specified for the OFFSET= key modifier for sorting must be a positive integer.

Diagnostic name: **errsrtoffpos**

437 The ENDAT= modifier must be a positive integer.

The value specified for the ENDAT= key modifier for sorting must be a positive integer.

Diagnostic name: **errsrteatpos**

438 The LENGTH= modifier must be a positive integer.

The value specified for the LENGTH= key modifier for sorting must be a positive integer.

Diagnostic name: **errsrtlenpos**

439 The FIELD= modifier must be a positive integer.

The value specified for the FIELD= key modifier for sorting must be a positive integer.

Diagnostic name: **errsrtfldpos**

440 The :RECORDLEN= switch must be a positive integer.

The value specified for the :RECORDLEN= switch for sorting must be a positive integer.

Diagnostic name: **errsrtrecpos**

441 The :MEMORY= switch must be a positive integer.

The value specified for the :MEMORY= switch for sorting must be a positive integer.

Diagnostic name: **errsrtmempos**

442 The :MAXRECORDLEN= switch must be a positive integer.

The value specified for the :MAXRECORDLEN= switch for sorting must be a positive integer.

Diagnostic name: **errsrtmaxpos**

443 A field must be at least one bit wide (STARTAT= + LENGTH= > ENDAT=).

The field for sorting defined by the STARTAT=, LENGTH=, and ENDAT= key modifiers must be at least one bit wide.

Diagnostic name: **errsrtsatoffgt**

444 The field is not big enough for the given length (LENGTH= > ENDAT=).

The field for sorting defined by the STARTAT=, LENGTH=, and ENDAT= key modifiers is not big enough.

Diagnostic name: **errsrtlengt**

445 A length must be specified.

A length must be specified for the sort field. No defaults are allowed.

Diagnostic name: **errsrtlen0**

446 The key length must be <= 32 bits for BINARY or BIT.

The length of the key for BIT or BINARY sort fields must be less than or equal to 32.

Diagnostic name: **errsrtlen32**

447 The key length must be <= 64 bits for FLOATINGPOINT or REAL.

The length of the key for FLOATINGPOINT or REAL sort fields must be less than or equal to 64.

Diagnostic name: **errsrtlen64**

448 FLOATINGPOINT or REAL must have a length of 32 or 64 bits.

The length of the key for FLOATINGPOINT or REAL sort fields must be 32 or 64

Diagnostic name: **errsrtlen3264**

449 A text file cannot have a record length greater than one (1) byte.

A text file for sorting must have a record length of one byte.

Diagnostic name: **errsrttxtlen**

450 The delimiter= modifier is required when field= is specified.

If the field= key modifier is specified, then the delimiter= key modifier must also be specified.

Diagnostic name: **errsrtdelreq**

451 The pattern is too complex.

The sort pattern specified is too complex for the SORT program.

Diagnostic name: **errscnpatcmplx**

452 The extension is not recognized.

The extension of the file is not recognized by the program. The program assumes certain file formats and/or contents based on the file extension.

Diagnostic name: **errunknownext**

453 The :attribute= switch did not match SWAPPABLE,DESENCRYPT,...

Valid values for the :attribute= switch are: SWAPPABLE,PREZEROMEM, POSTZEROMEM,DESENCRYPT,FASTENCRYPT,WATCHDOG,USER1,USER2,USER3, and USER4. No other values may be given for this switch.

Diagnostic name: **errinvattr**

454 The username/password cannot be validated.

To successfully access the public/private key file, a valid username and password must be supplied.

Diagnostic name: **errinvuserpass**

455 The data checksum is not valid.

The checksum calculated for the data does not agree with the checksum that was stored with the data.

Diagnostic name: **errinvdatacksum**

456 Error(s) occurred during assembly.

One or more errors were detected in the source files processed by WIMAC.

Diagnostic name: **errasmerr**

457 The terminal type is unsupported by this utility.

The terminal type that is assigned to this terminal is not supported by this utility.

Diagnostic name: **errunsupportterm**

458 The data read is inconsistent, invalid, or has missing bytes.

The data that was read by the utility had either inconsistent or invalid values in it, or it was detected to be missing some required data.

Diagnostic name: **errmissinvdata**

459-4095 No error assigned.

Appendix C

Remote System Calls

The following system calls are known as remote system calls because they can be executed over the network:

_alloc	_andevnt	_assign	_chdir	_clone
_close	_clrevnt	_connect	_create	_creats
_crprcs	_crproc	_dealloc	_defdprt	_defduic
_deinst	_delete	_dismnt	_duplun	_errno
_exproc	_flush	_gassign	_gengy	_getalc
_getattr	_getdnam	_getdprt	_getdst	_getduic
_getevnt	_getfcb	_getfid	_getfnam	_getfprt
_getfre	_getfrsz	_getfuic	_getglb	_getinst
_getlog	_getmlst	_getpcb	_getpid	_getpnam
_getpos	_getpri	_getprv	_getrel	_getrtr
_gettic	_gettim	_gettmsl	_getuic	_giodst
_gmail	_hibern	_install	_kclose	_kdelet
_kfind	_kflush	_kinfo	_kmovfb	_kopen
_kread	_kunlck	_kupdat	_kwrite	_lock
_mount	_mulcrps	_open	_orevnt	_origprv
_physio	_physop	_prclst	_prirat	_rdpmem
_read	_rename	_setattr	_setdprt	_setdst
_setduic	_setevnt	_setfcb	_setfid	_setfprt
_setfrsz	_setfuic	_setpnam	_setpos	_setpri
_setprv	_setrtr	_settim	_settmsl	_setuic
_siodst	_skip	_smail	_tranpid	_unlock
_version	_wake	_wakec	_write	_wtpmem

NOTE: To execute any of these system calls across a network, the NETWORK privilege must be set. To execute _clone, _crproc, or _setattr across a network, the SETATTR privilege must be set.

Remote system calls can receive the following diagnostic messages:

errinsufpriv	(1)	The process lacks the privileges required to perform the operation.
errnomemavail	(7)	All available memory has been allocated.
errinvsiteid	(8)	The specified site ID does not exist.
errundefsvc	(43)	The WMCS does not recognize the SVC number used by the process.
errremotelogon	(47)	The process was not allowed to log on to the remote system.
errnodevavail	(52)	No network virtual circuits are available for this operation.
errnonodefnd	(53)	The specified node could not be found.
errnoremcrproc	(55)	Remote process creation is not allowed by the remote system.
errsiteinvalid	(57)	The site ID verification failed for the specified network node.
errdupconnect	(72)	A connect packet was received after the connection was made.
errnoconnect	(73)	An SVC packet was received before the connect packet was received.
errnotremotesvc	(75)	A packet was received for a local-execution-only SVC.
errbadpktsize	(76)	The actual packet size is not the same as the size in the header.
errnoremotemem	(78)	All available memory has been allocated on the remote system.
errwrongos	(79)	The process is incompatible with the current operating system version.
errunknowncmd	(175)	The specified device driver function code is disallowed.
errbuftosmall	(176)	The process buffer is too small for the specified operation.
errnonetbufs	(184)	Not enough network buffers are available for a remote connection.
errnositeid	(194)	The network site ID on this machine is uninitialized.
errnocallestb	(286)	Connection to a remote computer has not been established.
errcallestb	(287)	Connection to a remote computer has already been established.
errdialfailed	(291)	The dial request failed.

WICAT Systems, Inc.
Product-documentation Comment Form

We are constantly improving our documentation, and we welcome specific comments on this manual.

Document Title: ______________________________

Part Number: ______________________________

Your Position:

☐ Novice user	☐ System manager
☐ Experienced user	☐ Systems analyst
☐ Applications programmer	☐ Hardware technician

Questions and Comments **Page No.**

Briefly describe examples, illustrations, or information that you think should be added to this manual.

______________________________ ________

______________________________ ________

______________________________ ________

What would you delete from the manual and why?

______________________________ ________

______________________________ ________

______________________________ ________

What areas need greater emphasis?

______________________________ ________

______________________________ ________

______________________________ ________

List any terms or symbols used incorrectly.

______________________________ ________

______________________________ ________

______________________________ ________

178-001-009 B

First Fold

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 00028 OREM, UTAH

POSTAGE WILL BE PAID BE ADDRESSEE

WICAT Systems, Inc.
Attn: Corporate Communications
1875 S. State St.
Orem, UT 84058

Second Fold

Tape