XOS

Programmer's Guide

For XOS version 3.2

July, 1999

Documentation release 3.0.2

Table of Contents

Chapter 1

Introduction

This publication is intended as both an overview and detailed reference manual for the XOS Application Program Interface (API). General programming information is provided for the XOS operating system, as well as a complete system calls reference.

The intent of this manual is to provide a complete description of the interface between a user program and the XOS kernel. There are no secret or undocumented system calls or functions. Although a few system calls are not intended for direct use by user programs, they are still described here. Most of these are noted as being subject to change in future versions of XOS. This warning should be taken seriously. Every effort is being made to keep the overall XOS API as constant as possible and to only make changes so that new versions will be fully backwards compatible with existing user programs. No major changes, except in those areas indicated, are expected.

Any information missing from this manual is a result of oversight, not intent. Any such omissions that are brought to the attention of the authors will be corrected in future editions.

The 80386, 80486, and Pentium processors all share a common architecture. Throughout this manual, the term “Intel 32-bit architecture” is used to refer to this common architecture.

This manual only describes the XOS native mode API. The DOS emulation mode API is described in the companion document, the XOS DOS/BIOS Emulation Guide, although for most purposes any DOS or BIOS reference can be used when working with DOS programs under XOS.

This is not an introductory text on operating systems. It is assumed that the reader has some familiarity with computers and programming in general and with the Intel 32-bit architecture and assembly language in particular.

The first part of this manual provides an overview of the architecture and capabilities of XOS. The remainder of the manual provides a detailed description of the system calls used to implement this architecture. The Appendices provide lists of error codes and other useful information.

Notation

Numeric values are specified in this manual using the C language conventions. A value that begins with a digit 1 to 9 is a decimal value. One that begins with a 0 is an octal value and one that begins with 0x is a hex value.

The C language conventions are also used to describe the calling sequences for the system calls with the extended storage modifiers *far* and *pascal* used. The modifier *far* indicates that an address is a full segment:offset value. The modifier *pascal* used with a function declaration indicates the function is called using the pascal calling convention. This means that arguments are pushed on the stack in the order given (left to right) and that the called function is responsible for removing the arguments from the stack.

Chapter 2

Structure of XOS

This chapter provides a general introduction to the XOS environment.

XOS is an operating system which supports execution of multiple processes simultaneously. It is created for use on processors which inplement the Intel 32-bit architecture. In addition to compatibility with the DOS and PC-BIOS APIs (including DPMI), XOS provides a native mode 32-bit API which allows access to many advanced features. This native API is described in this manual.

XOS is a protected mode operating system. It uses the processor@146s virtual-86 mode to emulate real mode operation when necessary. Whenever the term real mode is used in this manual, it refers to the emulated real mode implemented using virtual-86 mode.

User Processes

The main function provided by an operating system is to create an environment in which programs can execute. This environment is referred to as a virtual machine. This can be either an exact copy of the raw hardware environment provided by the machine or it can include features not offered by the underlying hardware. XOS is an extended virtual machine environment operating system in that it creates a virtual machine for program execution which offers many features beyond those supported by the CPU.

The extended virtual machine created by XOS for the execution of programs is usually referred to as a user process.

XOS imposes very few restrictions on the structure of a user process. Generally, a user process simply consists of the memory allocated to the process; there are no data areas used by the system which are part of the user process. The exception is the DOS emulator, which does set up and maintain the low memory area and the program segment prefix (PSP). A user process consists of a single execution stream, i.e., each process has a single program counter (PC) which specifies the next instruction to execute and a single set of processor register values. The current version of XOS does not support multiple execution threads in a single process.

Memory is allocated to a process as one or more segments. The Intel 32-bit architecture is very complex and supports a number of modes of operation. XOS provides access to as many of these modes as is practical in a protected multi-processing environment. Two major modes are supported: 32-bit protected mode, called native mode, and virtual-86 mode. The 16-bit protected mode, or 286 mode, is supported as a subset of the 32-bit protected mode. The process mode is associated with the segment from which instructions are being fetched. Usually, a process will possess only a single type of segment: native mode, 286 mode, or virtual-86 mode, although it is possible in special situations to create mixed mode processes which possess more than one type of segment.

Scheduling

A multiprocessing system operating on a single CPU machine such as the PC, must be able to share system resources effectively. All components of a machine can be considered resources and can be grouped into several types. The first of these, and most basic, is the CPU itself. System calls which control the sharing of the CPU are referred to as scheduler functions. They are described in Chapter 9.

Scheduling of system processor time is done on a conventional time-slice basis using three basic priority types. Generally, the priority of a process is raised as a result of the process receiving use of some system resource, such as the process of completing a transfer of data to or from a disk or user terminal. The priority of a process decreases when it is compute bound, i.e., using large amounts of CPU throughput for a period of time. The overall result of this is to give programs which are interacting with a user the highest priority, disk-bound programs the next highest priority, and compute bound programs the lowest priority. The intent of the scheduling philosophy is to provide fast response to the interactive user while still providing rea-

sonably fair allocation of processor time among all processes without requiring a complex and error prone tuning process to make it all work right.

Memory Management and Allocation

The Intel 32-bit architecture implements an extremely complex memory management scheme. It is basically a combination of the traditional paged flat address space memory model and the Intel 80286 segmented address space model. Depending on how the system uses the CPU's memory management functions, either model (or both) can be implemented.

XOS supports the use of full segmented addresses with 32-bit offsets. This lets the user program choose which way to structure memory. It can create a single segment and treat it as a simple 32 flat address space or it can create multiple segments and use as much of each as desired, subject of course to the restrictions on total segment size (4GB) imposed by the CPU. XOS does not attempt to virtualize segments to get around this limit.

When using the multi-segment scheme, the Intel 32-bit architecture is capable of supporting two rather different memory allocation models. The first model is the one supported by the 80286, which uses the processor's segmentation mechanism to manage memory. This will be called the segmented allocation model. Using this model, each unit of memory allocated is generally assigned to a different segment, with each segment being only as large as is needed. The second model is implemented using the paged MMU. It will be referred to here as the paged allocation model. Using this model, memory is allocated in pages, with the size of a page being determined by the hardware. In the case of the Intel 32-bit architecture, the size of a page is fixed at 4096 bytes. Segments are then each made up of one or more pages.

XOS can allocate memory using either allocation models. Normally, the paged is used, but in special cases (for example, whem emulating DPMI), the segmented model can be used also. The address of a location in memory is specified by a selector and an offset. The selector is always 16 bits in length and specifies the segment which contains the location. The offset is 32-bits in length, but is truncated to 16-bits when used in 286 mode. It specifies the position of the memory location in the address space.

Pages are allocated in contiguous groups which are referred to as memory sections or "msects". A memory section may be allocated starting at any offset in a segment which is an integral multiple of the size of a page (4096 bytes). It may contain any

number of pages - up to the physical limits imposed by the hardware being used - but it may not overlap any other memory section. There is no requirement that memory be allocated at any particular offset in any segment. Generally, memory is not allocated at a zero offset, although it can be. Leaving the first few pages of a segment unallocated provides a good way to trap references through uninitialized pointers.

Memory can be shared between processes at the memory section level. A process can also create memory sections which directly map physical memory. Multiple segments can be created which reference the same physical memory. This feature is most often used to allow a single segment to be addressed as both a data segment and as a code segment.

XOS generally attempts to isolate the user from the underlying memory management mechanisms as much as possible. When using the native XOS memory management functions, the user is unconcerned with the mapping of segments into a linear address space using the segmented MMU, which is in turn mapped into physical memory using the paged MMU. XOS does also provide a set of lower level level memory allocation functions which allow the user to directly manage allocation of linear memory and to directly manipulate the physical segment descriptors used by the CPU to define segments. This low level access will generally not be needed by the user and is used mainly to implement the DPMI (DOS Protected Mode Interface) API.

To summarize, XOS allows a program to allocate memory in whatever way and at whatever addresses are most convenient for the program. The scheme is very flexible, allowing multiple contiguous blocks of memory in a single segment, multiple segments, or a combination of both.

The XOS memory management system calls are described in Chapter 10.

Environment Strings

XOS provides a mechanism for associating the definition of various environment strings with a process. An environment string is a simple string of characters associated with a name. Programs executing in the process can retrieve the string by specifying its name. This is similar to the DOS environment capability, but is quite a bit more general in that environment strings can be defined or redefined at any time, without any restrictions on the space required to store the strings. This provides a powerful mechanism for specifying various defaults for programs and for transferring data between programs.

XOS environment strings are maintained by the operating system. The environment string definitions are stored outside of the user process@146 address space and are accessed using system calls. Each process in the system has a private set of environment definitions which are inherited from the process' parent when the process is created.

Even though each process has its own private set of environment string definitions, a process can change the definitions for other processes (privilege restrictions permitting) as well as its own definitions.

When a DOS program is run under XOS, the environment string definitions are copied into a memory block in the DOS virtual machine, completely emulating DOS's use of environment strings. When a DOS program runs an XOS program as a result of a DOS exec function, the XOS environment strings are initialized as specified in the exec function.

XOS also provides a set of system level environment string definitions, which are not directly associated with any process in the system. These definitions are used to initialize the environment string definitions for processes created by the INIT process. These are generally top level command processors or background programs (often referred to as symbionts).

Normally, the environment strings defined for a session's command processor (the session level process) are considered to be the current set of environment strings for the session. The SETENV command displays and modifies these definitions by default. The DOS compatible SET command always modifies these definitions.

Environment strings are typically used for a number of different purposes. The major one is to specify permanent defaults for various programs. Since environment string names are global to a session, there can be a problem of name conflicts if defaults are to be defined for many different programs. XOS has established a convention for naming environment strings used to specify defaults in order to prevent such conflicts. The use of this convention is optional, but it is highly recommended. All environment string names which follow this convention begin with the ^ character and consist of three sections, separated by ^ characters as follows:

^VENDOR^PROGRAM^USAGE

where VENDOR is the name of the vendor of the program which uses the environment string. All environment strings used by XOS utilities use the vendor name XOS. PROGRAM is the name of the program and USAGE is a descriptive name for the individual environment string. For example, the XOS COPY command uses an environment string ^XOS^COPY^OPT to specify default command line options. Most XOS commands use the environment string ^XOS^GCP to specify certain

global options (such as the level of DOS compatibility) which are of interest to all programs.

Note that while it does not follow this convention, the XOS command processor does use the PROMPT environment string to specify the format of the command prompt for compatibility with DOS.

XOS does not use an environment string to specify the directories to search when loading programs as does DOS with its PATH environment string. Instead, it uses the logical name CMD: (which is generally defined as a search list logical). The PATH and SET commands and the DOS environment segment created when a DOS program is loaded use the CMD: logical definition to fake up a DOS PATH environment string for compatibility.

The XOS functions for manipulating environment strings are described in Chapter 8.

Devices

A device is generally thought of as some physical unit connected to the system which can be used to input or output data. Disk drives, tape drives, and terminals are some examples of such devices. XOS extends the definition of a device to also include more abstract devices. The XOS IPM device provides a mechanism for transferring data between processes. Several network devices are implemented which provide access to various levels of the network protocol stacks, all of which use the same physical network interface.

XOS groups devices into device classes. A device class usually consists of all devices of a particular type. For example, the DISK device class consists of all random access mass storage devices directly connected to the system. The TRM device class consists of all terminal devices, which includes serial ports and the console display and keyboard.

XOS accesses all devices using a common set of system calls. These system calls are described in Chapters 15 and 16.

Device Names

XOS uses a 1 to 16 character alpha-numeric name to identify all devices, including disks. The standard convention is that a device name consists of one or more letters which identify the device class, followed by a numeric decimal unit number which identifies the device unit within the class. This may optionally be followed by a single letter followed by another decimal value which identifies a sub-unit. The letter is chosen to indicate the kind of sub-unit. For example, TRM0S3 identifies virtual screen 3 of terminal 0 and D0P2 identifies partition 2 of hard disk 0. TRM2 identifies a serial port (which does not support virtual screens) and D0 identifies an individual hard disk, independent of its partition structure.

Internally, XOS treats device names as a simple character string and does not impose any structure on the name. A device driver can use any name format desired, subject only to the requirement that it be unique in the system. All standard XOS device drivers follow the above naming conventions and it is strongly recommended that user written drivers also follow them.

The initial alphabetic part of the name is usually derived from the device class name, sometimes with one or more letters removed to make the name shorter to allow for unit and sub-unit numbers, but this is not a requirement. A class driver is free to name its devices independent of its class name. For example, the XOS DISK device class uses D to name IDE disks, S to name SCSI disks, and F to name floppy disks. The TRM device class, which includes the console display/keyboard and serial ports, uses TRM to name all of its devices.

Disk class devices also each support two alternate names. The first alternate name can be set using the device characteristics system call and is normally used to name a disk according to the DOS disk naming conventions, i.e., as A, B, C, etc. A utility program, DOSDRIVE, is provided which assigns the same DOS format name to each disk in the system as would DOS. This program is normally run as part of the XOS start up procedure.

The second alternate name is set to be the volume name of the disk, as specified by the disk's file structure. This feature is not used for DOS disks since they do not have a well defined volume name. DOS disks do have a volume label, but it is often not properly formatted for use as a device name and almost always was not chosen as a unique name to use when referencing the disk.

The XOS disk names uniquely identify a disk unit and, where applicable, a partition. Unlike the DOS disk names, the XOS names do not change when the system configuration is changed by adding or removing disks. Floppy disks are identified as Fn,

with F0 being the DOS A and F1 being the DOS B. If a system contains additional floppy disks, they are normally named as F2, F3, etc., although there is no restriction that the numbers be contiguous.

IDE disks are named as Dn and DnPm and SCSI disks are named as Sn and SnPm. The Dn and Sn format names represent an entire disk independent of its partition structure. If adisk is not partitioned, (as is the case with some removable hard disks) this name is also used to access the file structure on the disk. With a partitioned disk, it is normally only used when referencing the partition table on the disk, although it can also be used to perform raw mode reads and writes to the entire disk, independent of the disk's partitions. The DnPm or SnPm format name represents an individual disk partition. D0P1 is the first partition on IDE disk unit 0 (the first hard disk). This is normally the DOS disk C, although since different versions of DOS set up the disk partition table differently, this is not always the case.

XOS attempts to assign partition numbers in the same order as DOS, but may not succeed in all cases, especially with disks which were set up using third party disk partitioning software. XOS first scans the partition table looking only for standard DOS partitions (including huge partitions) and assigns these partitions numbers in the order they are found. It then scans the partition table looking for extended DOS partitions and assigns partition numbers to the logical volumes contained in the extended partitions in the order they are found. Finally, it scans the partition table looking for any remaining non-DOS partitions and assigns partition numbers to them in the order they are found.

XOS supports all standard DOS partition formats (12-bit, 16-bit, and 32-bit FATs) and disks using the EZ-Drive, EZ-BIOS BIOS enhancements.

File Specifications

A file specification completely specifies an individual file on some disk in the system or on a remote disk. It has the following general format:

DEV:NETADDR::RDEV:PATH\NAME.EXT

where DEV is the device name (as discussed above), NETADDR is an optional network address (used only when DEV specifies a network device), RDEV is the remote device name (also used only when DEV specifies a network device), PATH is the directory path, NAME is the file name, and EXT is the file extension.

If DEV and NETADDR are not included, the device Z: is assumed. If DEV is not included and NETADDR is included, the device NET: is assumed.

When an XOS network device is used to access files on a non-XOS remote system, (also referred to as a foreign system) the format of the part of the file specification following the :: is specified by the remote system. It may have any format required by the remote system subject only to the requirement that characters which XOS does not allow in a file specification cannot be used. One such character is / (which is reserved as a switch/option prefix by XOS). this causes a potential problem when accessing files on a remote UNIX system. In this case, XOS converts the \ character to /, allowing UNIX file specifications to look much like XOS file specifications.

Note that while the XOS device drivers treat the remote part of the file specification as an arbitrary string, some XOS utilities (such as DIR) make some assumptions about the format of a file specification. This means that some advanced options (such as the ... notation) may not work with all foreign systems.

XOS supports the use of multiple periods in a file specification. All characters before the last period make up the file name. All character after the last period make up the file extension. A period is not allowed at the end of the file specification. If there is not period in the file specification, the extension is null.

When a network device is specified, the remote device can also be a network device, allowing any level of multiple remote access. This is generally not a desirable (efficient) way to access remote files, but may be necessary in the special case where a remote system has access to a second network to which the local system is not connected and does not provide automatic routing or bridging between the networks.

Wildcard File Specifications

Most XOS system calls which take file specifications as arguments allow partially specified or wildcard file specifications. A wildcard file specification includes one or more of the following wildcard characters:

Character	Description
?	Matches any single character
*	Matches any number of characters

For example, the wildcard specification A*X.ZZZ would match the names AX.ZZZ, ABX.ZZZ, and ABCX.ZZZ. It would not match ABC.ZZZ or XA.ZZZ.

A?X.ZZZ would match ABX.ZZZ but would not match AX.ZZZ or ABCX.ZZZ, since the ? matches exactly one character only. The range of characters matched by a * terminates at the period which separates the name and extension. Thus *.XY would match any file name with the extension XY but would not match ABC.DXY. *.*XY would match ABC.DXY, however.

Destination Wildcard File Specifications

XOS does not allow wildcards in file specifications used to specify a destination. This includes any file which is to be created and the new name for a rename operation. Some of the XOS utilities do, however, allow a form of wildcard usage in destination specifications.

In this case, only the * wild-card character is allowed. It must appear alone in either or both the name and extension parts. It means to copy the entire name of extension from the source file specification.

Logical Names

XOS also supports a system of logical names. A logical name has the same format as a device name and can be used any place that a device name can be used. It is an arbitrary name which is defined to be equivalent to another device name, optionally followed by a string of characters (which normally represents a directory path). Logical name definitions can be nested, i.e., the definition of a logical name may be a logical name. This nesting is limited to 6 levels to make detection of definition loops reasonably efficient.

A logical name can be substituted or assigned. An assigned logical name is a simple alternate name for a device. The definition must be a device name only. No directory path is allowed. When an assigned name is used, the defined name is simply substituted for the logical name. If an attempt is made to set a current directory path for an assigned logical name, the current directory path is set for the underlying device. A substituted name behaves like a device name in its own right. Its definition can include a directory path specification. More importantly, a current directory path can be associated with the name, just as if it were a physical disk name. When a substituted logical name is expanded, the current directory path associated with the logical

name is appended to the definition of the logical name. The resulting string replaces the logical name in the file specification.

It should be noted that XOS assigned logical names are mostly equivalent to the DOS logical disk names created with the DOS ASSIGN command. XOS substituted logical names are mostly equivalent to the DOS logical disk names created with the DOS SUBST command.

A substituted logical name can be defined to represent a disk on a remote system. This name then behaves exactly like a local disk name. For example, if we have a file C:\SOME\WHERE\FOO.BAZ on the remote system LIZARD::, we can use the following definitions:

```
LOGICAL/SUB L:=LIZARD::C:
CD L:\SOME\WHERE
TYPE L:FOO.BAZ
```

Alternately, we could use:

```
LOGICAL/SUB L:=LIZARD::C:\SOME\
CD L:\WHERE
TYPE L:FOO.BAZ
```

Two special logical names are built into XOS. The name Z: is used as a default device name when no device name or network address is specified. The definition of this name thus specifies the default disk. The name NET: is used as a default device name when only a network address is specified. Since most XOS configurations will support only one network connection, this name will normally be defined to be the remote file system device using that interface (usually XFP0:). In configurations supporting more than one network connection, it is used to specify which one is the default network connection.

Input/Output Operations

XOS supports a mixture of direct and queued IO operations. Direct IO operations provide the same IO environment as DOS. All operations are blocking, in that execution of the program doing the IO cannot precede normally until the IO operation is complete. In most cases signals can be granted while waiting for IO to complete and in some cases the IO operation can be aborted by simpily executing another IO request for a signal routine. This method is often used to abort input from the controlling terminal. Queued IO, on the other hand, implements a tightly defined sequence

of IO operations. IO operations, in general, are non-blocking. When a program requests an IO operation, control returns immediately to the program, Execution of the program and the IO operation then precede in parallel. If a second IO operation is started on the same device before the first one completes, the second is queued and is then executed automatically when the first one completes. A program can poll the status of an IO operation, can wait until it is complete, or can requst a signal when it is complete.

Queued IO provides a much more powerful and better defined method for doing input and output. Unfortunately, its exclusive use has some undesirable side-effects, especially when dealing with programs originally written to execute under a system (such as DOS) which only implements direct IO. The most significant problem involves attempting to abort terminal input by doing another input operation. This simpily does not work in a queued environment. The second operation will be queued and will wait (probably for a long time) until the first operation is complete. This can be solved by having the program explicitly cancel the first input operation before issueing the second request, but this requires a non=trivial change to the program.

XOS attempts to provide the best of both worlds by supporting both methods of doing IO. Not all XOS devices support both methods. Devices (like disks) whose IO operations always complete in a short amount of time can (and do) use queued IO for all IO operations without creating any problems for the programmer. Devices which may not complete IO in a finite time (such as terminals) implement both queued and direct IO operations. A few devcies (notabiliy the console display) do not implement queued IO because all IO operations to these devices involve direct CPU data transferes. The CPU never waits for the device to complete a data transfer.

The DOS emulator does several things to ensure that IO operations behave as expected. First, it uses direct IO whenever it is available. Second, when a device (as as a disk) does not support direct IO, it disables signals during a device transfer so there will be no confussion as to when and in what order operations are done. This should not cause any major problems, since most DOS program do not use interrupts (which are emulated using signals under XOS) and never use XOS signals directly. A mixed DOS/XOS program should use XOS IO calls for any IO operations which need to be interruptable by signals.

Chapter 3

The Programming Environment

XOS supports four different programming environments. First, it provides an environment which closely emulates the environment provided by DOS running on an IBM PC or compatible, including emulation of most of the BIOS functions and direct hardware access commonly used by DOS programs. Second, XOS also provides a 32-bit DOS environment which supports the DPMI (DOS Protected Mode Interface) specification. Third, the virtual-86 mode is also used to implement the 16-bit XOS environment. This environment provides access to nearly all of the advanced features of XOS from programs developed using the 16-bit development tools designed for DOS. Finally, XOS supports a full 32-bit programming environment which provides full access to the features of the Intel 32-bit architecture, including full access to the 45 bit memory address provided by the Intel 32-bit architecture.

These four programming environments are not really separate from each other, although it is simpler to describe them as if they are. It is most accurate to think of a hierarchy of environments, with the 32-bit XOS environment at the top. A program running in this environment can create and load code into a 16-bit virtual-86 memory segment and transfer control to this code. This 16-bit code is then running in the 16-bit XOS environment. If the appropriate data structures are set up in the first 4KB of the virtual-86 segment and if the console display is mapped into the virtual-86 segment at the correct address and the physical BIOS ROMs are also mapped, then the program can execute DOS and BIOS functions and can be thought of as running in the 16-bit DOS environment. Among the functions which can be executed are the DPMI set up functions, which set up some data structures in the process's memory and allocate and initialize some segment descriptors, thus creating the 32-bit DOS environment.

This is actually a fairly accurate description of the way in which the various environments are created by XOS. Thus, the 16-bit DOS environment starts out as a 32-bit

XOS environment, which creates a 16-bit XOS environment, which creates a 16-bit DOS environment.

The XOS 32-bit Environment

This is the native XOS programming environment. It uses the protected mode of the processor and supports full 32-bit offsets. Programs running in the XOS 32-bit environment can be structured with multiple segments (each of which has a full 32-bit address space) or with the flat memory model, which uses one code and one data segment which map to the same physical memory, effectively converting the Intel 32-bit architecture from a segmented to a flat memory address architecture. A common varient of the flat model is one which uses two separate segments for code and data. This is used in almost exactly the same was as the pure flat model but also automatically protects all code from accidental modification by the program. It is also possible to use a mixture of these two memory models, using what is basically a flat model but allowing additional segments to be allocated and accessed as needed. Most of the user mode code provided as part of XOS is written using this mixed model.

The XOS API has been designed primarily to support the XOS 32-bit environment. It is expected that virtually all code written specifically for XOS will be written for this environment.

The XOS 16-bit Environment

This environment makes use of the Intel 32-bit architecture's virtual-86 mode to emulate the behavior of the 16-bit 8086 processors. The XOS native API is fully available in this environment, although the various data structures associated with the system calls still use 48-bit address fields. The XOS 16-bit environment is provided mainly to allow existing DOS programs access to the XOS API. While it is feasible to write native XOS programs specifically for the 16-bit environment, there is usually no benefit to be gained by doing this.

The DOS 16-bit Environment

This environment also makes use of the Intel 32-bit architecture's virtual-86 mode to emulate the behavior of the 16-bit 8086 processors. Virtually all of the documented DOS API functions and many of the undocumented functions are implemented in this environment. Also, the commonly used BIOS functions are implemented. Direct access to the console display interface and to the keyboard interface is also supported. This allows many DOS programs to run in this environment under XOS.

Any program executing in this environment can execute native XOS system calls at any time, even though it was loaded as a DOS program.

The DOS 32-bit Environment

The DOS 32-bit environment is an extension of the DOS 16-bit environment based on the DPMI (DOS Protected Mode Interface) specification version 0.9. It provides a complete DOS extender which supports direct execution of almost all DOS and BIOS functions in 32-bit protected mode. It is also compatible with programs which implement their own DOS extenders using the DPMI functions.

It should be noted that DOS 32-bit programs are always initially loaded as 16-bit programs which switch to the 32-bit environment during initialization.

Any program executing in this environment can execute native XOS 32-bit function calls at any time, even though it was loaded as a DOS DPMI program.

Chapter 4

Process Privileges

This chapter describes the privileges which may be associated with a process. A privilege is a named item which grants a process certain rights to access information or perform a function. When a process is created, it is given a list of available privileges. Any privilege in this list can then be made current by the process by setting the value of the CURPRIV PROCESS class characteristic to include that privilege (see Chapter 12). Privileges can be removed from the list of available privileges by changing the value of the AVLPRIV PROCESS class characteristic, but no privileges that are not in the list of available privileges can be added once the process has been created. If an available privilege is removed, it **cannot** be put back later.

> Note that if the user login feature is not used, each top level (command shell) process created is given all possible privileges as both current and available privileges.

Privileges are specified as a list with the elements separated by + or - characters. If the first character in the list is + or -, the list represents an incremental specification, and the privileges listed are added or removed for the process, depending on the character before each privilege name. Thus the list:

```
+ALLIO-BYPASS
```

would add the ALLIO privilege and remove the BYPASS privilege. If the first character is not + or -, the list is an absolute list and the privileges listed replace the current privileges. In this case, all of the separator characters must be +. Any privilege preceded by - is ignored. All changes to the privileges (either current or available) are made considering the restrictions listed above. Thus, privileges specified as current privileges are effectively anded with the available privileges. Privileges specified as available privileges are anded with the previous available privileges.

Most programs require no more than the IPM privilege to execute correctly. Privileges provide access to critical parts of the system and are very powerful and therefore should not be granted to processes unless absolutely necessary.

Table 4.1 contains a summary of the process privileges.

Figure 4.1 - Process privileges

Name	Description
ADMIN	System administrator privileges
ALLIO	May do restricted I/O operations
ALLPROC	May kill or interrupt any process
BYPASS	Bypass all file access checking
CHNGUSER	May change user name
DETACH	May detach process
IPM	May send IPM messages to other processes
LKELOAD	May execute LKE load functions
MEMLOCK	May lock memory pages in place
NEWSES	May create new process group
NOSWAP	May lock memory pages in memory
OPER	Operator privileges
READALL	Bypass file read access checking
READKER	May read kernel memory
READPHYS	May read physical memory
SCREENSYM	May execute screen symbiont functions
SESENV	May change group level environment strings
SHAREDEV	May share any device
SPCUSER	May specify user for acesss checking
SYSENV	May change system level environment strings
SYSLOG	May change system level logical names
USESYS	Access files using system protection
WRITEKER	May write kernel memory
WRITEPHYS	May write physical memory

Below is a detailed description of the process privileges.

ADMIN = System administrator privileges

This privilege allows various functions associated with managing the system. Among these are setting the system date and time and allocating disk buffers. This is one of the most powerful privileges. Normal programs should not be executed when it is in effect.

ALLIO = May do restricted I/O operations

This privilege allows restricted I/O operations. This includes directly reading and writing disk blocks and other operations at this level.

ALLPROC = May kill or interrupt any process

This privilege allows the process to use the svcIoKill system call to kill any process in the system, not just child processes of the process issuing the svcIoKill system call.

BYPASS = Bypass all file access checking

When this privilege is in effect, ALL file access privilege checking is by-passed. This allows complete read and write access to all files on the system.

CHNGUSER = May change user name

When this privilege is in effect, the process can change its user and group names at will.

DETACH = May detach process

When this privilege is in effect, the process is allowed to detach from its controlling terminal.

IPM = May send IPM messages to other processes

This privilege allows the process to send messages to other processes using the IPM device. This privilege will be in effect for most processes.

LKELOAD = May execute LKE load functions

When this privilege is in effect, the process can execute the svcSysLoadLke system call, which is used to load loadable kernel extensions (LKEs).

MEMLOCK = May lock memory pages in place

This privilege is not implemented yet.

NEWSES = May create new session

When this privilege is in effect, a process can create a child process which is the base process for a new session.

NOSWAP = May lock memory pages in memory

This privilege is not implemented yet.

OPER = Operator privileges

This privilege is not implemented yet.

READALL = Bypass file read access checking

When this privilege is in effect, all file system privilege checks for read accesses are by-passed. Privilege checking for write accesses is unchanged.

READKER = May read kernel memory

When this privilege is in effect, the process can use the svcMemLink system call to link to exec with read only access.

READPHYS = May read physical memory

When this privilege is in effect, the process can use the svcMemMap system call to map physical memory into its address space for read only access.

SCREENSYM = May execute screen symbiont functions

When this privilege is in effect, the process can execute the screen symbiont system calls. Normally, only the process running the screen symbiont will have this privilege.

SESENV = May change session level environment strings

When this privilege is in effect, the process can modify environment strings defined at the session level.

SHAREDEV = May share any device

When this privilege is in effect, the process can obtain shared access to any device, even if the device is in use by a different session and does not normally allow such sharing.

SPCUSER = May specify user for acesss checking

When this privilege is in effect, the IOPAR_USER to specify the user name to be used for access checks and IOPAR_ACSNETWK IO parameters may be used to specify that the network access protection class should be used. This privilege is intended to be used by network file servers.

SYSENV = May change system level environment strings

When this privilege is in effect, the process can modify environment strings defined at the system level.

SYSLOG = May change system level logical names

When this privilege is in effect, the process can modify logical names defined at the system level.

USESYS = Access files using system protection

When this privilege is in effect, all file access checking is done using the system access level.

WRITEKER = May write kernel memory

When this privilege is in effect, the process can use the svcMemLink system call to link to exec with read/write access.

WRITEPHYS = May write physical memory

When this privilege is in effect, the process can use the svcMemMap system call to map physical memory into its address space for read/write access.

Chapter 5

The Signal System

The XOS signal system closely parallels the *interrupt* systems implemented at the hardware level for most computers. The term *signal* is used here instead of *interrupt* to attempt to minimize the confusion caused by the misuse of the word *interrupt* when referring to the Intel 32-bit architecture. The word *interrupt* usually refers to an asynchronous event which causes a change in program execution. When discussing the Intel architecture, the word *interrupt* is also used when describing the INT instruction, which should really be described as causing a *trap*. A *trap* is a change in program execution (much like a subroutine call) which is caused by the processor itself, not by an external event.

The term *signal* is used unambiguously in the Unix and other environments to refer to a true externally caused *interrupt*. It is used the same way here. Synchronous or processor generated *interrupts* are referred to as *traps*.

A *signal* or *trap* causes the current execution state to be saved and control to be transferred to a specific address when some event occurs. This allows the program to take whatever action is necessary to respond to the event and then to return to whatever it was doing before the event occurred. Each possible event is assigned to a vector. A vector specifies the address to which control is transferred, the format in which the current execution state is saved (referred to as the vector's type), and a priority level.

The XOS signal system is a fully prioritized system. At any time, a process is executing at some priority level. When a *signal* is requested, the level associated with the *signal's* vector is compared to the level of the process. If the vector's level is higher than the level of the process, the *signal* is granted and the level of the process is set to the level of the vector. The process stays at this level until a higher level *signal* is granted or until this *signal* is dismissed, at which time the previous level is restored. If the level of the vector is the same or lower than the current level of the process, the *signal* request is held off and not granted until the level of the process is

reduced below the level of the vector. A special case occurs if the vector's level is the highest possible level. In this case, the *signal* is granted immediately, even if the process is already at this level.

Traps operate much like signals except that they are not level sensitive. A *trap* always occurs when requested, regardless of the process' current level. It does not change the level of the process.

Eight priority levels are defined for signals: levels 0 through 7. Level 0 is the main program level, where most programs execute. Signals are fully nested, with level 7 having the highest priority and being *non-maskable*; that is, it can *interrupt* itself. Signals may be disabled entirely by setting the *signal* level to a value of 8 or higher.

XOS separates the vectors used for processor traps from the vectors used by the INT instruction and external events. If the vector or a real mode *trap* is not set and the corresponding real mode vector (obtained by subtracting 0x120 from the vector number of the processor *trap*) is set, the real mode vector is used. This provides compatibility with 16-bit DOS programs which do not know about the special processor *trap* vectors. 32-bit DOS programs use the processor *trap* vectors directly through the use of the DPMI set exception vector functions.

Table 5.1 summarizes the general vector groups.

Table 5.1 - Vector Groups

Range (dec.)	Range (hex)	Description
0-255	0-0xFF	Protected mode signals or traps
256-287	0x100-0x11F	Protected mode processor traps
288-319	0x120-0x13F	Real mode processor traps
320-351	0x140-0x15F	XOS pre-assigned signals and traps
352-511	0x160-0x1FF	Reserved
512-767	0x200-0x2FF	Real mode signals or traps

The protected mode signals and traps (0-0xFF) are the usual 256 protected mode vectors provided by the Intel 32-bit hardware. It should be noted, however, that the first 32 vectors (0-0x1F) are not used as processor *trap* vectors. These first 32 vectors can be used for signals or INT instruction vectors.

Vectors 0x100 through 0x11F are the protected mode processor *trap* vectors. These vectors are used when the corresponding processor exception occurs in protected mode. They cannot be invoked using the INT instruction. If a processor *trap* occurs and the corresponding vector is set to type VT_NONE, or to the default vector, the process is terminated. These vectors are summarized in Table 5.2.

Table 5.2 - Protected mode processor trap vectors			
Name	Number (dec)	Number (hex)	Description
VECT_PDIVERR	256	0x100	Divide error trap
VECT_PDEBUG	257	0x101	Debug trap
VECT_PNMI	258	0x102	Non-maskable interrupt
VECT_PBRKPNT	259	0x103	Breakpoint trap
VECT_PINTO	260	0x104	INTO instruction trap
VECT_PBOUND	261	0x105	BOUND instruction trap
VECT_PILLINS	262	0x106	Illegal instruction trap
VECT_PFPPNAVL	263	0x107	FPP not available trap
VECT_PDBLEXP	264	0x108	Double exception trap
VECT_PFPPSOVR	265	0x109	FPP segment overrun trap
VECT_PITSS	266	0x10A	Invalid task state segment trap
VECT_PSEGNP	267	0x10B	Segment not present trap
VECT_PSTKERR	268	0x10C	Stack error trap
VECT_PPROT	269	0x10D	General protection trap
VECT_PPAGEFLT	270	0x10E	Page fault trap
	271	0x10F	Reserved
VECT_PFPUERR	272	0x110	FPP error trap
VECT_PALNCHK	273	0x111	Alignment error trap
	274-287	0x112-0x11F	Reserved

Vectors 0x120 through 0x13F are the real mode processor *trap* vectors. These vectors are used when the corresponding processor exception occurs in real mode. They cannot be invoked using the INT instruction. If a processor *trap* occurs and the corresponding vector is set to type VT_NONE, the process is terminated. If the corresponding vector is set to the default vector and a DOS environment exists, the corresponding real mode vector is used if it is set. Otherwise the process is terminated. These vectors are summarized in Table 5.3.

Table 5.3 - Real mode processor trap vectors			
Name	Number (dec)	Number (hex)	Description
VECT_RDIVERR	288	0x120	Divide error trap
VECT_RDEBUG	289	0x121	Debug trap
VECT_RNMI	290	0x122	Non-maskable interrupt
VECT_RBRKPNT	291	0x123	Breakpoint trap
VECT_RINTO	292	0x124	INTO instruction trap
VECT_RBOUND	293	0x125	BOUND instruction trap
VECT_RILLINS	294	0x126	Illegal instruction trap

Table 5.3 - Real mode processor trap vectors			
Name	Number (dec)	Number (hex)	Description
VECT_RFPPNAVL	295	0x127	FPP not available trap
VECT_RDBLEXP	296	0x128	Double exception trap
VECT_RFPPSOVR	297	0x129	FPP segment overrun trap
VECT_RITSS	298	0x12A	Invalid task state segment trap
VECT_RSEGNP	299	0x12B	Segment not Present
VECT_RSTKERR	300	0x12C	Stack error trap
VECT_RPROT	301	0x12D	General protection trap
VECT_RPAGEFLT	302	0x12E	Page fault trap
	303	0x12F	Reserved
VECT_RFPUERR	304	0x130	FPP error trap
VECT_RALNCHK	305	0x131	Alignment error trap
	306-319	0x132-0x13F	Reserved

Vectors 0x140 through 0x15F are the XOS pre-assigned *signal* and *trap* vectors. These vectors are used when the corresponding condition occurs. If the vector type is VT_NONE or if it is set to the default vector, the default action is taken. These vectors are summarized in Table 5.4.

Table 5.4 - XOS pre-assigned vectors			
Name	Number (dec)	Number (hex)	Description
VECT_EXIT	320	0x140	XOS process termination trap
VECT_CHILD	321	0x141	XOS child died interrupt
VECT_CNTC	322	0x142	XOS control-C interrupt
VECT_CNTP	323	0x143	XOS control-P interrupt
VECT_HNGUP	324	0x144	XOS terminal hung up interrupt
	325-511	0x145-0x1FF	Reserved

Vectors 0x200 through 0x2FF are the real mode vectors. These vectors only exist when a DOS real mode environment exists. The contents of these vectors is stored in the first 1024 bytes of the real mode segment in the normal real mode hardware format. They can be set either by using the XOS svcSetVector function, the DOS INT 21h, function 35h call, or by directly storing a value into the vector. When a process exception occurs in real mode and the corresponding real mode process *trap* vector is not set, then the corresponding real mode vector is used. These vectors are used directly when an INT instruction is executed in real mode. Since these vectors only exist when a DOS environment exists, this implies that INT instructions can only be used in real mode when a DOS environment exists. An attempt to use one of these vectors when a DOS environment does not exist will terminate the process.

Vectors 0 to 0x15F each have a type. Table 5.5 lists the vector types.

Table 5.5 - Vector types

Name	Val	Dismiss instruction	Lvl	Description
VT_NONE	0			Vector is not in use
VT_XOSS	1	CALLF svcSchDismiss	n	XOS signal vector
VT_XOST	2	CALLF svcSchDismiss	n	XOS trap vector
VT_HWS16	3	IRET	4	16-bit hardware signal vector
VT_HWS32	4	IRET	4	32-bit hardware signal vector
VT_HWT16	5	IRET	4	16-bit hardware trap vector
VT_HWT32	6	IRET	4	32-bit hardware trap vector
VT_DPMI16O	7	RET	4	DPMI version 0.9 16-bit trap vector
VT_DPMI32O	8	RET	4	DPMI version 0.9 32-bit trap vector
VT_DPMI16N	9	RET	4	DPMI version 1.0 16-bit trap vector
VT_DPMI32N	10	RET	4	DPMI version 1.0 32-bit trap vector

The Dismiss Instruction column specifies the instruction which is used to return from or dismiss a *signal* or *trap* for the vector type indicated. The Lvl column specifies the level associated with the vector. Note that the XOS vectors have settable levels but that all other vectors have a fixed level of 4.

Each vector is indicated as being either a *trap* vector or a *signal* vector. This refers to the way the vector affects the priority level of the process when it is used. It does not mean that only traps or signals can be handled by the vector. Any vector type can be used for either signals or traps.

Vectors indicated as *trap* vectors do not change the *signal* level of the process when a *signal* or *trap* is granted. The level of the vector is used, however, in determining when to grant a *signal*. Vectors indicated as *signal* vectors set the level of the process to the vector's level (except that the level of the process is never decreased as the result of granting a *signal* or *trap*) when a *signal* or *trap* is granted. Traps are always granted immediately, independent of the vector type, even if the level of the process is greater than the vector's level.

The following explains how a *signal* is handled. A user process normally executes at *main program* level, or level 0. When a *signal* occurs, XOS saves the process' state (its current program counter and status register, and optionally, some register values) onto the process' stack in the format associated with the vector's type. If the vector is an XOS vector, XOS may push one or more words of *interrupt* data onto the process' stack, depending on the source of the *signal*. XOS sets the signal level to

that of the signaling event and continues execution at the address appearing in the vector. It is the responsibility of the user *signal* routine to preserve all registers. For some vector types (but not for all) the segment registers are saved on the user's stack before the *signal* routine is called. After the *signal* has been serviced and before the user *signal* routine returns, it restores registers. Then it executes the appropriate instruction to dismiss the *interrupt* (given in Table 5.2), causing XOS to restore the user program counter and status register and to discard any *interrupt* data from the stack. Restoring the status register restores the previous *signal* level as well as the condition bits.

The format of the data stored on the stack varies depending on the type of the vector and, for some vector types, the processor modes of the interrupted code and of the *signal* routine. In some cases it is necessary to store the address of a special *dismiss routine* when the instruction, which will be used by the user program to return from the *signal* or *trap* routine, is not capable of fully restoring the previous state of the process. In the following descriptions, the word *saved* is used to indicate items which represent the actual saved state of the process and the word *dismiss* is used to indicate items which specify the special dismiss routine. In general, user programs can freely change the *saved* items to modify the process which will be restored. The *dismiss* items should **NEVER** be modified by the user program.

The following sections describe the *signal* and *trap* stack formats in detail.

Stack Formats for XOS Vectors

All XOS vectors generate the same stack format independent of the modes of the interrupted code and of the *signal* routine, except that the orginal SS and original ESP values are only present when switching from protected to real mode or from real mode to protected mode. The stack items are always 32 bits, even if interrupting real mode code to a real mode *signal* routine. The number of data items present depend on the source of the *signal*. It may be 0.

The EFR value stored in the XOS format stack frame is similar to the 80386 hardware EFR value. It is modified slightly to include the *signal* level, which is stored in the bits which contain the nested task flag and the I/Oprivilege level in the hardware EFR value. This is shown below. The *interrupt* enable bit (IF) is used as the complement of the 4th bit of the *signal* level. This results in its having its usual function of enabling signals (interrupts) when set. (This works since *signal* levels of 8 or greater indicate that signals are disabled.) For additional information on the use of the EFR

bits when dismissing signals or interrupts, see the discussion of the svcSchDismiss and svcSchIRet system calls in Chapter 9. The format of the EFR is shown below.

0	0	0	0	0	0	0	0	0	0	0	0	0	AV	VM	RF	0	SIL	OF	DF	IF	TF	SF	ZF	0	AF	0	PF	1	CF

This format for the EFR value is used in both 16-bit and 32-bit XOS stack frames.

The format of the XOS stack frame is show below.

Offset	High order word	Low order word
(36+(4*N))	Not used	Saved SS
(32+(4*N))	Saved ESP	
(28+(4*N))	Nth data item	
(24+(4*N))	N-1th data item	
36	...	
32	1st data item	
28	Signal number	Number of data items
24	Saved EFR	
20	Not used	Saved CS
16	Saved EPC	
12	Not used	Saved DS
8	Not used	Saved ES
4	Not used	Saved FS
0	Not used	Saved GS

The saved SS and saved ESP items are only present when switching between protected and real modes. The number of data items (each item is 4 bytes) depends on the source of the *trap* or *signal* and may be between 0 and 32. Note that, unlike the other stack frame formats, the values for the segment registers are always present, even if no mode switch occurs.

Stack Formats for Hardware Vectors

Different stack formats are used for hardware vectors depending on the mode of the interrupted code and the mode of the *signal* or *trap* routine.

The FR value used is almost identical to the 80386 EFR value. The *interrupt* enable bit (IF) reflects the state of the emulated hardware *interrupt* system. Since in real

mode hardware interrupts are implemented as level 4 signals, the IF bit is the complement of bit 2 of the *signal* level. When an IRET instruction is executed in real mode, this bit is used to restore bit 2 of the *signal* level only. The other bits of the *signal* level are not changed. This allows programs which are aware of the XOS *signal* system to run in a DOS environment with DOS programs (such as debuggers) which only know about hardware interrupts without conflicts. See the discussion of the svcSchDismiss and svcSchIRet system call in Chapter 9 for a more information about how this *signal* level is restored when dismissing a *signal* or *interrupt*. The unused bits in the EFR emulate the behavior of the 80386 EFR. The FR is the low order 16 bits of the EFR. This is summarized below.

0	0	0	0	0	0	0	0	0	0	0	0	0	AV	VM	RF	0	0	0	0	OF	DF	IF	TF	SF	ZF	0	AF	0	PF	1	CF

If the *signal* routine is in a 32-bit protected mode segment and the interrupted code is 16 or 32-bit protected mode code, the following stack format is generated:

Offset	High order word	Low order word
8	Saved EFR	
4	Not used	Saved CS
0	Saved EIP	

Note that this is the same as the 80386 32-bit hardware *interrupt* stack format.If the *signal* routine is 32-bit protected mode code and

the interrupted code is real mode, the following stack format is used:

Offset	High order word	Low order word
28	Not used	Saved SS
24	Saved ESP	
20	Saved EFR	
16	Not used	Saved CS
12	Saved EIP	
8	Dismiss EFR	
4	Not used	Dismiss CS
0	Dismiss EIP	

It is necesary to store the address of a special dismiss routine as the address to which the *interrupt* routine returns since it will normally return with an IRET instruction, which will not switch to real mode when executed in user mode.

If the *signal* routine is 16-bit protected mode code and the interrupted code is also 16-bit protected mode code, the following stack format is used:

Offset	Word
4	Saved FR
2	Saved CS
0	Saved IP

Note that this is the same as the 80386 16-bit hardware *interrupt* stack format.

If the *signal* routine is 16-bit protected mode code and the interrupted code is either real mode or 32-bit protected mode code, the following stack format is used. The saved GS, FS, DS, ES, SS and ESP values are only present if the interrupted code is real mode.

Offset	High order word	Low order word
38	Not used	Saved GS
34	Not used	Saved FS
30	Not used	Saved DS
26	Not used	Saved ES
28	Not used	Saved SS
24	Saved ESP	
20	Saved EFR	
16	Not used	Saved CS
12	Saved EIP	
8	Dismiss EFR	
4	Not used	Dismiss CS
0	Dismiss EIP	

If the *interrupt* routine is real mode code and the interrupted code is either 16 or 32-bit protected mode, the following stack frame format is used.

Offset	High order word	Low order word
38	Not used	Saved GS
34	Not used	Saved FS
30	Not used	Saved DS
26	Not used	Saved ES
22	Not used	Saved SS
18	Saved ESP	
14	Saved EFR	
10	Not used	Saved CS
6	Saved EIP	
4	Dismiss FR	
2	Dismiss CS	
0	Dismiss IP	

The first 3 words on the stack provide a standard real-mode hardware *interrupt* stack frame which points to a real mode dismiss routine which calls the svcSchIRet function to actually return to the interrupted code.

Stack Formats for DPMI Vectors

The DPMI vector types are provided mainly for use by the XOS DPMI emulation routines. While they can be specified directly by user programs, there should generally be no need to do this. It is strongly recommended that user programs use either XOS or hardware type vectors in all cases. The stack frames produced for DPMI vectors are the stack frames defined in the DPMI specifications. They are only valid for protected mode *signal* or *trap* routines. DPMI only uses these vector types for processor traps. XOS does not restrict their use in this way, but there should be no reason to use them in other ways.

DPMI version 0.9 stack frames are only valid for interrupts from protected mode code to protected mode *trap* routines. Since DPMI version 0.9 does not support a mixture of 16 and 32-bit protected mode code, it is also not possible to have a different width for the *trap* routine than for the interrupted code. If an attempt is made to use a DPMI version 0.9 vector other than in this way, the process will be terminated with a TC_BADSTK termination code.

The error code stored in the DPMI stack frame is only valid when used with the VECT_RSEGNP and VECT_RPROT vectors. In these cases it gives the selector

which caused the error as reported by the underlying hardware exception. For all other vectors the value stored in this field is undefined and should be ignored.

The stack frame format for a 16-bit DPMI version 0.9 stack frame is shown below.

Offset	Word
14	Saved SS
12	Saved SP
10	Saved FR
8	Saved CS
6	Saved IP
4	Error code
2	Return CS
0	Return IP

The stack frame format for a 32-bit DPMI version 0.9 stack frame is shown below.

Offset	High order word	Low order word
28	Not used	Saved SS *
24	Saved ESP	
20	Saved EFR	
16	Not used	Saved CS
12	Saved EIP	
8	Error code	
4	Not used	Return CS
0	Return EPC	

XOS also implements the DPMI version 1.0 stack frames. These stack frames are not used by the current version of the XOS DPMI emulator, since it only supports DPMI version 0.9. It is expected that a future version of the XOS DPMI emulator will use these vectors.

The format of the DPMI version 1.0 stack frame is somewhat unusual in that it consists of a version 0.9 stack frame with an second, extended stack frame stored above it. This allows programs to use either stack frame, and, in theory, allows mixing of version 0.9 and version 1.0 *interrupt* handlers in the same program.

The format of the 16-bit DPMI version 1.0 stack frame is show below.

Offset	High order word	Low order word
84	PTE value	
80	CR2 value	
76	Not used	Original GS *
72	Not used	Original FS *
68	Not used	Original DS *
64	Not used	Original ES *
60	Not used	Original SS
56	Saved ESP	
52	Saved EFR	
48	Exception bits	Saved CS
44	Saved EPC	
40	Error code	
36	Not used	Return CS
32	Return EIP	
28	Not used	
24	Not used	
20	Not used	
16	Not used	
12	Version 0.9 saved SS	Version 0.9 saved SP
8	Version 0.9 saved FR	Version 0.9 saved CS
4	Version 0.9 saved IP	Version 0.9 error code
0	Version 0.9 return CS	Version 0.9 return IP

The format of the 32-bit DPMI version 1.0 stack frame is shown below.

Offset	High order word	Low order word
84	PTE value	
80	CR2 value	
76	Not used	Original GS *
72	Not used	Original FS *
68	Not used	Original DS *
64	Not used	Original ES *
60	Not used	Original SS
56	Saved ESP	
52	Saved EFR	
48	Exception bits	Saved CS
44	Saved EPC	
40	Error code	
36	Not used	Return CS
32	Return EIP	
28	Not used	
24	Version 0.9 saved ESP	
20	Version 0.9 saved EFR	
16	Not used	Version 0.9 saved CS
12	Version 0.9 saved EIP	
8	Version 0.9 error code	
4	Not used	Version 0.9 return CS
0	Version 0.9 return EPC	

Emulated Hardware Interrupts

Certain hardware interrupts are emulated using signals. This includes the interrupts for the system clock (IRQ1), the keyboard (IRQ2), and the serial ports (*interrupt* request is assignable). XOS assumes that all vectors used for emulated hardware interrupts have a level of 4. Separate logic emulates the operation of the standard *interrupt* controllers within that single priority level.

Most programs which use the emulated hardware interrupts will be DOS programs using hardware type vectors, which always have a level of 4. If native XOS pro-

grams set up the vectors associated with devices to be XOS type vectors, the level must be set to 4. If this is not done, interrupts may be lost or the process may hang.

When a *signal* or *trap* is granted using an XOS type *interrupt* vector, the process' level is set directly to the level of the vector, except that the process' level is never decreased in granting a *trap*. (It will not be decreased when granting a *signal* because of the operation of the priority system.)

When a *signal* or *trap* is granted using a hardware type *interrupt* vector, only bit 2 of the level of the process is changed. This behavior is summarized in the following table:

Level before signal	Level after signal
0	4
1	5
2	6
3	7

This somewhat unusual behavior is implemented to allow for the combined use of emulated hardware interrupts and XOS type signals. If a DOS program is also using XOS type vectors for signals, it can control the interaction between the emulated hardware interrupts and the XOS type signals by assigning levels to the XOS type vectors that are either below or above 4. If the XOS vectors have levels less than 4, the XOS signals will be held off when DOS interrupts are disabled, but the act of disabling and re-enabling the DOS interrupts will not change the priority level as seen by the XOS vectors. (It must be remembered that not only is only bit 4 changed when the emulated hardware *interrupt* is granted, but also only bit 4 is changed when the emulated hardware interrupt is dismissed using the IRET instruction.) Thus, if the process is at level 2 ,for example, servicing an XOS *signal* and an emulated hardware *interrupt* is granted, the level will be raised to 6. When the emulated hardware *interrupt* is dismissed with an IRET instruction, the level will return to 2, since only bit 2 is changed.

This would be much more straight-forward if it were possible to simply save and restore the level of the process across the emulated hardware *interrupt*, but this cannot be done since there is only one bit available in the FR to indicate the *interrupt* state. Trying to use additional bits is extremely failure prone, since many DOS programs count on the exact behavior of the hardware when processing interrupts.

This somewhat convoluted scheme allows the two environments to coexist in almost all cases.

Mixed-mode Stack Management

The following discussion of mixed mode stack management is complex and can be ignored by most users of XOS. Most user written code executes in a single mode. Code written for the DPMI environment is the major exception to this. The rules for stack management for DPMI which are stated in the DPMI specification also apply under XOS. This section explains the underlying mechanism which XOS uses for mixed mode stack managment.

The term mixed-mode refers to an environment which includes both real mode and protected mode code. It does not refer to environments which include both 16-bit and 32-bit protected mode code. The rules for mixing 16-bit and 32-bit protected mode code are described in detail in the Intel documentation for the 80386/80486 CPU chips. These rules apply equally well for code executing under XOS. In general, XOS does not provide explicit support for mixing 16-bit and 32-bit code except that signals and traps always save the process state in such a way that the orginal state can be exactly restored in all cases. The problems of mapping 32-bit addresses to 16-bit addresses are not directly addressed by XOS.

XOS does fully support the mixing of real mode and protected mode (either 16-bit or 32-bit) code. This creates some special problems relating to management of the program stack. A protected mode stack pointer is not valid in real mode and a real mode stack pointer is not valid in protected mode. While it is always possible to create a protected mode stack pointer that addresses a real mode stack, the reverse is not always possible. In particular, if the protected mode stack is not in the real mode memory segment (and, in general, it will not be) it cannot be accessed at all by real mode code.

This problem is solved by using separate stacks for real mode and protected mode operation. When a switch is made from real mode to protected mode or vice versa, the stack pointer for the old mode is saved on the stack for the new mode and the stack pointer is changed to point to the stack for the new mode. It should be noted that the only way that this mode switch can occur is by the granting of a *signal* or *trap* or by the execution of an svcSchDismiss or svcSchIRet system call. When a *signal* or *trap* is granted, the stack pointer for the old mode is automatically saved on the new stack. It is not saved directly by the svcSchDismiss or svcSchIRet system calls. If a program uses one of these calls to change modes by explicitly constructing a return frame, then the program must separately save the old stack pointer if it is to be restored later.

When an svcSchDismiss or svcSchIRet system call which causes a mode switch is executed, the stack pointer for the old mode is stored internally by XOS. When

switching back to the original mode, it is used to re-establish the stack for that mode. This allows unlimited switching between modes, with the stack growing as needed.

When an XOS process is initially created, it only consists of a protected mode part and thus has only a protected mode stack. If a real mode memory segment is created, a special 4KB stack section is also created at the real mode address 0EC00:0. The initial real mode stack pointer is set to use this stack. If the protected mode part explicitly changes the real mode stack pointer to use a different stack (the usual case), the original stack is not used directly. This special stack section is used when mode switching and there is no user specified stack available. This happens most often when switching to protected mode to execute the user level protected mode code which implements much of the DOS emulator. Note that this stack segment is not used for the DPMI *locked protected mode stack*, but an additional memory page is allocated for this purpose when DPMI is initialized.

Even though XOS does everything it can to make stack switching when changing modes transparent, a user program which includes code executing in both real and protected modes generally must be aware of how the two stacks are being handled and must insure that no conflicts occur. This is especially true for mixed mode programs which make heavy use of signals and traps which cause mode switches. Any time a user program explicitly specifies a stack pointer value for the other mode, extreme care must be taken to insure that no conflicts are created.

Chapter 6

Interprocess Communication

XOS provides three independent mechanisms for communication between processes:

Shared memory

Interprocess messages

Events

These three methods are described in this chapter.

Shared Memory

An XOS process can convert part of its memory to a named shared memory section which can then be accessed by name by other processes. Each process accessing the shared section can map it at its orignal virtual address or at any addres it desires. Access is controlled by a owner/group/world scheme similar to that used with may file systems.

A program running in a DOS environment can also access shared memory. The unused upper memory area is initially mapped to null memory. This can be given up and the address space can then be used to map shared memory. DOS 32-bit programs can freely map shared memory sections by using the standard XOS system calls.

The system calls for creating and accessing shared memory sections are described in chapter 10.

Interprocess Messages

Interprocess messages provide another means for the exchange of a resonable amount of data between processes. Interprocess messages are implemented using a device (IPM) which uses the same system calls as all other devices. It is a datagram device, in that no connections are established to other processes, but each message is addressed independently. It operates much like the UDP network device. Each input or output system call receives or sends one message.

Messages can be addressed to a process (using its process ID) or to a named message destination. A process can set up to receive messages for as many named destinations as needed. Named destinations are global in the system and are generally used by servers to allow access by general clients.

To establish a named destination, a process opens an IPM device specifying the destination name using the normal file name syntax. Thus to estabish the named destination “IPMTEST”, the string “IPM:IPMTEST” would be used for the device name. Using the device name “IPM:” opens an unnamed IPM device which receives messages directed to the process’s process ID. A process can open at most one unnamed IPM device.

When an IPM message is output, it is queued to the destination process. If the process (or named destination) does not exist at the time the output is done, an error is returned. If the destination process terminates without inputting the message, however, the message is lost with no indication to the sender. Thus for most applications, it is desirable to use an application level protocol to acknowlege messages.

The content of IPM messages is not restricted in any way by XOS. Information about the sender of the message is obtained using device parameters rather than a header which is part of the message. Most XOS utilities which use interprocess messages use the first two bytes to specify a message type and sub-type, but this is not necessary, as long as the sender and receiver agree on the format of the message.

Events

Events provide yet another mechanism for communication between processes. An even is a simple data item which can hold a 32-bit value. When the value is 0, the event is considered to be “cleared”. When the value is other than 0, the event is considered to be “set”. Events are organized as event clusters. Each event cluster con-

tains between 1 and 31 events. Event clusters are named with an arbitary text string and events within a cluster are numbered. The name-space for event clusters is local to the process which created the event cluster, but is globalably available on the system. This means that an event is completely identified by a PID, an event cluster name, and an event number.

When a process creates an event cluster it can associate a signal vector which each event in the cluster. A signal will be posted to the process whenever one of the events is "set" (its value changes from 0 to non-0). A process can also wait until one or more of the events in a cluster has been set.

Any process can set (specify a non-0 value for) an event created by any process, but a process can only wait for, be signaled by, or clear (set to 0) events which is has created.

Events are normally used to signal another process that some action needs to be taken. They are often used in conjuction with a shared memory section. An event would be used to indicate to a process that data has been changed in a shared memory section that that process needs to handle.

Chapter 7

System Calls Overview

Each system call is described in a standard format, with no more than one system call per page. The format for each function is specified as a C function prototype declaration. A description of the function is provided, and a description of the value returned. An example is included for reference.

Calling Conventions

All XOS system calls are implemented as procedures which are called using the Pascal calling convention. The system calling convention is identical for virtual and native modes except as noted below. All arguments are first pushed onto the stack, with the first argument being pushed first. All arguments are 32-bit numeric or 64-bit address values. The procedure is then called using a far call instruction. The arguments are removed from the stack by the called procedure. This implies that the exact number of parameters specified must be pushed onto the stack for each system call or unspecified behavior will result.

Memory Addressing

Native mode addresses are always passed as full far addresses, with the selector part of the address pushed onto the stack first as a 32-bit value. Only the low order 16-bits of this value are used. The offset part of the address is next pushed onto the stack as a 32-bit value with all bits being used. Virtual mode addresses are also passed as two 32-bit values but in a different format. The first 32-bit value pushed onto the stack is a dummy value which is not used and which must be 0. The second

32-bit value is the full 32-bit virtual mode far address; the high order 16-bits contain the selector part and the low order 16-bits contain the offset part of the address.

Any native mode address with an offset value of 0 is considered to be a NULL address, even if the selector part is not 0. This convention is used to make the coding of system calls easier in programs which generally use only near pointers to specify addresses. A virtual mode address is considered to be a NULL address only when both the selector and offset parts are 0. A native mode address must not be specified with a NULL selector and a non-NULL offset value, as this will be interpeted as a virtual-mode address.

Return Value

All system calls return a 32-bit value. This value is returned in the EAX register. In virtual mode, the high order 16-bits of the value are also returned in the DX register. No other registers (except the stack pointer to remove arguments from the stack) are changed by a system call. A successful return from a system call always returns a positive value. An error is indicated by a negative return value, with the value being a standard error code (see Chapter 2 for definition of the error codes).

Header files

An XOSSVC.H header file is provided which define C prototypes for the system calls.

Chapter 8

Utility Function System Calls

This chapter describes the utility function system calls. These are the functions which do not naturally fall into any of the other classes of system calls. This includes functions to find and define environment strings, functions which deal with dates and times, functions to access the system's CMOS non-volatile memory, a function to generate standard error messages, and finally, a function which loads loadable kernel extensions (LKEs).

svcSysCmos - CMOS Memory Function

svcSysCmos

CALLING SEQUENCE:

```
XOSSVC svcSysCmos(long address, long data);
```

VALUE RETURNED:

Returns the 8-bit value read or written, zero extended to 32-bits if normal or a negative error code if an error occurred.

DESCRIPTION:

This system call reads or writes a location in the system@146s CMOS memory, the non-volatile configuration memory. The *address* argument specifies the CMOS memory address to access. The *data* argument specifies the function and the data to write if this is a write function. A 32-bit value of -1 specifies a read function. The contents of the CMOS location (8 bits) specified by *address* is zero extended to 32 bits and returned as the value of the system call. If the value of *data* is not -1, the call is a write function. The low 8 bits of the value are written into the CMOS location specified by *address*. The value written, zero extended to 32-bits, is returned as the value of the system call.

EXAMPLES:

svcSysDateTime - Date and Time Functions

svcSysDateTime

CALLING SEQUENCE:

```
XOSSVC svcSysDateTime(long func, void far *data);
```

VALUE RETURNED:

Returns 0 if normal or a negative error code if an error occurred.

HEADER FILE:

These functions are defined in the XOSTIME.H header file.

DESCRIPTION:

This system call implements various functions having to do with obtaining, setting, and converting date and time values. These functions are specified by the value of the argument *func* and are summarized in Table 8.1. The argument *data* is a pointer to a data block, the exact format of which depends on the function.

Table 8.1 - Date/time functions

Name	Value	Description
T_GTXDTTM	1	Get system date and time in XOS format
T_GTDDTTM	2	Get system date and time in DOS format
T_GTXDTTMTZ	3	Get system date, time, and time-zone in XOS format
T_GTDDTTMTZ	4	Get system data, time, and time-zone in DOS format
T_GTPETIM	6	Get process elapsed time
T_GTSETIM	7	Get session elapsed time
T_GTPCTIM	8	Get process CPU time
T_CVD2X	9	Convert DOS to XOS date and time
T_CVX2D	10	Convert date and time from XOS to DOS format
T_GTHRDTTM	11	Get system high-resolution date and time in XOS format
T_GTHRPETIM	12	Get process high-resolution elapsed time in XOS format
T_GTHRSETIM	13	Get session high-resolution elapsed time in XOS format
T_GTHRDTTMTZ	14	Get system high-resolution date, time, and time-zone in XOS format
T_GTRDTTM	15	Get RTC date and time (requires ADMIN privilege)
T_STXDTTM	16	Set system date and time (requires ADMIN privilege)
T_STRDTTM	17	Set RTC date and time (requires ADMIN privilege)
T_GTTZINFO	18	Get time zone information
T_STTZINFO	19	Set time zone information (requires ADMIN privilege)

Following is a detailed description of each of these functions

T_GTXDTTM - Get system date and time in XOS format

This function returns the current system date and time in XOS format. The XOS date/time format (which is specified by the C typedef name time_s) consists of a structure containing two 32 bit values as shown in Table 8.2. The first value gives the time of day in fractional days. This is a 32-bit unsigned binary fraction with the binary point to the left of the left-most (high order) bit. Thus a value of 0x40000000 would represent 6 AM, 0x80000000 would represent noon, 0xC0000000 would represent 6 PM, etc. The second 32-bit value gives the date as the number of days since the beginning of the year 1500.

Table 8.2 - XOS date/time format			
Element name	Offset	Size	Description
time	0	4	Time (fractional days)
date	4	4	Date (days since 1600)

The *data* argument points to a time_s structure which receives the date/time value.

The value returned by this function is only accurate to about 1/50 second. A more accurate value can be obtained by using the T_GTHRDTTM function. Since this function has significantly less overhead, it should be used unless the greater accuracy is necessary.

T_GTXDTTMTZ - Get system date, time, and time-zone in XOS format

This function returns the current system date and time in XOS format. The XOS date/time/time-zone format (which is specified by the C typedef name time_sz) consists of a structure containing two 32 bit values as shown in Table 8.2. The first value gives the time of day in fractional days. This is a 32-bit unsigned binary fraction with the binary point to the left of the left-most (high order) bit. Thus a value of 0x40000000 would represent 6 AM, 0x80000000 would represent noon, 0xC0000000 would represent 6 PM, etc. The second 32-bit value gives the date as the number of days since the beginning of the year 1500.

Table 8.3 - XOS date/time/time-zone format			
Element name	Offset	Size	Description
time	0	4	Time (fractional days)
date	4	4	Date (days since 1600)
tzone	8	2	Offset from GMT (in minutes)
dlst	10	2	Daylight savings time offset (in minutes)

The *data* argument points to a time_s structure which receives the date/time value.

The value returned by this function is only accurate to about 1/50 second. A more accurate value can be obtained by using the T_GTHRDTTMTZ function. Since this function has significantly less overhead, it should be used unless the greater accuracy is necessary.

T_GTDDTTM - Get system date and time in DOS format

This function returns the current system date and time in DOS format. The DOS date/time format consists of a structure (which is specified by the C typedef name time_d) containing the fully broken down date and time values as shown in Table 8.4.

Table 8.4 - DOS date/time format

Element name	Offset	Size	Range	Description
tmx_msec	0	2	0-999	Milliseconds
tmx_sec	2	2	0-59	Seconds
tmx_min	4	2	0-59	Minutes
tmx_hour	6	2	0-23	Hours
tmx_mday	8	2	1-31	Day of the month
tmx_mon	10	2	1-12	Month
tmx_year	12	2	1600-	Year
tmx_wday	14	2	0-6	Day of the week
tmx_yday	16	2	0-365	Day of the year

The *data* argument points to a time_d structure which receives the date/time value.

T_GTDDTTMTZ - Get system date, time, and time-zone in DOS format

This function returns the current system date, time, and time-zone in DOS format. The DOS date/time/time-zone format consists of a structure (which is specified by the C typedef name time_dz) containing the fully broken down date and time values as shown in Table 8.5.

Table 8.5 - DOS date/time/time-zone format

Element name	Offset	Size	Range	Description
tmx_msec	0	2	0-999	Milliseconds
tmx_sec	2	2	0-59	Seconds
tmx_min	4	2	0-59	Minutes
tmx_hour	6	2	0-23	Hours
tmx_mday	8	2	1-31	Day of the month
tmx_mon	10	2	1-12	Month
tmx_year	12	2	1600-	Year
tmx_wday	14	2	0-6	Day of the week
tmx_yday	16	2	0-365	Day of the year
tmx_tzone	18	2	0-1339	Offset from GMT (in minutes)
tmx_dlst	20	2	0-60	Daylight savings time offset (in minutes)0

The *data* argument points to a time_dz structure which receives the date/time value.

T_GTPETIM - Get process elapsed time

This function returns the elapsed time since the process was created. The data argument points to a time_t structure (XOS format date and time) which receives the value, which is stored as an incremental XOS date/time value consisting of days and fractional days.

The value returned by this function is only accurate to about 1/50 second. A more accurate value can be obtained by using the T_GTHRPETIM function. Since this function has significantly less overhead, it should be used unless the greater accuracy is necessary.

T_GTSETIM - Get session elapsed time

This function returns the elapsed time for all processes in the current session since the session was created. The *data* argument points to a time_t structure (XOS format date and time) which receives the value, which is stored as an incremental XOS date/time value consisting of days and fractional days.

The value returned by this function is only accurate to about 1/50 second. A more accurate value can be obtained by using the T_GTHRSETIM function. Since this function has significantly less overhead, it should be used unless the greater accuracy is necessary.

T_GTPCTIM - Get process CPU time

This function returns the total CPU time for the current process. The *data* argument points to a time_t structure (XOS format date and time) which receives the value, which is stored as an incremental XOS date/time value consisting of days and fractional days. In this format, the least significant bit of the fractional days part represents slightly over 20 microsec.

T_CVD2X - Convert DOS to XOS date and time

This function converts a DOS format(time_d) date and time value to an XOS format (time_t) date and time value. The *data* argument points to a time_x structure, which consists of a time_t structure followed immediately by a time_d structure. The time_t structure must contain the XOS format date and time value to convert. The DOS format time_d value is written to the time_d structure.

T_CVX2D - Convert date and time to DOS format

This functions converts an XOS format (time_t) date and time value to a DOS format (time_d) date and time value. The *data* argument points to a time_x structure, which consists of a time_t structure followed immediately by a time_d structure. The time_d structure must contain the DOS format date and time value to convert. The XOS format time_t value is written to the time_t structure.

T_GTHRDTTM - Get system high-resolution date and time in XOS format

This function is the same as the T_GTXTDTM function except that the time is returned accurate to about 30 microseconds instead of 1/50 second.

T_GTHRPETIM - Get process high-resolution elapsed time in XOS format

This function is the same as the T_GTPETIM function except that the time is returned accurate to about 30 microseconds instead of 1/50 second.

T_GTHRSETIM - Get session high-resolution elapsed time in XOS format

This function is the same as the T_GTSETIM function except that the time is returned accurate to about 30 microseconds instead of 1/50 second.

T_GTHRDTTMTZ - Get system high-resolution date, time, and time-zone in XOS format

This function is the same as the T_GTXTDTMTZ function except that the time is returned accurate to about 30 microseconds instead of 1/50 second.

T_GTRTIM - Get RTC date and time (requires ADMIN privilege)

This function gets the date and time from the real-time clock in DOS format (broken down values). The *data* argument points to a time_d structure which receives the value.

T_STXTIM - Set system date and time (requires ADMIN privilege)

This function sets the current system date and time. The *data* argument points to a time_t structure (XOS format date and time) which must contain the new date and time for the system.

T_STRTIM - Set RTC date and time (requires ADMIN privilege)

This function sets the real-time clock (RTC) date and time. The *data* argument points to a time_d structure (DOS format broken down date and time) which must contain the new date and time for the real-time clock. Note that this function does not change the system date and time.

T_GTTZINFO - Get time zone information

This function returns data which defines the system's time-zone. This data is specified in a structure which is described in table 8.6.

specified by the C typedef name time_d) containing the fully broken down date and time values as shown in Table 8.4.

Table 8.6 - Time-zone information

Element name	Offset	Size	Description
tzone	0	4	Offset from GMT (in minutes)
begintime	4	4	Time of day when daylight savings time begines (in fractional days)
beginday	8	4	Day of the year when daylight savings time begins
endtime	12	4	Time of day when daylight savings time ends (in fractional days)
endday	16	4	Day of the year when daylight savings time ends
offset	20	4	Daylight savings time offset (in minutes)

Note that the offset value is valid, even if daylight savings time is not in effect.

The *data* argument points to a structure which receives the time-zone information information.

T_STTZINFO - Set time zone information (requires ADMIN privilege)

This function sets the time-zone data for the system. It uses the same data structure as specified in table 8.6. All values from the structure are used to set the system's values.

svcSysDefEnv - Define Environment String

svcSysDefEnv

CALLING SEQUENCE:

```
XOSSVC svcSysDefEnv(long proc, char far *str, char far *def);
```

VALUE RETURNED:

The value returned is 0 if normal or a negative error code if an error occurred.

DESCRIPTION:

The environment string whose name is specified by the argument *str* (a null terminated string up to 128 bytes in length) is defined to have the value specifed by the argument *def* (a null terminated string up to 512 bytes in length). The name of an environment is not case sensitive. XOS internally converts all name characters to upper case. The definition is case sensitive. Characters are stored exactly as given. Any character except for null can be included in the definition string.

The environment string is defined for the process specified by the argument *proc*. The meaning of the *proc* argument is specified by bits 14 and 15 as summarized below.

Name	Bit	Description
FES$SESSION	15	Set to specify level relative to session process
FES$PROC	14	Set to specify level relative to current process
FES$SYSTEM	15, 14	Set to specify system level

When both of these bits are 0, the remainder of the 32-bit value specifies a process ID. Bits 31 to 16 specify the sequence number part of the ID and bits 13 to 0 specify the process number part of the ID. A value of zero for both the sequence number and process number parts specifies the process issuing the system call.

If bits 14 and 15 are both 1, the remaining bits are ignored. This indicates a system level definition, which is not directly associated with any process, but with the system as a whole. The system level environment strings are copied as the initial environment strings when a new process is created by INIT.

If bit 15 is 1 and bit 14 is 0, then bits 7 to 0 specify the offset below the session process. A value of 0 specifies the session process itself. A value of 1 specifies the child of the session process which is in the direct line of inheritance to the process issuing the system call. A value of 2 specifies that process' child, etc. This format can only specify processes down to the process issuing the system call.

If bit 15 is 0 and bit 14 is 1, then bits 7 to 0 specify the offset above the process issuing the system call. A value of 0 specifies the process itself, a value of 1 specifies its parent, etc. This format can only specify processes up to and including the session process.

A general discussion of XOS environment strings is given in Chapter 2.

EXAMPLES:

svcSysErrMsg - Get Error Message

svcSysErrMsg

CALLING SEQUENCE:

```
XOSSVC svcSysErrMsg(long code, long format, void far *buffer);
```

VALUE RETURNED:

Will return 0 or a negative error code if an error occurred.

DESCRIPTION:

This system call generates a standard error message given an error code value. The argument *code* specifies the error code for which the message should be generated. The argument *str* specifies the elements to include in the generated message. It is bit encoded as follows:

Name	Bit	Description
EM$TEXT	0	Include error text
EM$CODE	1	Include error code in braces
	2-31	Not used

The argument *buffer* specifies the address of the buffer to receive the message. This buffer must be in writable memory and be at least 80 bytes long.

EXAMPLES:

svcSysFindEnv - Find Environment String

svcSysFindEnv

CALLING SEQUENCE:

```
XOSSVC svcSysFindEnv(long proc, char far *search, char far *name, char far
        *buffer, long length, long far *skip);
```

VALUE RETURNED:

The value returned is the length of the string stored in the buffer specified by the *buffer* argument if normal or a negative error code if an error occurred.

DESCRIPTION:

This system call searches for an environment string. The name of the environment string to be found is specified as a string whose address is given by the *search* argument (a null terminated string). It may contain wild-card characters . The number of names to skip before checking for a match is specified by the 32-bit value whose address is given by the *skip* argument. This value is updated when a match is found so that another call will find the next matching name.

Environment strings can be defined for every process in the system and for the system itself. Generally, a process only searches its own environment strings; however, it can search those of any other process in the system, privileges permitting.

The *proc* argument specifies the process level to search. The meaning of this argument is specified by bits 14 and 15 as summarized below.

Name	Bit	Description
FES$SESSION	15	Set to specify level relative to session process
FES$PROC	14	Set to specify level relative to current process
FES$SYSTEM	15, 14	Set to specify system level

When both of these bits are 0, the remainder of the 32-bit value specifies a process ID. Bits 31 to 16 specify the sequence number part of the ID and bits 13 to 0 specify the process number part of the ID. A value of zero for both the sequence number and process number parts specifies the process issuing the system call.

If bits 14 and 15 are both 1, the remaining bits are ignored. This indicates system level environment strings, which are not directly associated with any process, but with the system as a whole. The system level environment strings are copied as the initial environment strings when a new process is created by INIT.

If bit 15 is 1 and bit 14 is 0, then bits 7 to 0 specify the offset below the session process. A value of 0 specifies the session process itself. A value of 1 specifies the child of the session process which is in the direct line of inheritance to the process issuing the system call. A value of 2 specifies that process', child, etc. This format can only specify processes down to the process issuing the system call.

If bit 15 is 0 and bit 14 is 1, then bits 7 to 0 specify the offset above the process issuing the system call. A value of 0 specifies the process itself, a value of 1 specifies its parent, etc. This format can only specify processes up to and including the session process.

The matching name found is returned in the buffer whose address is given by the *name* argument. This buffer must be at least 129 bytes long to allow for a maximum length environment string name of 128 bytes plus a final null character. The definition of the environment string is returned in the buffer whose address is given by the *buffer* argument and whose length is given by the *length* argument. If the definition will not fit in the buffer, the final two characters stored are RUBOUT (0xFF) followed by a null (0x00). Otherwise the definition is stored in the buffer followed by a null. A buffer length of at least 64 bytes is recommended; a buffer length of 513 bytes will contain the longest possible environment string plus a terminating null character.

A general discussion of XOS environment strings is given in Chapter 2.

EXAMPLES:

svcSysGetEnv - Get All Environment Strings

svcSysFindEnv

CALLING SEQUENCE:

```
XOSSVC svcSysGetEnv(long proc, char far *buffer, long length);
```

VALUE RETURNED:

The value returned is the length of the string stored in the buffer specified by the *buffer* argument if normal or a negative error code if an error occurred.

DESCRIPTION:

This system call returns all environment string defined for a process. Each string is stored as a null terminated string in the format:

envname=envstring

The next string starts in the byte immediately following the terminating null.

The process is specified by the proc argument, which has the same meaning as for the svcSysFindEnv and svcSysDefEnv system calls.

This system call is intended mainly to be used when setting up a DOS enviroment, but it can be used by any user program if desired.

svcSysLoadLke - Load LKE

svcSysLoadLke

CALLING SEQUENCE:

```
XOSSVC svcSysLoadLke(struct lkedata *lkedata);
```

VALUE RETURNED:

Returns 0 if normal or a negative error code if an error occurred.

DESCRIPTION

This system call loads an LKE (Loadable Kernel Extension). The sequence involved and the argument block are somewhat complex and are explained in detail in Chapter 20. Generally, a user program should not attempt to use this function directly. It is intended exclusively for the use of the LKELOAD utility. The details of the operation of this system call may change significantly in future versions of XOS.

EXAMPLES:

svcSysLog - Place Entry in System Log File

svcSysLog

CALLING SEQUENCE:

```
XOSSVC svcSysLog(char *msg, long length);
```

VALUE RETURNED:

Returns 0 if normal or a negative error code if an error occurred.

DESCRIPTION:

The text string pointed to by msg containing length characters is written to the system log file. The first two characters in the message are not used, but must be present.

EXAMPLES:

Chapter 9

Scheduler System Calls

The scheduler system calls are those which control the creation, termination, or scheduling of user processes. They are described in this chapter.

svcSchAlarm - Alarm Functions

svcSchAlarm

CALLING SEQUENCE:

XOSSVC svcSchAlarm(long func, long handle, long vector, time_t datetime);

VALUE RETURNED:

Returns an alarm handle or 0 (depending on the function) if normal or a negative error code if an error occurred.

DESCRIPTION:

This system call creates or removes alarms. An alarm requests a signal after a specified interval has passed or at a specified time. The possible function argument values are summarized in Table 9.1.

Table 9.1 - Alarm interrupt function values

Name	Value	Function
TIF_REMOVE	1	Remove alarm
TIF_DATETIME	2	Request alarm interrupt at specified date and time
TIF_INTERVAL	3	Request alarm interrupt after specified interval
TIF_REPEAT	4	Request clock interrupt at specified interval (repeated)

The *vector* argument specifies the vector to be used for the alarm signal. It **MUST** be set up (with the svcSetVector system call) before this system call is issued.

The TIF_DISABLE function removes an alarm which was previously created. The alarm to be removed is specified by the value of the *handle* argument. This can be either a one-short alarm which has not gone off yet (created with TIF_DATETIME or TIF_INTERVAL) or a repeated alarm (created with TIF_REPEAT).

The TIF_DATETIME function sets an alarm to request a signal at the specified date and time. If the *handle* argument is 0, a new alarm is created. If it is non-zero, it must be a value returned by this system call for an alarm which is still active. In this case the alarm previously associated with the handle is canceled. After the signal is requested, the alarm will be

removed. The date and time are specified in the standard XOS format (days since 1500 and fractional days).

The TIF_INTERVAL function sets an alarm to request a signal at the end of a specified interval. If the *handle* argument is 0, a new alarm is created. If it is non-zero, it must be a value returned by this system call for an alarm which is still active. In this case the alarm previously associated with the handle is canceled. After the signal is requested, the alarm will be removed. The interval is specified in fractional days in the time part of the *datetime* argument. The value of the date part must be 0.

The TIF_REPEAT function is the same as the TIF_INTERVAL function except that the alarm created is not removed after a signal is requested. It continues to request signals at the interval specified until it is removed with the TIF_REMOVE function.

EXAMPLES:

svcSchClrEvent - Clear Event(s)

svcSchClrEvent

CALLING SEQUENCE:

XOSSVC svcSchClrEvent(char *cluster, long events);

VALUE RETURNED:

Returns the previous event state if normal or a negative error code if an error occurred.

DESCRIPTION:

This function clears the event(s) specified by the argument *events* in the cluster specified by the argument *cluster* for the calling process. Bit 0 in the *events* value is ignored. Each other bit corresponds to an event in the specified event cluster. Bit 0 corresponds to event 0, bit 1 to event 1, etc. If the cluster contains less than 31 events, the extra bits are ignored. If bit *n* is 1, event *n* is cleared. If bit *n* is 0, the state of event *n* is not changed. The value returned is the state of the events in the cluster before any were cleared. If a bit is 1, the corresponding event was set. An event is said to be set when it has any value other than 0.

EXAMPLES:

svcSchCtlCDone - Report ctl-C Processing Done

svcSchCtlCDone

CALLING SEQUENCE:

XOSSVC svcSchCtlCDone(void);

VALUE RETURNED:

The value returned is always 0.

DESCRIPTION:

This system call is used to report to the system that a process has finished processing a ctrl-C software interrupt. This system call must be executed before dismissing a ctrl-C interrupt or output to the sessions controlling terminal will probably hang. The main function performed by this system call is to synchronize the display of terminal output from child processes. Whenever a process receives a ctrl-C interrupt, all output to the process' controlling terminal is blocked. This is done to allow the process to handle the ctrl-C interrupt, including possibly carrying on a dialog with the user, without interference from terminal output generated by a child process which may still be executing. This system call re-enables output to the session's controlling terminal. It should not be executed unless a ctrl-C interrupt is being processed. Doing so will produce undefined results which are likely to include hanging or otherwise undesirably affecting terminal output.

EXAMPLES:

svcSchDismiss - Dismiss Signal

svcSchDismiss

CALLING SEQUENCE:

XOSSVC svcSchDismiss(void);

VALUE RETURNED:

Since this system call never returns directly to the caller, no value can be returned.

DESCRIPTION:

This system call dismisses a signal. It is the normal way of terminating a signal routine entered with an XOS type vector as described in chapter 5 on page 30. ny signal data on the user stack is discarded and the state of the process is restored from the user stack. If this system call is executed in virtual-86 mode, the contents of the segment registers are also restored from the user stack. Otherwise, the contents of the registers are not affected. It is the responsibility of the user program to save registers across signal routines.

Certain bits in the EFR value on the user's stack are used to control the operation of this system call. these bits are normally set to 0 when the EFR value is stored when a signal is granted. hey can be set directly by the user program if the indicated behavior is described. These bits are listed below. Any bits not listed here are simply restored to the EFR. The listed bits are all bits which have not defined use in the EFR.

Bit	Meaning
26	Always restore SS:ESP and segment registers from the user stack if set
25	Always restore SS:ESP from the user stack if set
24	Do not change signal level if set
23	Restore interrupt level using DOS conventions

EXAMPLES:

svcSchExit - Terminate Process

svcSchExit

CALLING SEQUENCE:

```
XOSSVC svcSchExit(long status);
```

VALUE RETURNED:

Since this call causes the process to terminate, no value can be returned.

DESCRIPTION:

The svcSchExit system call terminates a process and specifies a termination status which is made available to the process' parent (see "child died" signal in Chapter 4). This function is the usual way that a process is terminated. By convention, a termination status of 0 means a normal termination; a non-zero value means some kind of problem occurred during execution of the process. Only the low order 24-bits of the termination status value are used. The XOS kernel does not use the termination status value; it is simply passed to the parent process. The standard command processor (SHELL) does use it as described above.

Note that when XOS terminates a process, it also terminates all of its child processes. An XOS process cannot exist without a parent process, insuring orderly termination of all created sub-processes when a process is terminated.

EXAMPLES:

svcSchGetVector - Get Signal Vector

svcSchGetVector

CALLING SEQUENCE:

```
XOSSVC svcSchGetVector(long vector, void far **address);
```

VALUE RETURNED:

Returns the vector type and signal level associated with the vector specified (always a positive number) if normal or a negative error code if an error occurred.

DESCRIPTION:

This system call gets the current contents of the signal vector specified by the *vector* argument. The vector contents consist of a vector type, signal level, and a signal routine address. The vector type and signal level are returned as the value of the system call as shown below. The address of the signal routine for the vector is stored in the location specified by the argument *address*.

Bits	Meaning			
31-16	Always 0			
15-8	Vector type	Value	Name	Description
		0	VT_IDLE	Idle
		1	VT_XOSS	XOS signal vector
		2	VT_XOST	XOS trap vector
		3	VT_HWT16	16-bit hardware trap vector
		4	VT_HWT32	32-bit hardware trap vector
		5	VT_HWS16	16-bit hardware signal vector
		6	VT_HWS32	32-bit hardware signal vector
		7	VT_DPMI16O	DPMI v0.9 16-bit CPU exception vector
		8	VT_DPMI32O	DPMI v0.9 32-bit CPU exception vector
		9	VT_DPMI16N	DPMI v1.0 16-bit CPU exception vector
		10	VT_DPMI32N	DPMI v1.0 32-bit CPU exception vector
7-0	Signal level	Value between 1 and 7		

EXAMPLES:

svcSchIntrProc - Interrupt Child Process

svcSchIntrProc

CALLING SEQUENCE:

XOSSVC svcSchIntrProc(long pid, long function, long status);

VALUE RETURNED:

Returns a value of 0 if the child process was interrupted, 1 if the child process was killed, or a negative error code if an error occurred.

DESCRIPTION:

This system call performs several functions related to managing the execution of child processes, depending on the value of the *function* argument. The following describes the action for each legal value of *function*.

function = 1 - Interrupt or kill normal process

This functions causes a ctrl-C interrupt for the process specified by the *pid* argument or, if the process has not been set up to receive ctrl-C interrupts, terminates the process. The specified process must be a child of the process issuing the call.

If the child process has set up the ctrl-C software interrupt, that interrupt is requested. It may or may not occur immediately, depending on the current state of the child's software interrupt system. The status argument is ignored in this case. If the child process has not set up the ctrl-C software interrupt, it is terminated with the value of the *status* argument as its termination status.

When ctrl-C or BREAK is typed on the keyboard controlling a session, the parent process for the session, which is usually a command shell, receives a ctrl-C software interrupt if it has set the ctrl-C interrupt vector (this is the usual case). If it has not set it up, the entire session (the parent process for the session and all of its child processes) is terminated. If the parent process for the session is interrupted, the kernel does nothing further; it is the responsibility of the parent process to interrupt or terminate any child processes it might have. This system call provides the mechanism for doing this. It basically passes the ctrl-C interrupt one level down the process tree. A process receiving a ctrl-C interrupt will normally issue a svcSchIntrProc system call for each of its child processes. The child process is then interrupted or terminated, just as if it had been the parent process for

the group and had received the ctrl-C interrupt directly. This, combined with the use of the svcSchCtlCDone system call, allows for the orderly termination of all processes in a session, without race conditions, when the user types ctrl-C.

function = 2 - Interrupt or kill halted process

The action of this function is almost identical to that of function1 except that the process must have been previously halted by issuing function 3 (see below). The action of this function is undefined if the child process has not been halted.

function = 3 - Halt process

This function halts execution of the specified process, which must be a child process of the process issuing the system call. The child process is placed in the HALTED state and will not execute further, including not responding to software interrupts, until resumed by the parent process (see description of function 4, below).

function = 4 - Resume process

This function resumes execution of the specified process, which must be a child process of the process issuing the system call and which must have been previously halted by issuing function 3 (see above). The action of this function is undefined if the child process has not been halted.

Functions 2, 3, and 4 are used by the command shell when executing a batch file to implement the DOS-style *Terminate batch file* question. All child processes are halted using function 3 before the question is displayed so that any output from the child processes will not appear until the user responds. This is necessary, since XOS, unlike DOS, is a true multitasking system, with the parent and its child processes executing concurrently.

EXAMPLES:

svcSchIRet - Return From Interrupt

svcSchIRet

CALLING SEQUENCE:

XOSSVC svcSvcIRet(void);

VALUE RETURNED:

Since this system call never returns directly to the caller, no value can be returned.

DESCRIPTION:

This system call emulates the operation of the hardware IRET instruction except that it correctly handles transitions between real mode (really V86 mode) and protected mode. This call always assumes a 32-bit stack frame, even when executed from real mode (V86 mode) or from a 16-bit protected mode segment. The signal level is always restored using the DOS conventions. The svcSchDismiss system call is used to dismiss a signal using the standard conventions.

Certain bits in the EFR value on the user's stack are used to control the operation of this system call to provide additional features beyond simple emulation of the IRET instruction. These bits are normally set to 0 when the EFR value is stored when a signal is granted. They can be set directly by the user program if the indicated behavior is described. These bits are listed below. Any bits not listed here are simply restored to the EFR. The listed bits are all bits which have not defined use in the EFR.

Bit	Meaning
26	Always restore SS:ESP and segment registers from the user stack if set
25	Always restore SS:ESP from the user stack if set
24	Do not change signal level if set

EXAMPLES:

svcSchKill - Terminate Any Process

svcSchKill

CALLING SEQUENCE:

XOSSVC svcSchKill(long pid, long status);

VALUE RETURNED:

Returns 0 if normal or a negative error code if an error occurred. If a process uses this call to terminate itself, this call does not return and thus no value is returned.

DESCRIPTION:

This function terminates any process and specifies a termination status which is made available to the process' parent (see "child died" signal). Any process may terminate itself (specified by a *pid* of 0) or any of its child processes. A process running with the KILL privilege can terminate any process in the system. The operation of this system call is identical to that of the svcSchExit call except that the process to terminate is also specified.

EXAMPLES:

svcSchMakEvent - Make Event Cluster

svcSchMakEvent

CALLING SEQUENCE:

```
XOSSVC svcSchMakEvent(char *cluster, long size, long vector);
```

VALUE RETURNED:

Returns 0 if normal or a negative error code if an error occurred.

DESCRIPTION:

This system call creates or gives up an event cluster. The name of the cluster is specified in the *cluster* argument. The size of the cluster to create is specified in the *size* argument. It must be between 1 and 31, inclusive. A value of 0 indicates that the specified cluster is to be given up. If a cluster is being created, it must not already exist. If it is being given up, it must exist.

The *vector* argument specifies the base signal vector for a cluster being created. When an event is set, a signal is requested on the interrupt whose number is the sum of the base vector number specified here and the event number. A *vector* value of -1 indicates that no signals are to be requested for the event cluster.

EXAMPLES:

svcSchRelEvent - Release Event

svcSchRelEvent

CALLING SEQUENCE:

```
XOSSVC svcSchRelEvent(char *cluster, long events);
```

VALUE RETURNED:

Returns a mask of the remaining reserved events in the cluster if normal or a negative error code if an error occurred.

DESCRIPTION:

This system call releases events which were previously reserved by the svcSchResEvent system call. The cluster containing the events is specified by the *cluster* argument and the events are specified by the bits set in the *events* argument. Each event corresponding to a bit set in the *events* argument is released if it was reserved. It is not an error to specify an event which was not reserved. If bit 31 of the *events* argument is set, the event cluster is given up if there are no remaining reserved events.

Note that being reserved or not reserved is an attribute of an event which is completely separate from its value. The intent of this mechanism is to provide a way to allow independent parts of a program to share the a single event cluster.

EXAMPLES:

svcSchResEvent - Reserve Event

svcSchResEvent

CALLING SEQUENCE:

XOSSVC svcSchResEvent(char *cluster, long event);

VALUE RETURNED:

Returns the number of the event reserved if normal or a negative error code if an error occurred.

DESCRIPTION:

This system call reserves an event in the specified cluster. If the *event* argument is not -1, the specified event is reserved if it is not already reserved. If it is reserved, an ER_EVRES error is returned. If the event argument is -1, the lowest number event in the cluster which is not reserved is reserved. If there are no non-reserved events in the cluster, an ER_EVRES error is returned.

Note that being reserved or not reserved is an attribute of an event which is completely separate from its value. The intent of this mechanism is to provide a way to allow independent parts of a program to share the a single event cluster.

EXAMPLES:

svcSchSetEvent - Set Event(s)

svcSchSetEvent

CALLING SEQUENCE:

```
XOSSVC svcSchSetEvent(char *cluster, long event, long value, long pid);
```

VALUE RETURNED:

Returns 0 if normal or a negative error code if an error occurred.

DESCRIPTION:

Sets the event specified in the argument *event* in the cluster specified by the argument *cluster* for the process specified by the argument *pid*.

The event cluster must have been previously created with the svcSchMakEvent system call. The event number specified must be less than the number of events in the cluster. The value specified must be a non-zero positive number. If a signal vector was associated the cluster when it was created and the event being set was previously cleared, a signal will be requested for the process specified.

This system call is intended to provide a simple means of informing a process that some event of interest has occurred and specifying a single integer value describing the event. If more information needs to be passed to another process, the Interprocess Message device (IPM) should be used instead. This call, however, has significantly less overhead and should be used where possible.

EXAMPLES:

svcSchSetLevel - Set Signal Level

svcSchSetLevel

CALLING SEQUENCE:

XOSSVC svcSchSetLevel(long level);

VALUE RETURNED:

Returns the level level in effect when the call was issued (which is positive) if normal or a negative error code if an error occurred unless bit 4 in *level* is set, in which case EAX is unchanged.

DESCRIPTION:

This system call sets the current signal level for the current process and returns the previous signal level. The value given in the argument *level* specifies the new signal level as follows:

Bit(s)	Meaning
31-8	Not used, ignored
7	Not used, must be 0
6-5	00 = Set level from bits 3-0
	01 = OR bits 3-0 to level
	10 = AND complement of bits 3-0 to level
	11 = Do not change level
4	0 = Return previous value
	1 = Do not change EAX
3-0	New level value

EXAMPLES:

svcSchSetVector - Set Signal Vector

svcSchSetVector

CALLING SEQUENCE:

```
XOSSVC svcSchSetVector(long vector, long type, void *address);
```

VALUE RETURNED:

Returns 0 if normal or a negative error code if an error occurred.

DESCRIPTION:

This system call sets the vector type, signal level, and address for a signal vector. The argument *vector* specifies the vector to set. This value must be between 0 and 255. All vector use is controlled by the program; consequently, if a vector number has not previously been used, it is available. The indicated signal vector is set to the values given by the arguments *type* and *address*.

The argument *type* specifies the vector type and signal level to associate with the vector as described below.

Bits	Meaning			
31-16	Not used, must be 0			
15-8	Vector type	Value	Name	Description
		0	VT_IDLE	Idle
		1	VT_XOSS	XOS signal vector
		2	VT_XOST	XOS trap vector
		3	VT_HWT16	16-bit hardware trap vector
		4	VT_HWT32	32-bit hardware trap vector
		5	VT_HWS16	16-bit hardware signal vector
		6	VT_HWS32	32-bit hardware signal vector
		7	VT_DPMI16O	DPMI v0.9 16-bit CPU exception vector
		8	VT_DPMI32O	DPMI v0.9 32-bit CPU exception vector
		9	VT_DPMI16N	DPMI v1.0 16-bit CPU exception vector
		10	VT_DPMI32N	DPMI v1.0 32-bit CPU exception vector
7-0	Signal level	Value between 1 and 7		

The argument *address* specifies the address of the signal routine for the vector.

EXAMPLES:

svcSchSpawn - Create Child Process

svcSchSpawn

CALLING SEQUENCE:

XOSSVC svcSchSpawn(SAB **sab*);

VALUE RETURNED:

Value returned is 0 if operation started succesfully or a negative error code if an error occured. To verify complete success, caller must check both the returned value and the value stored in sab_error (which will be 0 if no errors).

DESCRIPTION:

This system call creates a child process. The initial memory for the new process is taken from memory belonging to the processing executing the system call. The single argument, sab, provides the address of a "spawn argument block" (SAB) which has the following format:

Name	Size	Description
sab_func	2	Function
sab_status	2	Returned status
sab_error	4	Returned error code
sab_pid	4	Returned PID
sab_type	1	Returned process type
	3	Not used, must be 0
sab_vector	2	Vector number
sab_numseg	1	Number of segments to give to new process
	1	Not used, must be 0
sab_option	4	Option bits and event number
sab_name	8	Address of name for new process
sab_EIP	4	Initial EIP value for new process
sab_CS	4	Initial CS value for new process
sab_ESP	4	Initial ESP value for new process
sab_SS	4	Initial SS value for new process
sab_parm	8	Address of parameter list
sab_segsel	8 * n	Selectors for segments to give to new process

The high order byte of the sab_func value is bit encoded. Currently only the high-order bit (0x8000) is used. When set it indicates that the system should not return until the operation is complete. Otherwise, it returns as soon as the operation has been started. The low order byte specifies the function. Currently the only function implemented is to create a child process which is indicated with a value of 0x01.

The create child function is complete when the child process has been created and scheded to execute.

The sab_status, sab_error, and sab_vector values are the same as are described for the qab_status, qab_error, and sab_vector values for the svcIoQueue system call.

When the operation is complete, the process ID (PID) for the child process is stored in sab_pid and the process type is stored in sab_type. The process type values are as follows:

Name	Value	Description
IT_XOS32	1	XOS 32-bit protected mode process
IT_XOS16	2	XOS 16-bit protected mode process
IT_XOSV86	3	XOS 16-bit “real” mode process
IT_DOSEXE	8	DOS process loaded from .EXE file
IT_DOSCOM	9	DOS process loaded from .COM file
IT_BATCH	15	Batch process

The sab_option value contains option bits in the 3 high order bytes and an event number in the low order byte. The option bits are defined below.

Name	Value	Description
R$SAMEPROC	0x80000000	Use same process
R$SESSION	0x04000000	Create new session for child process
R$CPYENV	0x00800000	Copy current enviroment to new process
R$ACSENV	0x00400000	Allow new process to access this process’s enviroment
R$CHGENV	0x00200000	Allow new process to change this process’s enviroment
R$CPYPTH	0x00040000	Copy default paths to new process
R$CHGPTH	0x00020000	Allow new process to change this process’s default

The event number specifies the event in the ^XOS^PROC event cluster for the process issueing this system call which is to be set when the child process terminates. A value of 0xFF indicates that no event is to be set..

The sab_name field specifies the address of a string which gives the initial name for the child process.

The sab_EIP, sab_CS, sab_ESP, and sab_SS fields specify the initial values for the CS:EIP and SS:ESP for the child process.

The sab_parm field specifies the address of a paremeter list. This list can contain any of the IOPAR_RUNxxxx parameters except IOPAR_RUNDEVLIST.

The sab_segsel field contains a table of segment selector pairs. It contains one pair for each segment which is to be transfered to the child process. The first segment of each pair specifies the selector for the segment in the caller's address space. The second segment specifies the selector to give the segment in the child's address space.

This system call is intended mainly for use by the user mode system routine which implements the svcIoRun system call. While it can be executed directly by protected mode user code, this is usually not useful. It cannot be executed from virtual-86 mode.

svcSchSuspend - Suspend Process

svcSchSuspend

CALLING SEQUENCE:

```
XOSSVC svcSchSuspend(long far *flag, long time);
```

VALUE RETURNED:

The value returned is 0 if normal or a negative error code if an error occurred.

DESCRIPTION:

If the address of the *flag* argument is not NULL and the contents of the longword (32-bits) pointed to by the *flag* argument is 0, no action is taken, and control is immediately returned to the caller. Otherwise, the current process is suspended for the length of time, in fractional days, specified in the *time* argument. The maximum allowable value is just less than 12 hours (a value of 0x7FFFFFFF). If the *time* argument is -1, the process is suspended forever. If it is 0, control returns immediately.

Signals will be granted while the process is suspended, provided, of course, that the process state allows the signal. When the signal is dismissed, the svcSchSuspend call resumes waiting for the time interval to expire. The time elapsed while executing the signal routine is subtracted for the time remaining to wait. The signal routine can terminate the wait immediately, setting the longword pointed to by *the flag* argument to 0. Changing the time value during execution of the interrupt routine, however, has no effect.

The flag word is intended to be used to prevent race conditions that could occur if the suspend function is used to wait for an expected software interrupt and the interrupt occurs after the decision has been made to suspend but before the suspend function is executed. The proper sequence is to set the flag word to a non-zero value, check to see if the interrupt is still expected, and if so, execute the suspend function. The software interrupt routine should zero the flag word. Thus, if the interrupt occurs after the test but before the suspend function, the flag word will be 0 and the suspend function will return immediately. If this interlock is not needed, a NULL address may be passed to the suspend function, this checking will not be done, and the process will always be suspended.

EXAMPLES:

svcSchWaitProc - Wait for Process to Terminate

svcSchWaitProc

CALLING SEQUENCE:

```
XOSSVC svcSchWaitProc(long pid, long timeout);
```

VALUE RETURNED:

Returns 0 if normal or a negative error code if an error occurred. The normal return is given if the process does not exist.

DESCRIPTION:

Waits for the process specified by the argument *pid* to terminate. In order to wait for a process to terminate, a process must have the privileges needed to terminate that process. The *timeout* argument specifies the maximum time to wait, in fractional days. The maximum value is just less than 12 hours (a value of 0x7FFFFFFF). This system call will wait forever if *timeout* is set to -1. It will return immediately if it is set to 0 (which is generally not useful). If the specified process does not exist in the system when this call is executed, which will be the case if it has already terminated, an ER_BDPID error is returned.

EXAMPLES:

svcSchWaitMEvent - Wait for Multiple Events

svvSchWaitMEvent

CALLING SEQUENCE:

```
XOSSVC svcSchWaitMEvent(char *cluster, long events, long timeout);
```

VALUE RETURNED:

Value returned is the event state mask if normal or a negative error code if an error occurred.

DESCRIPTION:

This system call waits until one or more events in a cluster are set. The cluster is specified by the *cluster* argument. The events to wait for are specified by bits 30-0 of the *events* argument. Each bit which is set indicates that the corresponding event is to be waited for. If bit 31 is 0, the wait will terminate when any of the indicated events is set. If bit 31 is 1, the wait will terminate when all of the indicated events are set. The *timeout* argument specifies the maximum time to wait in fractional days. A value of 0xFFFFFFFF means to wait forever and a value of 0 means to return immediately. Note that returning because of the timeout value is not considered an error. The caller must check the returned state mask to determine if any of the desired events were really set or if a timeout occurred. A set bit in the returned value indicates that the corresponding event is set.

EXAMPLES:

svcSchWaitSEvent - Wait for Single Event

svcSchWaitSEvent

CALLING SEQUENCE:

```
XOSSVC svcSchWaitSEvent(char *cluster, long event, long timeout);
```

VALUE RETURNED:

Value returned is the value of the event if normal or a negative error code if an error occurred.

DESCRIPTION:

This system call waits until a single event in a cluster are set. The cluster is specified by the *cluster* argument. The event to wait for is specified by the *event* argument. The event is specified as a numeric value between 0 and the maximum event number for the cluster, not as a bit mask. The *timeout* argument specifies the maximum time to wait in fractional days. A value of 0xFFFFFFFF means to wait forever and a value of 0 means to return immediately. Note that returning because of the timeout value is not considered an error. The caller must check the returned value to determine if the desired event was really set or if a timeout occurred. A non-zero returned value indicates that the event is set.

Note that using this system call is the only way that the value of an event flag can be obtained. If the intent of the call is to simply obtain the current value of the event without waiting, the *timeout* argument should be 0.

EXAMPLES:

Chapter 10

Memory System Calls

This chapter describes the memory system calls. The memory system calls create and destroy memory segments, allocate and deallocate memory within segments, manage shared memory sections, allocate and deallocate linear memory blocks not associated with segments, allocate and deallocate segment descriptors, read and write segment descriptors, and obtain information about memory allocation. For a general description of the XOS memory architecture, refer to Chapter 2.

The memory system calls fall into two general groups: those that deal with segments, memory sections, and pages; and those that deal directly with segment selectors and the linear address space. The latter group are implemented mainly to support the DPMI routines and generally should not be used by native XOS programs. These functions may not be supported in future versions of XOS.

svcMemBlkAlloc - Allocate Linear Memory Block

svcMemBlkAlloc

CALLING SEQUENCE:

XOSSVC svcMemBlkAlloc(long lapage, long size, long bits, long client);

VALUE RETURNED:

This call returns the page number of the first linear address page allocated if normal or a negative error code if an error occurred.

DESCRIPTION:

This system call allocates a block of consecutive 4096 byte pages in the process' linear address space. It can only be issued from protected mode. It will fail (with no memory allocated) if the requested number of pages cannot be allocated in a single contiguous block. The *lapage* argument specifies the page number of the first linear address page to allocate. A value of 0 means to allocate the pages at any available page. The *size* argument specifies the number of pages to allocate. The *bits* argument specifies the attributes of the pages allocated as follows:

Bit	Meaning
0	Must be 1
1	Set if pages are to be writable
2	Must be 0
3	Set if pages are to be virtual
4-7	Must be 0

The *client* argument specifies the DPMI client number to be associated with the memory block. It must have a value between 1 and 63.

This system call is intended primarily for the use of the DPMI routines, but it can be used by any user code operating in protected mode.

EXAMPLES:

svcMemBlkChange - Change Size of Linear Memory Block

svcMemBlkChange

CALLING SEQUENCE:

XOSSVC svcMemBlkChange (long lapage, long size, long bits, long client);

VALUE RETURNED:

This call returns the page number of the first linear address page in the block or the number of pages in the block (if *size* = -1) if normal or a negative error code if an error occurred. Note that the number of first page of the block may change as a result of this call.

DESCRIPTION:

This system call changes the size of a linear address space memory block. It can only be issued from protected mode. The *lapage* argument specifies the page number of the first linear address page in the block. The size argument specifies the new size for the block, in pages. A value of 0 means to completely deallocate the block. A value of -1 means to return the current size of the block in pages. The *bits* argument specifies the attributes for any new pages allocated as follows:

Bit	Meaning
0	Must be 1
1	Set if pages are to be writable
2	Must be 0
3	Set if page is to be virtual
4-7	Must be 0

This value is ignored if no new pages are allocated by the call. The *client ar*gument specifies the DPMI client number associated with the memory block. It must be a number between 1 and 63 and must match the DPMI client number specified when the block was initially allocated. If it does not match, an ER_NOMEM error is returned.

This system call is intended primarily for the use of the DPMI routines, but it can be used by any user code operating in protected mode.

EXAMPLES

svcMemBlkFree - Give up All Linear Memory Blocks

svcMemBlkFree

CALLING SEQUENCE:

XOSSVC svcMemBlkFree(long client);

VALUE RETURNED:

This call returns the number of blocks given up if normal or a negative error code if an error occurred.

DESCRIPTION:

This system call gives up all linear address space memory blocks allocated for the specified DPMI client. It can only be issued from protected mode. The *client* argument, which must be a number between 1 and 63, specifies the DPMI client.

This system call is intended primarily for the use of the DPMI routines, but it can be used by any user code operating in protected mode.

EXAMPLES:

svcMemChange - Change Memory Allocation

svcMemChange

CALLING SEQUENCE:

XOSSVC svcMemChange(void far *base, long type, long size);

VALUE RETURNED:

This call returns the new actual amount of memory allocated in the memory section if normal or a negative error code if an error occurred.

DESCRIPTION:

This function changes the amount of memory allocated to a memory section. If no memory section exists beginning at the address specified by the argument *base*, one is created. The size of the memory section beginning at the address specified is changed to be as close as possible to, but not less than, the specified size. It may not be possible to allocate the exact amount requested because of the granularity of memory allocation (4Kbytes). If a size of 0 is specified, the memory section is completely removed from the segment. If a size of -1 is specified, the current size is unchanged and is returned. This is the normal method of obtaining the current size of a memory section. This call is used to create, remove, increase the size of, and decrease the size of memory sections. When creating or increasing the size of a memory section, all pages to be allocated must be currently unused, or an ER_MACFT error is generated. Also, when creating or increasing the size of a memory section, the argument *type* specifies the page type for all newly allocated pages (the type of already allocated pages, if any, is not changed). The page type value is bit encoded as specified below.

Name	Bit	Meaning
PG$VIRTUAL	3	Virtual
PG$EXECUTE	2	Executable
PG$WRITE	1	Writable
PG$READ	0	Readable

When the PG$VIRTUAL bit is set, the address space for the page is allocated but no physical page is mapped. A page containing all 0s will be allocated and mapped the first time the program touches the page. This is

useful for allocating space for automatic expansion of a stack, for example.

The PG$EXECUTE bit is not used in the current version of XOS since the Intel 32-bit architecture does not implement the executable attribute at the page level. This attribute can be set only at the segment level.

The PG$WRITE bit is meaningful only for pages in data segments. When the bit is not set, the page is read-only. This is generally not useful when a page is initially allocated (since it will initially contain all 0s). The svcMemPageType system call allows this bit to be set after data has been written to a memory page.

The PG$READ bit is not used in the current version of XOS since the Intel 32-bit architecture does not implement no-read access at the page level (no-read access can only be specified at the segment level, and then only for code segments).

EXAMPLES:

svcMemConvShr - Convert to Shared Section

svcMemConvShr

CALLING SEQUENCE:

```
XOSSVC svcMemConvShr(void far *base, char far *name, long type, long level,
        long protection, void far *acl);
```

VALUE RETURNED:

This call returns the segment ID of the shared segment (a positive number) if normal or a negative error code if an error occurred.

DESCRIPTION:

This system call converts a private memory section to a shared memory section which is accessible by other processes. The argument *base* specifies the base address of the memory section to convert. The argument *name* specifies the name for the shared section. The argument *type* specifies the type the shared section. This type is unique to shared sections and specifies how sharing of the segment is managed. The valid values are defined in Table 10.2.

Table 10.2 - Shared section types

Name	Value	Meaning
SST$READWRITE	1	Read/write access
SST$READONLY	2	Read only access
SST$COPYWRITE	3	Copy on write

All pages in a shared section are either read/write or read only as indicated by the shared section type, regardless of whether the page was writable in the original private section. The copy on write shared section is not implemented in the current version of XOS. If it is specified, the section is treated as a read/write section.

The argument *level* specifies the compatibility level for the shared section. This is a 32-bit value. The high order 16-bits specify the major compatibility level and the low order 16-bits specify the minor compatibility level. A compatibility level must also be specified when a program links to an existing shared section. For the link to succeed, the two major levels must match exactly and the minor level specified by the program linking must be less than or equal to the minor level specified when the shared section

was created. This is intended to provide a way to verify that the version of the shared section is compatible with the version of the program which is accessing it. If this verification is not desired, this argument should be set to 0 both when creating the shared section and when linking to it.

The argument *protection* specifies the protection level of the shared section. This feature is not implemented in the current version of XOS. This argument should be specified as 0xFFFFFFFF to insure compatibility with future versions of XOS.

The argument *acl* specifies the access control list for the shared section. This feature is not implemented in the current version of XOS. This argument should be specified as a NULL address to insure compatibility with future versions of XOS.

EXAMPLES:

svcMemCopy2PM - Copy Data to Protected Mode Memory

svcMemBlkAlloc

CALLING SEQUENCE:

XOSSVC svcMemCopy2PM(void far16 *dest, void far32 *src, long count);

VALUE RETURNED:

This call returns 0 if normal or a negative error code if an error occurred.

DESCRIPTION:

This system call can only be issued from virtual-86 mode. It copies a block of data to protected mode memory. The *dest* argument specifies the protected mode address that the data is to be copied to. The *src* argument specifies the real mode address that the data is to be copied from. The *count* argument specifies the number of bytes to copy. The destination area must be allocated before this call is issued.

This system call allows virtual-86 mode code to copy data directly to protected mode memory, which it can not otherwise access. It is intended primarily for use by the DPMI routines, but can be used by any user code executing in real mode.

EXAMPLES:

svcMemCreate - Create New Segment

svcMemCreate

CALLING SEQUENCE:

XOSSVC svcMemCreate(long select, long type);

VALUE RETURNED:

This call returns the selector used (high 16 bits are zero) if normal or a negative error code if an error occurred.

DESCRIPTION:

This system call creates a new memory segment. It does not allocate any memory to this segment (see the svcMemChange system call). The *select* argument specifies the selector for the new segment (0 means to allocate a selector; 0FFFFh specifies the virtual-86 mode segment.) The *type* argument specifies the segment type, and is defined in Table 10.3.

Table 10.3 - Segment type values

Name	Value	Description
AT_32RODATA	1	32bit read only data
AT_32RWDATA	2	32bit read/write data
AT_32STACK	3	32bit stack
AT_32NXOCODE	4	32bit execute only normal code
AT_32NXRCODE	5	32bit execute/read normal code
AT_32CXOCODE	6	32bit execute only conformable code
AT_32CXRCODE	7	32bit execute/read conformable code
AT_16RODATA	9	16bit read only data
AT_16RWDATA	10	16bit read/write data
AT_16STACK	11	16bit stack
AT_16NXOCODE	12	16bit execute only normal code
AT_16NXRCODE	13	16bit execute/read normal code
AT_16CXOCODE	14	16bit execute only conformable code
AT_16CXRCODE	15	16bit execute/read conformable code
AT$TOP	0x80000000	Allocate highest available selector

A segment of the type specified is created. If a selector is specified, it is assigned to the segment; otherwise an available selector is allocated. Normally, the local selector, with the lowest value which is not in use, is used. If AT$TOP is set in the *type* argument, the local selector with the

highest value which is not in use is used. If a selector is specified, it must not be in use. The meaning of “highest value” is variable. Selector table space is allocated internally in blocks of 32 selectors. When allocating from the “highest value”, only currently allocated selector table space is considered. Thus, if no higher valued selector has been explicitly allocated, the “highest value” will be 0xF8 (the 32th selector).

If the virtual-86 segment is specified, the *type* argument is not used. When the virtual-86 segment is created, it is not immediately addressable by a protected mode program. The selector returned by this call will be 0xFFF8 in this case, which can only be used as an argument to the svcMemLink call to link an actual selector to the virtual-86 segment.

This function is not valid in virtual-86 mode.

EXAMPLES:

svcMemDebug - Memory Debug Functions

svcMemDebug

CALLING SEQUENCE:

```
XOSSVC svcMemDebug(long func, void far *addr, void far *value);
```

VALUE RETURNED:

This call returns the segment type (see svcMemCreate system call) if normal or a negative error code if an error occurred.

DESCRIPTION:

This function is intended for use by debuggers. It allows memory references to be made without worrying about memory traps, since if there is a problem, such as a non-existent memory location, an error code is returned. It also provides the type of segment for the address referenced. This information is generally needed by a debugger to interpret the contents of memory. Finally, it provides a simple way to modify the contents of a code segment, without having to set up an aliased data segment.

For the read functions, the value currently in the address specified by the argument *addr* is zero extended to 32-bits, if necessary, and is stored in the location pointed to by the argument value. For the write function, the value pointed to by the argument *value* is written into the address specified by the argument *addr*. Only as many bits as needed are used. The address being modified may be in any type of segment, even a code segment, which is not normally modifiable.

The argument *func* specifies the function as defined in Table 10.4.

Table 10.4 - Memory debug functions

Name	Value	Meaning
MDB_READBYTE	1	Read Byte
MDB_READWORD	2	Read Word
MDB_READLONG	3	Read Long
MDB_WRITEBYTE	4	Write Byte
MDB_WRITEWORD	5	Write word
MDB_WRITELONG	6	Write long
MDB_PHYSADDR	8	Return physical address

EXAMPLES:

svcMemDescAlloc - Allocate Segment Descriptor

svcMemDescAlloc

CALLING SEQUENCE:

XOSSVC svcMemDescAlloc(long selector, long num, long kind);

VALUE RETURNED:

This call returns the first selector allocated if normal or a negative error code if an error occurred.

DESCRIPTION:

This system call allocates a block of consecutive local segment descriptors. It can only be issued from protected mode. It will fail (with no descriptors allocated) if the requested number of descriptors cannot be allocated in a single contiguous block. The selector argument specifies the selector for the first descriptor to allocate. If bit 31 is set, the system will search for an available block of descriptors starting with the selector specified. If bit 31 is not set, the call will fail if the requested number of descriptors cannot be allocated starting with the selector specified. The num argument specifies the number of descriptors to allocate. The kind argument specifies the selector kind. Only the low 8 bits of the value are used. Bits 2 through 7 specify the DPMI client number and bits 0 through 1 specify the segment kind as follows:

Bits 0-1	Segment kind
0	Illegal
1	DPMI static segment
2	DPMI allocated segment
3	DPMI DOS memory segment

The descriptor is initialized to be a 16-bit data segment descriptor with byte granularity. It has a segment size of 1 byte and a base linear address of 0. It must be set up with the svcMemDescWrite or svcMemDescSet system calls before it can be used. The increment between consecutive descriptors allocated by this call is always 8.

This system call is intended primarily for the use of the DPMI routines, but it can be used by any user code operating in protected mode.

EXAMPLES:

svcMemDescFind - Find Segment Descriptor

svcMemDescFind

CALLING SEQUENCE:

XOSSVC svcMemDescFind(long kind, long linaddr);

VALUE RETURNED:

This call returns the value of the selector for the descriptor found if normal or a negative error code if an error occurred.

DESCRIPTION:

Finds a local segment descriptor given the descriptor kind and base linear address. The kind argument specifies the selector kind. Only the low 8 bits of the value are used. Bits 2 through 7 specify the DPMI client number and bits 0 through 1 specify the segment kind as follows:

Bits 0-1	Segment kind
0	Illegal
1	DPMI static segment
2	DPMI allocated segment
3	DPMI DOS memory segment

The linaddr argument specifies the base linear address for the selector. If more than one descriptor matches the *kind* and *linaddr* values, only the one with the lowest value selector is found.

This system call is intended primarily for the use of the DPMI routines, but it can be used by any user code operating in protected mode.

EXAMPLES:

svcMemDescFree - Give Up Segment Descriptor

svcMemDescFree

CALLING SEQUENCE:

```
XOSSVC svcMemDescFree(long selector);
```

VALUE RETURNED:

This call returns 0 if normal or a negative error code if an error occurred.

DESCRIPTION:

This system call gives up a segment descriptor allocated with the svcMemDescAlloc system call. It can only be issued for protected mode. The *selector* argument specifies the selector for the descriptor to give up.

This system call is intended primarily for the use of the DPMI routines, but it can be used by any user code operating in protected mode.

EXAMPLES:

svcMemDescRead - Read Segment Descriptor

svcMemDescRead

CALLING SEQUENCE:

XOSSVC svcMemDescRead(long selector, desc_value far *data);

VALUE RETURNED:

The value returned is 0 if normal or a negative error code if an error occurred. On a normal return, the *data* structure is filled in.

DESCRIPTION:

This system call returns the contents of the segment descriptor associated with the segment selector specified in the *selector* argument. The *data* argument specifies the address of an 8 byte data structure which is filled in with the contents of the descriptor. The format of this structure is the same as the format of the 80386 hardware segment descriptor, which is shown below.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Bit
Base 31:24								G	D	0	X	Limit 19:16				1	1	1	1	DC	0	WR	A	Base 23:16								+4
Base 15:00																Limit 15:00																+0

It should be noted that the segment limit value (20-bits) is stored as two separate items and the base linear address (32-bits) is stored as three items. This is the result of the 80386 descriptor maintaining compatibility with the original 80286 descriptor format. The single bit fields indicated as containing either 0 or 1 must contain the indicated value. The other single bit fields are as follows:

Bit	Description
G	Granularity: 0 = byte, 1 = page
D	Default width: 0 = 16-bit, 1 = 32-bit
X	Not used, available to user software
D/C	Code/data: 0 = code, 1 = data
W/R	If code, 0 = execute only, 1 = execute/read; if data, 0 = read only, 1 = read/write
A	Accessed; 0 = not accessed, 1 = accessed

This system call is intended primarily for the use of the DPMI routines, but it can be used by any user code operating in protected mode. This call can be used to read any user accessible descriptor, not just those allocated with the svcMemDescAlloc system call.

Examples:

svcMemDescSet - Set Value in Segment Descriptor

svcMemDescSet

CALLING SEQUENCE:

XOSSVC svcMemDescSet(long selector, long field, long data);

VALUE RETURNED:

This call returns the selector for the descriptor modified if normal or a negative error if an error occurred.

DESCRIPTION:

This system call sets a single field in a segment descriptor allocated with the svcMemDescAlloc call. It can only be issued from protected mode. The selector argument specifies the selector for the descriptor to modify. The field argument specifies the field to modify as follows.

Name	Value	Field
SDF_BASE	1	Base linear address
SDF_LIMIT	2	Segment limit
SDF_ACCESS	3	Access bits

When changing the access bits (SDF_ACCESS), the data value has the following format (the high order 16 bits must be 0).

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
G	D	0	X	0	0	0	0	1	1	1	1	DC	0	WR	A

This system call is intended primarily for the use of the DPMI routines, but it can be used by any user code operating in protected mode.

EXAMPLES:

svcMemDescWrite - Write Segment Descriptor

svcMemDescWrite

CALLING SEQUENCE:

```
XOSSVC svcMemDescWrite(long selector, desc_value far *data);
```

VALUE RETURNED:

This call returns 0 if normal or a negative error code if an error occurred.

DESCRIPTION:

This system call sets the contents of a segment descriptor which was allocated with the svcMemDescAlloc system call from the user's 8 byte data structure. It can only be issued from protected mode. The *selector* argument specifies the selector which corresponds to the descriptor being written. The data argument specifies the address of the user's data structure. The format of this data structure is the same as is used with the svcMemDescRead system call and is described on page 110.

This system call is intended primarily for the use of the DPMI routines, but it can be used by any user code operating in protected mode.

EXAMPLES:

svcMemDosSetup - Set Up DOS Memory

svcMemDosSetup

CALLING SEQUENCE:

```
XOSSVC svcMemDosSetup(long amount, dos_data far *data);
```

VALUE RETURNED:

This call returns 0 if normal or a negative error code if an error occurred. Also, values are stored in the caller's data structure as described below on a normal return.

DESCRIPTION:

This system call creates a real mode DOS environment. It can only be issued from protected mode. This call is primarily intended for internal use by svcIoRun routine. It can be issued directly by protected mode user programs, but this should not be necessary and is not recommended.

The process issuing this call must not have a real mode segment allocated. This call performs the following actions:

1. It allocates a real mode segment. This also allocates one page at 0xEC000 for the real mode stack page and maps the pages between 0xED000 and 0xEFFFF to the real mode system memory section. This memory section contains the real mode code used to implement the XOS native system calls and the DOS and DPMI system calls. These system calls are mostly implemented in protected mode, but a small amount of real mode code is required.

2. It allocates the first real mode page (page 0).

3. It fills all additional pages (if any) below the address specified by the value of the REALBASE PROCESS class characteristic with non-existent memory.

4. It physically allocates the amount of memory specified by the *amount* argument (starting at the address specified by the value of REALBASE).

5. It virtually allocates the additional pages needed to fill memory up to the size specified by the value of the value of the REALSIZE PROCESS class characteristic.

6. It fills any remaining pages below 0xA0000 with non-existent memory.

7. It maps the console display buffer between 0xA0000 and 0xBFFFF with unused memory in this range mapped to non-existent memory.

8. It pages between 0xC0000 and 0xEBFFF are filled with non-existent memory.

9. It pages between 0xED000 and 0xEF000 are mapped to the real mode system memory section.

10. It pages between 0xF0000 and 0xFFFFF are mapped to the physical BIOS ROMs.

11. The values of the REALBASE, REALSIZE, DOSFCBN, and DOSFCBP PROCESS class characteristics are stored in the data structure specified by the *data* argument. Each value is stored as a long (4 bytes).

12. It initializes the real mode vectors, the DOS FCB table, the other low memory areas used by the DOS emulator, and the DOS memory arena headers.

After a normal return from this system call, the process has created a 16-bit real mode DOS environment.

EXAMPLES:

svcMemLink - Link Segment Selectors

svcMemLink

CALLING SEQUENCE:

XOSSVC svcMemLink(long newselect, long oldselect, long type);

VALUE RETURNED:

This call returns the selector used (high 16-bits are 0) if normal or a negative error code if an error occurred.

DESCRIPTION:

This system call creates a new segment which references the same physical memory as an existing segment. It is most often used to create code and data spaces which map the same physical memory, allowing programs to appear to use only a single segment. It is also used to map user segments to exec mode segments for privileged programs which need to access kernel data and to allow protected mode user code access to the virtual-86 mode segment of the same process.

The argument newselect specifies a selector to link to the existing segment. A value of 0 means to allocate a new selector. The argument oldselect specifies the selector for the existing segment. It must reference an existing segment and can be a value of 0xFFF8 to 0xFFFF to reference the virtual-86 mode segment. It can reference an exec mode segment if the caller has sufficient privileges for the access requested (read or write). The argument type specifies the segment type for the linked segment, which does not have to be the same as that of the segment being linked to. See the description of the svcMemCreate system call for a definition of the segment type values.

EXAMPLES:

svcMemLinkShr - Link to Shared Section

svcMemLinkShr

CALLING SEQUENCE:

```
XOSSVC svcMemLinkShr(void far *base, char far *name, long level);
```

VALUE RETURNED:

This call returns the section ID of the shared section that was linked (a positive number) if normal or a negative error code if an error occurred.

DESCRIPTION:

This system call maps an existing shared memory section into the process' address space. This effectively creates a new memory section in the caller's address space. The size of this memory section is the size of the shared section and cannot be changed, except it can be set to 0 to unmap the shared section. The segment specified for the base address must exist. The section being linked to must have been previously created by some process with the svcMemConvShr system call, and must have been linked by at least one process at all times since it was created.

The argument base specifies the base address for the section in the callers process. Note that a shared section can be mapped at any base address, independent of the address where it was created. The argument name specifies the name of the segment, as created by the svcMemConvShr system call. The argument level determines the protection level of the segment. See the description of the svcMemConvShr call for a explanation of the use of this argument.

EXAMPLES:

svcMemMap - Map Physical Section

svcMemMap

CALLING SEQUENCE:

XOSSVC svcMemMap(void far *base, long phys, long type, long size);

VALUE RETURNED:

This call returns the actual amount of memory mapped(in bytes) if normal or a negative error code if an error occurred.

DESCRIPTION:

This function maps physical memory to the caller's address space. This effectively creates a new memory section. The size of this memory section is the size specified by the *size* argument and cannot be changed, except that it can be set to 0 to unmap the section. The segment specified for the base address must exist. A process must have the PHYMEM privilege to issue this call.

The argument *base* specifies the base address of the memory section to be created. The argument *phys* specifies the starting physical address. The type of the memory section created is specified by the *type* argument. See the description of the svcMemChange system call for a definition of the memory section type value. The amount of memory to map is specified in the *size* argument.

EXAMPLES:

svcMemMove - Move Memory Section

svcMemMove

CALLING SEQUENCE:

```
XOSSVC svcMemMove(void far *obase, void far *nbase);
```

VALUE RETURNED:

The value returned is the positive size in bytes of the memory section moved if normal or a negative error code if an error occurred.

DESCRIPTION:

This function moves a memory section (msect) for one segment and offset to another or to a different offset in the same segment. The move is done by changing the memory mapping data; no actual data is copied, so the operation is fairly efficient. The address space to which the section is being moved must be empty or an ER_MACFT error will result. This means that the new position cannot overlap the old position. If an overlap move is necessary, it can be done in two steps: first moving the section to an unused part of some segment's address space, and then moving it to the desired final position. The first argument, *obase*, specifies the original base address of the memory section to move. The second argument, *nbase*, specifies the new base address for the memory section. The segment referenced by the selector of the new base address must exist.

EXAMPLES:

svcMemNull - Map Null Memory

svcMemNull

CALLING SEQUENCE:

XOSSVC svcMemNull(void far *base, long pagebits, long size);

VALUE RETURNED:

This call returns the actual amount of memory mapped if normal or a negative error code if an error occurred.

DESCRIPTION:

This system call creates a memory section (msect) at the address specified by the base argument (rounded down to the beginning of a 4096 byte page). The characteristics of the pages allocated are given by the *pagebits* argument as follows:

Name	Bit	Meaning
PG$VIRTUAL	3	Virtual
PG$WRITE	1	Writable
	0	Must be 1

The size of the msect is specified by the value of the size argument (rounded up to whole 4096 pages). The msect created is filled with nonexistent memory pages. Bytes on these pages will usually be read as 0xFF. The pages mapped are actual non-existent physical pages, so the actual value read depends on the behavior of the hardware. Any data stored on these pages will be lost.

This system call is useful when it is necessary to avoid memory traps when unallocated memory is referenced.

EXAMPLES:

svcMemPageType - Change Memory Page Type

svcMemPageType

CALLING SEQUENCE:

```
XOSSVC svcMemPageType(void far *base, long bottom, long top, long type);
```

VALUE RETURNED:

This call returns 0 if normal or a negative error code if an error occurred.

DESCRIPTION:

This system call changes the type of one or more contiguous memory pages within a memory section. This function is most often used to set pages to read only, to change the allocation of real pages to virtual, or to force virtual pages to be allocated in physical memory.

The first argument, base, specifies the base address of the memory section containing the pages to be changed. The second argument, bottom, specifies the offset part of the address of the first page within the memory section to change. The third argument, top, specifies the offset part of the address of the first page within the memory section to not change. The fourth argument, type, specifies the new type for the pages. See the description of the svcMemChange" system call for a definition of the valid values for the page type.

Setting the PG$VIRTUAL bit to 0 for a page which was physically allocated causes the page (but not the address space) to be deallocated. The next time an address on the page is touched by the program, a new page containing all 0s will be allocated and mapped to that space. Setting the PG$VIRTUAL bit to 0 for a page which was not physically allocated has no effect. Setting the PG$VIRTUAL bit to 1 for a page which was not physically allocated causes a page containing all 0s to be allocated and mapped to that page. This is equivalent to touching the page. Setting the PG$VIRTUAL bit to 1 for a page which was physically allocated has no effect.

Changing the value of PG$WRITE bit changes the writability of the page without affecting its contents.

EXAMPLES:

svcMemRemove - Remove Segment

svcMemRemove

CALLING SEQUENCE:

XOSSVC svcMemRemove(long select);

VALUE RETURNED:

This call returns 0 if normal or a negative error code if an error occurred.

DESCRIPTION:

This system call removes the segment specified by the selector given in the select argument. If any memory sections are allocated in the segment, they are all deallocated before the segment is removed. This system call is not valid in virtual-86 mode.

EXAMPLES:

svcMemRmvMult - Remove Multiple Segments

svcMemRmvMult

CALLING SEQUENCE:

XOSSVC svcMemRmvMult(long data);

VALUE RETURNED:

This call returns 0 if normal or a negative error code if an error occurred. It is not an error if the segment specified for function 1 does not exist.

DESCRIPTION:

This system call removes multiple segments. The function is specified by bits 31 through 24 of the data argument. Bits 15 through 0 provide additional data for some of the functions. The functions are described below.

0. Remove all segments. All local memory segments allocated by the process are given up.

1. Remove all but one segment. The low order 16 bits of the data argument specifies the selector for the segment to keep (must be a local segment). All other local segments are given up.

2. Remove all but real mode segment. All segments except the real mode segment are given up.

3. Remove DPMI segments. Removes all DPMI segments for the DPMI client specified by the low order 6-bits of the *data* argument. These are all segments whose descriptors were allocated with the svcMemDescAlloc system call.

If this system call is executed from a segment which is given up, a segment not present trap will result when it attempts to return. This implies that function 0 can only be executed from a global segment and thus cannot be directly executed by user programs. This system call is used by the svcIoRun function which loads a program into the same process and by the DOS emulator to give up DPMI memory when terminating a DPMI client. It is normally not executed by user programs..

EXAMPLES:

svcMemSegType - Change Segment Type

svcMemSegType

CALLING SEQUENCE:

```
XOSSVC svcMemSegType(long select, long type);
```

VALUE RETURNED:

This call returns 0 if normal or a negative error code if an error occurred.

DESCRIPTION:

This function is used to change the type of an existing segment. The type of the segment specified by the argument select is set to the value specified by the argument *type*. See the description of the svcMemCreate system call for definition of the valid segment types. This system call is not valid in virtual-86 mode.

EXAMPLES:

svcMemWPFunc - Watchpoint Functions

svcMemWPFunc

CALLING SEQUENCE:

XOSSVC svcMemWPFunc(long function, long bits);

VALUE RETURNED:

The value returned depends on the function if normal or is a negative error code if an error occurred.

DESCRIPTION:

The system call returns the status of the process' watchpoints or clears watchpoints.

A function value of 1 returns the status of the watchpoints for the process and clears the triggered state of the watchpoints indicated by the *bits* argument. If bit 0 is set, watchpoint 0 is cleared, if bit 1 is set, watchpoint 1 is cleared, etc. If a bit is not set, the triggered state of the corresponding watchpoint is not changed.

In the value returned, bit 0 gives the status of watchpoint 0, bit 1 of watchpoint 1, etc. A set bit indicates that the corresponding watchpoint has been triggered. Attempting to clear the triggered state of a watchpoint which is not triggered or is not set is not an error.

A function value of 2 removes the indicated watchpoints. If bit 0 is set, watchpoint 0 is removed, if bit 1 is set, watchpoint 1 is removed, etc. If a bit is not set, the corresponding watchpoint is not removed. A value of 0 is returned. Attempting to remove a watchpoint which is not set is not an error.

EXAMPLES:

svcMemWPSet - Set Watchpoint

svcMemWPSet

CALLING SEQUENCE:

XOSSVC svcMemWPSet(long type, long size void far *address);

VALUE RETURNED:

The value returned is a watchpoint number if normal or a negative error code if an error occurred.

DESCRIPTION:

This system call sets watchpoints. The type argument specifies the type of the watchpoint as follows.

Name	Value	Meaning
WPT_EXECUTE	1	Trigger when instruction executed
WPT_WRITE	2	Trigger when location modified
WPT_ACCESS	3	Trigger when location accessed

The size argument specifies the size (in bytes) of the item in memory being monitored. Valid values are 1, 2, and 4. The value is ignored for WPT_EXECUTE watchpoints. The address argument specified the address for the watchpoint. The offset part of this address must be even modulo the value of the size argument. The size of WPT_EXECUTE watchpoints is always 1.

Because of hardware restrictions, only 4 watchpoints can be set.

EXAMPLES:

Chapter 11

I/O Parameters

A number of the I/O system calls use a device parameter list. This is a data structure consisting of a sequence of device parameter items, each of which specifies a function (get, set, or both get and set), a value description (format and length), a parameter index, and a value. The parameter list provides a general mechanism for reading or changing the values of various parameters associated with a device. Each parameter specification consists of a 4 byte header followed by a variable length data field. The format of the header is:

First byte	Second byte	Third byte	Fourth byte
Function & format	Value size	Parameter index	

The first byte specifies the function to perform and the value format. It is described in detail in Table 11.1.

Table 11.1 First I/O Parameter Header Byte

Bit	Name	Set by	Meaning		
7	SET	User	1 if should set value of parameter		
6	GET	User	1 if should get value of parameter		
5	ERROR	System	1 if error associated with this parameter		
4			Not used		
3-0	FORMAT	User	Value format as follows:		
			Value	Name	Meaning
			0		Not used
			1	DECV	Decimal value
			2	HEXV	Hex value
			3	OCTV	Octal value
			4	BINV	Binary value
			5	DECB	Decomal bytes
			6	HEXB	Hex bytes
			7	OCTB	Octal bytes
			8	VERN	Version number
			9	TIME	Time value
			10	DATE	Date value
			11	DT	Date/time value
			12	DATAB	Data bytes
			13	DATAS	Data string
			14	TEXT	Text bytes
			15	STR	String value

The value format field provides a fairly detailed specification of the type of value associated with the parameter. The current version of XOS does not differentiate between types 1 through 11, but simply treats them as general numeric values. Any of these values can be specified whenever a numeric value is called for. It is suggested that some effort be made to match the type with the specific type of value actually used, however, since future versions of XOS may use this information. Value types of 1 through 12 and 14 indicate a value which is stored in the value field which immediately follows the header. Value types of 13 and 15 indicate a value which is stored in a buffer which is specified by a buffer descriptor which immediately follows the header.

The C language symbols for the function bits are generated by prefixing PAR_ to the names given in the table above. The assembler names are generated by prefixing PAR$. The C and assembler names for the value formats are generated by prefixing REP_ to the name given above.

The second header byte contains the length of the value, in bytes. If a shorter value is specified than is needed when setting a parameter value, the value is zero extended to the necessary length. If a shorter value field than is expected is specified when getting a value and the actual value to be returned will fit in the field specified without truncation, the value is stored. If the value would be truncated, an ER_PARMS error is returned for the I/O operation and the PAR$ERROR bit is set in the parameter. When the value type is REP_STR, the second byte is not used and should contain 0. In this case, the value field consists of an 8 byte far pointer (high 2 bytes of the selector part are not used) which points to a buffer containing the string value (if PAR$SET set) or to receive the string value (if PAR$GET set), followed by two 2 byte values. The first of these values specifies the length of the buffer and the second specifies the length of the string in the buffer. The buffer length value must be set by the caller before issuing the system call. The string length value must be set by the caller if PAR$SET is specified. It is set by the system to indicate the actual length of the string returned if PAR$GET is specified.

The next two bytes contain the 16-bit parameter index value. This value indicates the parameter whose value is being set or read. Parameter index values of 0x8000 or less indicate global parameters which have the same meaning for all devices (although not all global parameters are legal for all devices). Parameter index values greater than 0x8000 indicate private parameters which only apply to devices in a particular class.

Table 11.2 summarizes the global parameters. The function (fnc) field indicates the valid function which can be used with the parameter (G = GET allowed, S = SET allowed, V = SET allowed but used to verify rather than change value). The type field indicates the minimum value size that will accept all possible values for the GET function. A smaller value size can be used when it is known the value being obtained will fit in the smaller size or when setting to a value which will fit in the smaller size.

Table 11.2 - Summary of I/O parameters

Name	Index	Fnc	Format	Size	Description
FILOPTN	0x0001	GS	HEXV	4	File specification option bits
FILSPEC	0x0002	GS	STR	1024	File specification string
DEVSTS	0x0003	GS	HEXV	4	Device status
UNITNUM	0x0004	GS	DECV	4	Unit number
GLBID	0x0005	GS	DATAS	16	Global device ID
DELAY	0x0006	GS	HEXV	4	I/O delay value (fractional days)
TIMEOUT	0x0007	GS	HEXV	4	I/O time-out value (fractional days)
INPSTS	0x0008	GS	DECV	1	Device input ready status

Table 11.2 - Summary of I/O parameters

Name	Index	Fnc	Format	Size	Description
OUTSTS	0x0009	GS	DECV	1	Device output ready status
INPQLMT	0x000A	GS	DECV	1	Input queue limit
OUTQLMT	0x000B	GS	DECV	1	Output queue limit
SIGVECT1	0x000C	GS	DECV	1	First signal vector
SIGVECT2	0x000D	GS	DECV	1	Second signal vector
SIGDATA	0x000E	GS	HEXV	4	Signal data
NUMOPEN	0x000F	GS	DECV	2	Number of times device is open
BUFRLMT	0x0010	GS	DECV	2	Internal buffering limit
DIRHNDL	0x0101	S	HEXV	2	Directory handle for search (old name if rename)
SRCATTR	0x0102	GS	HEXV	2	File attributes for search
FILATTR	0x0103	GS	HEXV	2	File attributes
DIROFS	0x0104	GS	DECV	4	Directory offset for search
ABSPOS	0x0105	GS	DECV	4	Absolute position in file
RELPOS	0x0106	GS	DECV	4	Relative position in file (returns absolute)
EOFPOS	0x0107	GS	DECV	4	Position in file relative to EOF (returns absolute)
VBOF	0x0108	GS	DECV	4	Virtual beginning of file position
LENGTH	0x0109	GS	DECV	4	Written length of file (bytes)
REQALLOC	0x010A	GS	DECV	4	Request file space allocation (bytes)
RQRALLOC	0x010B	GS	DECV	4	Require file space allocation (bytes)
GRPSIZE	0x010C	GS	DECV	4	Allocation group size (bytes)
ADATE	0x010D	GS	HEXV	8	Last access date/time
CDATE	0x010E	GS	HEXV	8	Creation date/time
MDATE	0x010F	GS	HEXV	8	Modify date/time
PROT	0x0110	GS	HEXV	4	File protection
OWNER	0x0111	GS	STR	16	Owner name
USER	0x0112	S	STR	-	User name for access
SETLOCK	0x0113	GS	HEXV	8	Set file lock
CLRLOCK	0x0114	GS	HEXV	8	Clear file lock
CLSTIME	0x0115	GS	HEXV	4	Close time value
CLSNAME	0x0116	GS	STR	256	Close name
CLSMSG	0x0117	GS	TEXT	16	Close message destination
SHRPARMS	0x0118	GS	HEXV	4	File sharing parameter values
ACSNETWK	0x0119	S	DECV	0	Use network access field
TRMSINPMODE	0x0201	GS	HEXV	4	Set input modes
TRMCINPMODE	0x0202	GS	HEXV	4	Clear input modes

Table 11.2 - Summary of I/O parameters

Name	Index	Fnc	Format	Size	Description
TRMSOUTMODE	0x0203	GS	HEXV	4	Set output modes
TRMCOUTMODE	0x0204	GS	HEXV	4	Clear output modes
TRMBFRLIMIT	0x0205	GS	DECV	4	Input buffer limit value
TRMCLRBUFR	0x0206	S	DECV	1	Clear buffer(s)
TRMCURTYPE	0x0207	GS	HEXV	2	Cursor type
TRMCURPOS	0x0208	GS	HEXV	4	Cursor position
TRMDISPAGE	0x0209	GS	DECV	1	Display page
TRMSPMODEM	0x020C	S	HEXV	1	Modem control
TRMSETDFC	0x020D	S	DATAS	-	Set data forwarding characters
TRMCLRDFC	0x020E	S	DATAS	-	Clear data forwarding characters
TRMLSTDFC	0x020F	GS	DATAS	-	Set or clear data forwarding characters
TRMALLDFC	0x0210	GS	DATAS	-	Set or clear all data forwarding characters
TRMCCVECT	0x0211	GS	DECV	2	Control-C vector
TRMCPVECT	0x0212	GS	DECV	2	Control-P vector
DSKFSTYPE	0x0301	G	DECV	1	File system type
DSKSECTSIZE	0x0302	G	DECV	2	Sector size
DSKCLSSIZE	0x0303	G	DECV	1	Cluster size
DSKTTLSPACE	0x0304	G	DECV	4	Total space (in clusters)
DSKAVLSPACE	0x0305	G	DECV	4	Available space (in clusters)
DSKNUMHEAD	0x0306	G	DECV	1	Number of heads
DSKNUMSECT	0x0307	G	DECV	1	Number of sectors
DSKNUMCYLN	0x0308	GS	DECV	4	Number of cylinders
TAPRECMIN	0x0401	G	DECV	4	Minimum record length
TAPRECMAX	0x0402	G	DECV	4	Maximum record length
NETSUBMASK	0x0501	G	HEXV	4	Sub-unit bit mask
NETPROTOCOL	0x0502	GS	HEXV	2	Network protocol
NETLCLPORT	0x0503	GS	DECV	2	Network local port number
NETRMTHWAS	0x0504	GS	HEXV	8	Network remote hardware address (send)
NETRMTHWAR	0x0505	GS	HEXV	8	Network remote hardware address (receive)
NETRMTNETAS	0x0506	GS	HEXV	4	Network remote network address(send)
NETRMTNETAR	0x0507	GS	HEXV	4	Network remote network address (receive)
NETRMTPORTS	0x0508	GS	DECV	2	Network remote port number (send)
NETRMTPORTR	0x0509	GS	DECV	2	Network remote port number (receive)

Table 11.2 - Summary of I/O parameters

Name	Index	Fnc	Format	Size	Description
NETDSTNAME	0x050A	GS	STR	256	Network destination name
NETSMODE	0x050B	GS	HEXV	4	Set network mode bits
NETCMODE	0x050C	GS	HEXV	4	Clear network mode bits
NETRCVWIN	0x050D	GS	DECV	4	Network receive window size
IPMRMTPID	0x0601	GS	HEXV	4	IPM remotePID
DGLCLADDR	0x0701	GS	STR	64	Datagram local address
DGRMTADDRS	0x0702	GS	STR	64	Datagram remote address (send)
DGRMTADDR	0x0703	G	STR	64	Datagram remote address (receive)
RUNCMDTAIL	0x1001	S	STR	-	Command tail (argumant list)
RUNDEVLIST	0x1002	S	DATAS	-	Device list
RUNENVLIST	0x1003	S	STR	-	Additional environment data
RUNDEBUGBFR	0x1004	G	DATAS	***	Buffer for debug data
RUNADDRESS	0x1005	GS	HEXV	8	Load address
RUNRELOCVAL	0x1006	GS	HEXV	4	Relocation value
RUNFCB1	0x1007	S	DATAS	12	First DOS FCB
RUNFCB2	0x1008	S	DATAS	12	Second DOS FCB
RUNACTPRIV	0x1009	S	STR	-	Active privileges for child
RUNAVLPRIV	0x100A	S	STR	-	Available privileges for child
RUNWSLIMIT	0x100C	S	DECV	4	Working set size limit for child
RUNWSALLOW	0x100D	S	DECV	4	Working set size allowed for child
RUNTMLIMIT	0x100E	S	DECV	4	Total user memory limit for child
RUNTMALLOW	0x100F	S	DECV	4	Total user memory allowed for child
RUNPMLIMIT	0x1010	S	DECV	4	Protected mode memory limit for child
RUNPMALLOW	0x1011	S	DECV	4	Protected mode memory allowed for child
RUNRMLIMIT	0x1012	S	DECV	4	Real mode memory limit for child
RUNRMALLOW	0x1013	S	DECV	4	Real mode memory allowed for child
RUNOMLIMIT	0x1014	S	DECV	4	Overhead memory limit for child
RUNOMALLOW	0x1015	S	DECV	4	Overhead memory allowed for child
CLASS	0x8000	S	TEXT	-	Device class

Even though these I/O parameters are all defined for all devices in the system, some of them apply mainly to specific types of devices. When a given parameter does not apply to a device, the parameter will either be ignored (if the result of ignoring it would be mostly benign) or will generate an ER_PARMF error (if ignoring it would produce significantly different behavior than expected). For example, most of the

terminal specific global parameters are ignored by disk devices, allowing terminal specific output to be redirected to a disk file with more or less reasonable results.

The following sections provide detailed descriptions of each global parameter.

Common Parameters

Common parameters apply to all devices in the system. All devices accept all of these parameters.

IOPAR_FILOPTN = 0x0001 - File option bits

This parameter is used to set the file option bits. These bits specify which parts of the device name and file specification are to be stored in the file specification string (see IOPAR_FILSPEC on page 134). This value may only be set. It may be specified for any I/O operation, but complete information is returned only for operations which act on a file specification (open, delete, rename, etc.). Only the FO$DEVNAME, FO$DOSNAME, or FO$VOLNAME bits are used if the I/O operation does not act on a file specification. It is ignored if the IOPAR_FILSPEC parameter is not also specified. It must be specified as having a numeric value. The maximum useful length of the value is 4 bytes. The value is bit encoded as follows (all bits not listed are reserved for future use and should always be set to 0):

Bit	Name	Meaning
31	FO$NOPREFIX	Do not insert prefix codes
23	FO$PHYNAME	Return physical device name even if rooted name
22	FO$DOSNAME	Return DOS disk name
21	FO$VOLNAME	Return disk volume name
20	FO$XOSNAME	Return XOS device name
19	FO$NODENUM	Return network node number
18	FO$NODENAME	Return network node name
17	FO$NODEPORT	Return network node port number
15	FO$RPHYNAME	Return remote physical device name even if rooted name
14	FO$RDOSNAME	Return remote DOS disk name
13	FO$RVOLNAME	Return remote disk volume name
12	FO$RXOSNAME	Return remote XOS device name

Bit	Name	Meaning
11	FO$PATH	Return file path
10	FO$FILENAME	Return file name and extension
7	FO$ATTR	Return file attribute bytes
6	FO$VERSION	Return file version number (VAX/VMS file systems only)
0	FO$MASK	Return search mask

IOPAR_FILSPEC = 0x0002 - File specification string

This parameter is used to specify the buffer to receive the file specification string, as requested by the IOPAR_FILOPTN parameter. It can be specified with the PAR$GET function only. The string stored in the buffer consists of the parts of the device and file specification as requested by the value of the IOPAR_FILOPTN parameter, with each optionally preceded by a prefix code. If all bits except for FO$ATTR and FO$MASK are set, the string returned forms the canonical file specification for the file being operated on (after prefix bytes are stripped) and can always be used as an argument to specify that file (the canonical file specification is the complete name, including the device, path, filename, extension, and version number. No logical names are permitted in the canonical specification). If the IOPAR_FILOPTN parameter is not specified, a null string is returned as the value of this parameter. The prefix byte values are summarized below.

Name	Value	Description
FS_MASK	0xD0	Search mask follows
FS_MASKAST	0xD1	* match is next
FS_MASKQUES	0xD2	? match is next
FS_MASKPER	0xD3	period is next
FS_RTDNAME	0xE0	Rooted device name follows
FS_DOSNAME	0xE1	DOS disk name follows
FS_VOLNAME	0xE2	Volume name follows
FS_XOSNAME	0xE3	XOS device name follows
FS_NODENUM	0xE4	Node number follows
FS_NODENAME	0xE5	Node name follows
FS_NODEPORT	0xE6	Node port number follows
FS_RRTDNAME	0xE8	Remote rooted device name follows
FS_RDOSNAME	0xE9	Remote DOS disk name follows
FS_RVOLNAME	0xEA	Remote volume name follows

Name	Value	Description
FS_RXOSNAME	0xEB	Remote XOS device name follows
FS_PATH	0xF0	Path follows
FS_FILENAME	0xF1	File name and extension follow
FS_VERSION	0xF2	File version number follows
FS_ATTR	0xF3	File attributes follow
FS_NPATH	0xF8	New path follows
FS_NFILENAME	0xF9	New file name and extension follow
FS_NVERSION	0xFA	New file version number follows
FS_TRUNC	0xFE	Entry truncated
FS_ESC	0xFF	Next byte is literal

The prefix bytes are described in detail below.

Only one of the following four prefix codes will appear. If FO$XOSNAME is set, then FS_XOSNAME or FS_RTDNAME will always be present. If FO$VOLNAME is set, then FS_VOLNAME will be present if the device has a volume name; otherwise FO_DEVNAME will be present. If FO$DOSNAME is set, then FS_DOSNAME will be present if the device has a DOS device letter associated with it; FO_VOLNAME will be present if the device has a volume name; otherwise FO_DEVNAME will be present.

FS_RTDNAME = 0xE0

Rooted device name follows. The name which follows is a rooted logical name, followed by a colon.

FS_DOSNAME = 0xE1

Disk DOS name follows. The DOS drive letter associated with the device follows, followed by a colon.

FS_VOLNAME = 0xE2

Disk volume name follows. The name which follows is the disk volume name followed by a colon.

FS_XOSNAME = 0xE3

XOS device name follows. The name which follows is the name of an XOS physical device followed by a colon.

The following two prefix codes are only used when the device is a network device. At most one will appear. If FS$NODENUM is set, FS_NODENUM will be present. If FS$NODENAME is set,

FS_NODENAME will be present if a name is associated with the network node; otherwise FS_NODENUM will be present.

FS_NODENUM = 0xE4

Network node number follows. The number of the remote node in dotted decimal notation follows, followed by two colons. For example, 200.3.5.2::.

FS_NODENAME = 0xE5

Network node name follows. The domain name of the remote node follows; it is followed by two colons (for example: TOM.SALES.XYZCORP.COM::).

FS_NODEPORT = 0xE6

The network node port number follows. This prefix is only used when the device is a network device. The report port number used follows, given as an ASCII decimal value.

The following four prefix codes are only used when the device is a network device. At most one will appear. They describe the remote device being accessed in the same way as the three device prefix codes describe the local device.

FS_RDEVNAME = 0xE8

Remote rooted device name follows. The name which follows is a remote rooted logical name, followed by a colon.

FS_RDOSNAME = 0xE9

Remote disk DOS name follows. The DOS drive letter associated with the remote device follows, followed by a colon.

FS_RVOLNAME = 0xEA

Remote disk volume name follows. The remote disk volume name follows, followed by a colon.

FS_RXOSNAME = 0xEB

Remote XOS device name follows. The name which follows is the name of a remote XOS physical device followed by a colon.

FS_PATH = 0xF0

Path is next - the full directory path follows. It always begins and ends with the \ character. If the file is in the root directory, the path consists of a single \ character.

FS_FILENAME = 0xF1

File name is next, including the extension. The file name follows. For DOS and VMS file systems, a period is always included, even if there is no extension. For UNIX file systems, periods are not special.

FS_VERSION = 0xF2

File version number follows. The file version number follows as a string of decimal digits. This is only present for files on a VMS file system.

FS_ATTR = 0xF3

File attribute bytes follow. This is followed by two bytes. The first byte contains the attribute byte for the file as a binary value and the second byte always contains the binary value 0 (a future version of XOS may return additional information in the second byte).

The following three prefix values are returned only by a rename operation. They provide information about the new file specification.

FS_NPATH = 0xF8

Path for new specification follows. This is the same as FS_PATH except it applies to the new file specification.

FS_NFILENAME = 0xF9

New file name follows, including extension. This is the same as FS_FILENAME except it applies to the new file specification.

FS_NVERSION = 0xFA

New file version number follows. This is the same as FS_VERSION except it applies to the new file specification.

FS_TRUNC = 0xFE

Entry has been truncated. This will appear as the last byte in the file specification string if the buffer was too short to hold the entire string.

FS_ESC = 0xFF

Escape. This indicates that the following byte is a literal character and not a prefix code. It is used whenever a character with a value between 0xD0 and 0xFF appears in a file specification.

IOPAR_DEVSTS = 0x0003 - Device status

This parameter returns the device status bits. The function must be PAR$GET and the value must be numeric (REP_HEXV) with a length of 4 bytes. The bits returned are defined as follows:

Bit	Name	Description
21	DS$UPRNAME	File names are forced to upper case
20	DS$LWRNAME	File names are forced to lower case
15	DS$SPOOL	Device is spooled
14	DS$CONTROL	Device is controlling terminal for group
13	DS$NOABORT	Device cannot be aborted
12	DS$UNBFRD	Device should be unbuffered
10	DS$MAPPED	Memory mapped device
9	DS$MLTUSER	Multi-user device (any process can open device)
8	DS$DUPLEX	Device is full duplex (simultaneous input and output)
7	DS$FILE	Device is file structured
6	DS$SODIR	Device supports search open directory operation
5	DS$PHYS	Physical device
4	DS$REMOVE	Removable media device
3	DS$DOUT	Device can do direct output
2	DS$DIN	Device can do direct input
1	DS$QOUT	Device can do queued output
0	DS$QIN	Device can do queued input

IOPAR_UNITNUM = 0x0004 - Unit number

This parameter returns the unit status bits. The function must be PAR$GET and the value must be numeric (REP_HEXV) with a length of 4 bytes. This parameter always returns a value of 0 in the current version of XOS.

IOPAR_GLBID = 0x0005 - Global device ID

This parameter returns a unique value which identifies a file or device. The function must be PAR$GET and the value must be numeric (REP_HEXV) with a length of 16 bytes. The value obtained may be used to determine if two files which were opened using different file specifications are really the same file. If the IOPAR_GLBID value for two files is identical, then the two files are really the same file. If the values are different, then the two files are not the same file. This value has no other use.

IOPAR_DELAY = 0x0006 - I/O delay value (fractional days)

This parameter specifies a delay value. All queued I/O operations for the device will be delayed the length of time specified. The delay occurs before the I/O is done. The value stays in effect until changed. The initial delay value when a device is opened is always 0.

IOPAR_TIMEOUT = 0x0007 - I/O time-out value (fractional days)

This parameter specifies a time out value for an I/O operation. The function must be PAR$SET and the value must be numeric (REP_HEXV) with a length of 4 bytes. The value specifies fractional days as an unsigned binary fraction with the binary point to the left of bit 31. The maximum value which can be specified is 0x7FFFFFFF, which is just under 12 hours.

IOPAR_INPSTS = 0x0008 - Device input ready status

This parameter returns the input ready status for a device. The value is 1 if the device has input available (an input operation will return at least one byte without blocking) or 0 if no input is currently available. Disk devices always indicate ready unless at end of file. The function must be PAR$GET and the value must be numeric (REP_HEXV) with a length of 1 byte.

IOPAR_OUTSTS = 0x0009 - Device output ready status

This parameter returns the output ready status for a device. The value is 1 if the device can accept output (an output operation will accept at least one byte without blocking) or 0 if output is currently busy. Disk devices always indicate ready unless the disk is full. The function must be PAR$GET and the value must be numeric (REP_HEXV) with a length of 1 byte.

IOPAR_INPQLMT = 0x000A - Input queue limit

This parameter returns or sets the maximum number of input operations which can be queued for a device or file. When a device is opened, the input queue limit is initialized to 1, effectively disabling input queuing for the device. The value of this parameter may be set to a larger value to enable the desired level of input queuing. The value set remains in effect until changed or until the device is closed. This parameter is only allowed for full duplex devices. Specifying it for a half duplex device will result in a ER_PARMI error. The value must be numeric.

IOPAR_OUTQLMT = 0x000B - Output queue limit

This parameter returns or sets the maximum number of output operations which can be queued for a device or file. When a device is opened, the output queue limit is initialized to 1, effectively disabling output queuing for the device. The value of this parameter may be set to a larger value to enable the desired level of output queuing. The value set remains in effect until changed or until the device is closed. Since half duplex devices use the same queue for input and output, this parameter sets the combined limit for queued input and output requests for such devices. The value must be numeric.

IOPAR_SIGVECT1 = 0x000C - First signal vector

This parameter returns or sets the value of the first software interrupt vector for a device. The value must be between 0 and 31 to specify a software interrupt vector. Specifying this parameter for a device which does not use it causes an ER_PARMF error. The use of this vector by the device is device dependent. The value must be numeric with a length of 1 byte.

IOPAR_SIGVECT2 = 0x000D - Second signal vector

This parameter returns or sets the value of the second software interrupt vector for a device. The value must be between 0 and 31 to specify a software interrupt vector. Specifying this parameter for a device which does not use it causes an ER_PARMF error. The use of this vector by the device is device dependent. The value must be numeric (REP_HEXV) with a length of 1 byte.

IOPAR_SIGDATA = 0x000E - Signal data

This parameter returns or sets a value which is associated with the use of software interrupts by a device. Specifying this parameter for a device which does not use it causes an ER_PARMF error. The allowable range and the use of the value are device dependent. The value must be numeric (REP_DECV) with a length of 4 bytes.

IOPAR_NUMOPEN = 0x000F - Number of times device is open

This parameter returns the number of times the file or device is open. The function must be PAR$GET and the value must be numeric (REP_DECV) with a length of 2 bytes.

Mass Storage Parameters

The following parameters are specific to mass storage devices. This includes all DISK class devices and all remote mass storage devices.

IOPAR_DIRHNDL = 0x0101 - Directory handle for search

This parameter specifies the handle of an open directory (see description for O$ODF) which is to be searched for an operation which uses a file specification. When this parameter is specified, only the name and extension of the file specification is used. The device, node, remote device, and path parts are ignored. This parameter can only be specified for file structured devices. If the device is not file structured, an ER_PARMF error is returned. The function must be PAR$SET and the value must be numeric with a length of 4 bytes.

IOPAR_SRCATTR = 0x0102 - File attributes for search

This parameter specifies the attribute byte value to use when searching for a file. The bits in the value are interpreted as follows:

Bit	Name	Meaning
7	A$NORMAL	Normal file
4	A$DIRECT	Directory
3	A$LABEL	Volume label
2	A$SYSTEM	System file
1	A$HIDDEN	Hidden file

If a bit is set, the corresponding type of file is matched. If A$LABEL is set, only volume labels will be matched; otherwise all types indicated will be matched. The value is ignored if a file is being created, in which case all types are matched to prevent the creation of more than one file with the same name. This parameter is ignored if the device is not file structured. The function must be PAR$SET and the value must be numeric with a length of 1 byte.

IOPAR_FILATTR = 0x0103 - File attributes

This parameter gets or sets the attributes for a file. The bits in the value are interpreted as follows:

Bit	Name	Meaning
7	A$NORMAL	Normal file
5	A$ARCH	Archive bit (set if file has been modified)
4	A$DIRECT	Directory
3	A$LABEL	Volume label
2	A$SYSTEM	System file
1	A$HIDDEN	Hidden file
0	A$RDONLY	Read only file

The A$DIRECT and A$LABEL attributes cannot be changed once a file exists. The other bits can be changed at any time. The normal way to create a directory is to specify setting the A$DIRECT attribute and set the O$CREATE command bit in an open call. This parameter is ignored if the device is not file structured. The value must be numeric (REP_HEXV) with a length of 2 bytes. The current version of XOS always stores a value of 0 in the second byte of the value. This byte may be used in future versions of XOS to store additional information.

IOPAR_DIROFS = 0x0104 - Directory offset for search

This parameter sets or returns the position in a directory at which a directory search begins. It can be specified with any operation which involves searching a directory. It applies only to the operation with which it is specified. When the PAR$SET bit is set, the value given is used to specify where the search begins. When the PAR$GET bit is set, the value returned specifies the position in the directory immediately following the entry for the file which was matched by the call. A value of 0 specifies the beginning of a directory. Other than this, the meaning of the value is not defined and may vary for different file systems. The value returned by this parameter should only be used as a value when setting this parameter. The value must be numeric (REP_HEXV) with a length of 4 bytes.

IOPAR_ABSPOS = 0x0105 - Absolute position in file

This parameter returns or sets the absolute position in a file at which data will be transferred next. The value is the byte offset from the beginning of the file. This parameter is processed before any I/O is done. If it is specified with a system call which transfers data, data is transferred starting at the position specified. The value returned indicates the position at which the transfer started. The value must be numeric (REP_HEXV) with a length of 4 or more bytes. The current version of XOS does not support file lengths greater than 2^{32} bytes, but future versions may support very

long files on some file systems. In this case, a value longer than 4 bytes will be required for this parameter.

IOPAR_RELPOS = 0x0106 - Relative position in file

This parameter returns the the absolute position in a file at which data will be transferred next or sets the position relative to the current I/O position for a file. The value is the byte offset from the beginning of the file. When PAR$SET is set, the value specified, taken as a signed number, is added to the current I/O position. This parameter is processed before any I/O is done. If it is specified with a system call which transfers data, data is transferred starting at the position specified. The value returned indicates the position at which the transfer started. The value must be numeric (REP_HEXV) with a length of 4 or more bytes. The current version of XOS does not support file lengths greater than 2^{32} bytes, but future versions may support very long files on some file systems. In this case, a value longer than 4 bytes will be required for this parameter.

IOPAR_EOFPOS = 0x0107 - Position in file relative to EOF

This parameter returns the absolute position in a file at which data will be transferred next or sets the position relative to the end of the file. The value is the byte offset from the beginning of the file. When PAR$SET is set, the value specified, taken as a signed number, is added to the length of the file.

This parameter is processed before any I/O is done. If it is specified with a system call which transfers data, data is transferred starting at the position specified. The value returned indicates the position at which the transfer started. The value must be numeric (REP_HEXV) with a length of 4 or more bytes. The current version of XOS does not support file lengths greater than 2^{32} bytes, but future versions may support very long files on some file systems. In this case, a value longer than 4 bytes will be required for this parameter.

IOPAR_VBOF = 0x0108 - Virtual beginning of file position

This parameter returns or sets the position in the file which is considered to be the beginning of the file. Data before this position cannot be accessed. All file positions are calculated relative the virtual beginning of file.

IOPAR_LENGTH = 0x0109 - Written length of file

This parameter returns or sets the written length of a file in bytes. When PAR$GET is set, the value returned is the written length of the file in bytes. If this is specified with an I/O operation which will change the length of the file, the value returned is the length before the I/O operation is done. When PAR$SET is set, the written length of the file is set to the value specified. If this value is less than the current written length, the file is truncated. If it is greater, the file is extended by writing zeros between the current end of file and the requested position.

If PAR$SET is set and the file has not been opened for output (O$OUT not set in the open call), an ER_PARMF error is returned. The value must be numeric (REP_HEXV) with a length of 4 or more bytes. The current version of XOS does not support file lengths greater than 2^{32} bytes, but future versions may support very long files on some file systems. In this case, a value longer than 4 bytes will be required for this parameter.

IOPAR_REQALLOC = 0x010A - Request file space allocation

This parameter returns the current allocated length of a file in bytes (PAR$GET set) or requests that the number of bytes specified be allocated to the file (PAR$SET set). If the requested number of bytes cannot be allocated, all available space is allocated. This parameter is processed before any I/O is done. If it is specified with a system call which transfers data, the value returned reflects the amount allocated to the file before the data is transferred.

The value must be numeric (REP_HEXV) with a length of 4 or more bytes. The current version of XOS does not support file lengths greater than 2^{32} bytes, but future versions may support very long files on some file systems. In this case, a value longer than 4 bytes will be required for this parameter.

IOPAR_RQRALLOC = 0x010B - Require file space allocation

This parameter returns the current allocated length of a file in bytes (PAR$GET set) or allocates the number of bytes specified to the file (PAR$SET set). If the requested number of bytes cannot be allocated, an ER_DSKFL error is returned. This parameter is processed before any I/O is done. If it is specified with a system call which transfers data, the value returned reflects the amount allocated to the file before the data is transferred.

The value must be numeric (REP_HEXV) with a length of 4 or more bytes. The current version of XOS does not support file lengths greater than 2^{32} bytes, but future versions may support very long files on some file systems. In this case, a value longer than 4 bytes will be required for this parameter.

IOPAR_GRPSIZE = 0x010C - Allocation group size

This parameter returns or sets the amount of space allocated (in bytes) each time space is allocated to a file, except when the IOPAR_REQALLOC and IOPAR_RQRPALLOC parameters are used. When a file is opened using the DOS file system, this value defaults to six times the cluster size for the device. If it is changed with this parameter, the changed value is only in effect until the file is closed. Other file systems may save this value as a file attribute and use it when a file is subsequently accessed. If a value larger than the maximum value allowed for the file system is specified, the maximum value is used. The maximum value for the DOS file system is 256 times the cluster size. The value must be numeric (REP_DECV) with a length of two bytes.

IOPAR_ADATE = 0x010D - Last access date/time

This parameter returns or sets the date and time at which a file was last accessed. For file systems which do not keep track of the last access date/time, this parameter references the modify date/time, if it is kept, or the creation date/time. For the DOS file system, it references the creation date/time, which is the only date/time value associated with a file. Changing this value requires that the caller have the ability to change file attributes; otherwise an ER_PRIV error is returned. Setting the parameter for non-file structured devices has no effect; in this case a value of 0 is returned. The value must be numeric (REP_HEXV) with a length of 4 or 8 bytes. If the length is 4 bytes, the value is in the DOS file system date/time format. If the length is 8 bytes, the value is in the XOS system date/time format.

IOPAR_CDATE = 0x010E - Creation date/time

This parameter returns or sets the date and time at which a file was created. Changing this values requires that the caller have the ability to change file attributes; otherwise an ER_PRIV error is returned. Setting the parameter for non-file structured devices has no effect; in this case a value of 0 is returned. The value must be numeric (REP_HEXV) with a length of 4 or 8 bytes. If the length is 4 bytes, the value is in the DOS file system

date/time format. If the length is 8 bytes, the value is in the XOS system date/time format.

IOPAR_MDATE = x010F - Modify date/time

This parameter returns or sets the date and time at which a file was last modified. For file systems which do not keep track of the last modified date/time, this parameter references the creation date/time which is the case for the DOS file system. Changing this value requires that the caller have the ability to change file attributes; otherwise an ER_PRIV error is returned. Setting the parameter for non-file structured devices has no effect; getting the value in this case returns a value of 0. The value must be numeric (REP_HEXV) with a length of 4 or 8 bytes. If the length is 4 bytes, the value is in the DOS file system date/time format. If the length is 8 bytes, the value is in the XOS system date/time format.

IOPAR_PROT = x0110 - File protection

This parameter returns or sets the protection values associated with a file.

Since the current version of XOS does not support file protection, this parameter has no effect for any device.

Setting the parameter for non-file structured devices has no effect; getting the value in this case returns a value of 0. The value must be numeric (REP_HEXV) with a length of 4 bytes.

IOPAR_OWNER = 0x0111 - Owner name

This parameter returns or sets the owner name associated with a file. Changing this value requires that the caller have the ability to change file attributes; otherwise an ER_PRIV error is returned. Setting the parameter, for non-file structured devices or for file systems which do not support file ownership features, has no effect; getting the value in this case returns a text string containing all nulls. The value must be text (REP_TEXT) with a length of 16 bytes.

IOPAR_USER = 0x0112 - User name for access

This parameter returns or sets the group name associated with a file. Changing this value requires that the caller have the SPCUSER privilege; otherwise an ER_PRIV error is returned. Setting the parameter for non-file structured devices or for file systems which do not support file ownership features has no effect; getting the value in this case returns a text string containing all nulls. The value must be a string (REP_STR) which can have a length of up to 33 bytes.

IOPAR_SETLOCK = 0x0113 - Set file lock

This parameter specifies that an area of a file is to be locked. When an area is locked, no one else accessing the file may lock any area which overlaps the locked area. It does not restrict other access to the locked region (such as reading or writing data) in any way. If the some part of the region is already locked, the operation will be repeated as specified by the value of the IOPAR_SHRPARMS parameter. If the lock cannot be granted after the specified number of tries, an ER_LOCK error is returned. The value of this parameter must be numeric and be 8 bytes in length. The first 4 bytes contain the offset in the file of the first byte of the region to lock and the second 4 bytes contain the length of the region to lock, in bytes.

IOPAR_CLRLOCK = 0x0114 - Clear file lock

This parameter specifies that an area of a file which is currently locked is to be unlocked. The area specified must exactly match (both offset and length) the area specified when the area was locked. The value of this parameter must be numeric and be 8 bytes in length. The first 4 bytes contain the offset in the file of the first byte of the region to unlock and the second 4 bytes contain the length of the region to unlock, in bytes.

IOPAR_CLSTIME = 0x0115 - Close time value

This parameter is only used with spooled devices. It is only valid with the OPEN function. It specifies the length of time (in fractional days) that can elapse without any IO before the device is automatically closed. Each spooled device has a default close time which is normally used. This parameter can be used to override the default value for a single instance of a spooled device. The value must be numeric with a length of 4 bytes.

IOPAR_CLSNAME = 0x0116 - Close name

This parameter is only used with spooled devices. It is only valid with the OPEN function. It specifies the name to be used for the spool file when the device is closed. This name should contain a sequence of 1 or more #'s. The #'s will be replaced with the sequence number of the spooled file. The number of #'s indicates the number of digits inserted and thus the number of unique spooled file names which will be generated. Each spooled device has a default close name which is normally used. This parameter can be used to override the default value for a single instance of a spooled device. The value must be a string.

IOPAR_CLSMSG = 0x0117 - Close message destination

This parameter is only used with spooled devices. It is only valid with the OPEN function. It specifies the name to be used as the destination for the IPM message which is generated when the device is closed. Each spooled device has a default close message destination which is normally used. This parameter can be used to override the default value for a single instance of a spooled device. The value must be a string.

IOPAR_SHRPARMS = 0x0118 - File sharing parameters

This parameter specifies the values used when setting file locks (see the IOPAR_SETLOCK and IOPAR_CLRLOCK IO parameters). The value must be numeric with a length of 4 bytes. The first two bytes specify the number of times to retry a lock attempt. The second two bytes specify the time to delay between attempts, in milliseconds.

IOPAR_ACSNETWK = 0x0119 - Use network access field

This parameter, if present, specifies that file access is to be determined based on the “network” access field. This parameter will fail with an ER_PRIV error is the process does not have the ACSNETWK privilege. The value of this parameter is not used. It should be specified as numeric with a length of 0.

Terminal Parameters

The following parameters are specific to terminal devices.

IOPAR_TRMSINPMODE = 0x0201 - Set input mode bits

This parameter sets terminal input mode bits or returns the input mode bits. When PAR$SET is set, all bits which are set in the value are set in the terminal input mode bits. When PAR$GET is set, the current terminal input mode bits are returned. The terminal input mode bits are defined as follows:

Bit	Name	Meaning
30	TIM_SCNALL	Return all scan codes from keyboard, including key break codes
29	TIM_SCNCODE	Return raw scan codes from keyboard (keyboard mode 3)
28	TIM_PCSCNC	Convert keyboard scan codes to PC (keyboard mode 1) scan codes
27	TIM_PC101	Use PC mode special codes
26	TIM_PCDOS	Filter PC scan codes for DOS (changes 0xE0 prefix to 0)
25	TIM_ANSI	Return ANSI special codes
24	TIM_ALTPAD	Suppress ALT keyboard handling
23	TIM_NOCO	Control-O is not special on input
22	TIM_NOCC	Control-C is not special on input
15	TIM_DEFER	Do not wake up process immediately on image input (wait for full buffer)
9	TIM_ILFACR	Insert LF after CR in input stream
8	TIM_RCRBLF	Remove CR before LF in input stream
7	TIM_OVER	Initial input edit mode is overstrike
6	TIM_DEBUG	Debug mode input
5	TIM_CHAR	Character mode input
4	TIM_IMAGE	Image mode input
3	TIM_XIMAGE	Special image mode input
1	TIM_ECHO	Echo input

The value must be numeric (REP_HEXV) with a length of at least 4 bytes.

IOPAR_TRMCINPMODE = 0x0202 - Clear input mode bits

This parameter clears terminal input mode bits or returns the input mode bits. The bits are the same as defined above for the

IOPAR_TRMSINPMODE parameter. The value must be numeric (REP_HEXV) with a length of at least 4 bytes.

IOPAR_TRMSOUTMODE = 0x0203 - Set output mode bits

This parameter sets terminal output mode bits or returns the output mode bits. When PAR$SET is set, all bits which are set in the value are set in the terminal output mode bits. When PAR$GET is set, the current terminal output mode bits are returned. The terminal output mode bits are defined as follows:

Bit	Name	Meaning
26	TOM$ANSICM	Do ANSI character mapping
9	TOM$ICRBLF	Insert CR before LF is output
4	TOM$IMAGE	Image mode output

The value must be numeric (REP_HEXV) with a length of at least 4 bytes.

IOPAR_TRMCOUTMODE = 0x0204 - Clear output mode bits

This parameter clears terminal output mode bits or returns the current output mode bits. The bits are the same as defined above for the IOPAR_TRMSOUTMODE parameter. The value must be numeric (REP_HEXV) with a length of at least 4 bytes.

IOPAR_TRMBFRLIMIT = 0x0205 - Input buffer limit value

This parameter sets or returns the input buffer limit value. The buffer limit value specifies the maximum number of characters which will be buffered in image mode when an input request is active. Setting the buffer limit to 1 is equivalent to clearing the TIM$DEFER input mode bit. More data will be buffered, up to the actual length of the terminal input buffer, if no input operation is pending. The value must be numeric (REP_DECV) with a length large enough to hold the value.

IOPAR_TRMCLRBUFR = 0x0206 - Clear terminal buffer(s)

Setting this parameter (PAR$SET set) causes the buffers indicated by the value to be cleared. If PAR$GET is set, an ER_PARMF error is returned. The value is a single byte with the bits interpreted as follows:

Bit	Name	Meaning
6	CB$OUTPUT	Clear output buffer when set
5	CB$INPUT	Clear current input buffer when set
4	CB$AHEAD	Clear type-ahead buffer when set

Bits not specified here are reserved for future use and should be 0. The value must be numeric (REP_HEXV) with a length of 1 byte.

IOPAR_TRMCURTYPE = 0x0207 - Cursor type

This parameter sets or reports the cursor type for display terminals. It is ignored if specified for a non-display terminal. The first byte of the value specifies the starting scan line for the alphanumeric mode cursor block. A value of 0 specifies the bottom line of the character cell and a value of 255 specifies the top line of the character cell. A value of 128 specifies the middle line of the character cell, etc.

The second byte specifies the top scan line for the cursor block with the same encoding as the first byte. If the bottom scan line is more than the top scan line, no cursor is displayed. If the bottom and top scan lines are equal, a single scan line cursor is displayed. The value must be numeric (REP_HEXV) with a length of 2 bytes.

IOPAR_TRMCURPOS = 0x0208 - Cursor position

This parameter returns or sets the current cursor position for display terminals. It is ignored if specified for a non-display terminal. The first byte of the value specifies the column number and the second byte specifies the row number for the alphanumeric mode cursor. The value must be numeric (REP_HEXV) with a length of 2 bytes.

IOPAR_TRMDISPAGE = 0x0209 - Display page

This parameter returns or sets the current display page for display terminals. It is ignored if specified for a non-display terminal. The value specifies the current page number. The value must be numeric (REP_DECV) with a length of 1 byte.

IOPAR_TRMSPMODEM = 0x020C - Modem control

This parameter returns and sets the modem control bits associated with a serial port. The use of these bits is summerized below.

Name	Value	Description
DTRVAL	0x01	specifies and returns DTR state
RTSVAL	0x02	specifies and returns RTS state
CTSVAL	0x10	returns CTS state
DSRVAL	0x20	returns DSR state

CDVAL	0x80	returns CD state

Other bits are not used and should be 0. The value must be numeric (REP_HEXV) with a length of 1 byte.

IOPAR_TRMSETDFC = 0x020D - Set data forwarding characters

This parameter sets one or more characters to be data forwarding characters. The argument is a data string. Each byte in the string specifies a character which is set to be a data forwarding character.

IOPAR_TRMCLRDFC = 0x020E - Clear data forwarding characters

This parameter sets one or more characters to not be data forwarding characters. The argument is a data string. Each byte in the string specifies a character which is set to not be a data forwarding character.

IOPAR_TRMLSTDFC = 0x020F - Set or clear data forwarding characters

This parameter sets the data forwarding state for one or more characters. The argument is a data string, which must contain an even number of bytes.. Each byte pair in the string specifies a character whose data forwarding state is to be set. The first byte specifies the character, the second byte specifies the data forwarding state. A 0 values clears the data forwarding state. Any other value sets the data forwarding state for the character.

IOPAR_TRMALLDFC = 0x0210 - Set or clear all data forwarding characters

This parameter sets or clears the data forwarding state of characters using a bit array. Each byte specifies the data forwarding state of 8 contiguous characters, with the low order bit specifying the state of the lowest value character. The states of the lowest valued 8 * N characters are set, where there are N bytes specified. The arguement is a byte string which can be up to 32 bytes in length. A 32 byte string specifies the states of all 256 possible characters.

IOPAR_TRMCCVECT = 0x0211 - Control-C vector

This parameter specifies a signal vector which is used to request a signal whenever ^C is typed on the controlling terminal. It is ignored if the device is not the controlling terminal for a session. The value must be numeric with a length of 2 bytes.

IOPAR_TRMCPVECT = 0x0212 - Control-P vector

This parameter specifies a signal vector which is used to request a signal whenever ^P is typed on the controlling terminal. It is ignored if the device is not the controlling terminal for a session. The value must be numeric with a length of 2 bytes.

Disk Parameters

The following parameters are specific to local or remote disk devices. They are all read-only parameters which are used to obtain various values associated with the disk. These values are also available as device characteristics and are duplicated here as I/O parameters to make them easier to access for remote disks. In particular, these parameters are used to implement the various DOS functions which return disk information.

IOPAR_DSKFSTYPE = 0x0301 - File structure type

This parameter returns the file structure type. This is the value stored in the partition table on the disk to identify a partition. The possible values are as follows:

Name	Value	Description
	0x00	Not file structured
FS_DOS12	0x01	DOS file system with 12-bit FAT tables
FS_DOS16	0x04	DOS file system with 16-bit FAT tables
FS_DOS16H	0x06	DOS file system with 16-bit FAT tables 32MB or larger
FS_DOS32	0x0B	DOS file system with 32-bit FAT tables
FS_DOS32X	0x0C	DOS file system with 32-bit FAT tables using LBA
FS_DOS16X	0x0E	DOS file system with 16-bit FAT tables using LBA
FS_XFS	0x23	XOS native file system
FS_DSS12	0xE1	DOS SpeedStor compatible with 12-bit FAT tables
FS_DSS12L	0xF1	DOS SpeedStor extended with 12-bit FAT tables
FS_DSS16	0xE4	DOS SpeedStor compatible with 16-bit FAT tables
FS_DSS16L	0xF4	DOS Speedstor extended with 16-bit FAT tables

The value must be numeric (REP_HEXV) with a size of at least 1 byte.

IOPAR_DSKSECTSIZE = 0x0302 - Sector size

This parameter returns the sector size (in bytes) for the disk. The value must be numeric (REP_DECV) with a size of at least 2 bytes.

IOPAR_DSKCLSSIZE = 0x0303 - Cluster size

This parameter returns the cluster size (in sectors) for the disk. If the disk is not file structured, a value of 0 is returned. The value must be numeric (REP_DECV) with a size of at least 2 bytes.

IOPAR_DSKTTLSPACE = 0x0304 - Total space

This parameter returns the total space (in clusters) on the disk. If the disk is not file structured, a value of 0 is returned. The value must be numeric (REP_DECV) with a size of at least 4 bytes.

IOPAR_DSKAVLSPACE = 0x0305 - Available space

This parameter returns the currently available space (in clusters) on the disk. If the disk is not file structured, a value of 0 is returned. The value must be numeric (REP_DECV) with a size of at least 4 bytes.

IOPAR_DSKNUMHEAD = 0x0306 - Number of heads

This parameter returns the number of heads on the disk. The value must be numeric (REP_DECV) with a size of at least 1 byte.

IOPAR_DSKNUMSECT = 0x0307 - Number of sectors

This parameter returns the number of sector per track. The value must be numeric (REP_DECV) with a size of at least 1 byte.

IOPAR_DSKNUMCYLN = 0x0308 - Number of cylinders

This parameter returns the number of cylinders on the disk. The value must be numeric (REP_DECV) with a size of at least 2 bytes.

Tape Parameters

The following parameters are specific to tape devices.

IOPAR_TAPRECMIN = 0x0401 - Minimum record length

This parameter specifies or returns the minimum allowed length for a physical tape record.

IOPAR_TAPRECMAX = 0x0402 - Maximum record length

This parameter specifies or returns the maximum allowed length for a physical tape record.

IOPAR_TAPRECLEN = 0x0403 - Current fixed record length

This parameter specifies or returns the allowed minimum length for a physical tape record.

Network Parameters

The following parameters are specific to network devices. Unless otherwise specified, these parameters are valid for all devices. Non-network devices ignore attempts to set parameter values and always return a numeric value of 0 or a null string.

IOPAR_NETSUBUMASK = 0x0501 - Sub-unit bit mask

The value returned for this parameter specifies the sub-units which exist for a network device. Each subunit provides a physical network connection. Each network device unit can support a number of such connections. The value returned is numeric and is bit encoded. If bit 0 is set, sub-unit A exists, if bit 1 is set, sub-unit B exists, etc. The value should have a length of at least 4 bytes.

IOPAR_NETPROTOCOL = 0x0502 - Network protocol

This parameter returns or sets the protocol value associated with a network device which uses an interface which supports multiple protocols. The value can only be set for devices which directly access raw network data (such as NETn:). It can be read for any network device which uses an interface which supports multiple protocols. If the device is not a network device, an ER_PARMI error is returned. If the device does not support multiple protocols, a value of 0 is returned. In this case, an ER_PARMF error is returned if the value is set. The value must be numeric (REP_HEXV) with a length of at least 2 bytes.

IOPAR_NETLCLPORT = 0x0503 - Network local port number

This parameter returns or sets the local port number for a network device which uses port numbers. If the device is not a network device or does not use port numbers, an ER_PARMI error is returned. If PAR$SET is set and bit 31 of the value is set, the low order byte of the port number is not changed. If bit 31 of the value is set, the value will be incremented repeatedly until an acceptable port number is found. If bit 30 of the value is set, duplicate port numbers are allowed; otherwise an ER_PARMV error is returned if the port number specified is in use. Network devices which use a port number are assigned a default port number when they are opened. This parameter can be used to obtain the value of this default port number or to change it to some other value. For IP devices, the default port number is always a private port with a value greater than 1024. If a public port (value less than 1024) is needed, this parameter is used to change the port

number to the desired value. The value must be numeric (REP_HEXV) with a length of at least 2 bytes.

IOPAR_NETRMTHWAS = 0x0504 - Network remote hardware address (send)

This parameter returns or sets the remote hardware address. It is usually used to obtain the value, since the remote hardware address is usually obtained automatically by the low level network routines. The value must be numeric (REP_HEXV) with a length of at least 8 bytes to guarantee that all format addresses can be returned.

IOPAR_NETRMTHWAR = 0x0505 - Network remote hardware address (receive)

This parameter obtains the local network hardware address for a network device. The value must be numeric (REP_HEXV) with a length of at least 8 bytes to guarantee that all format addresses can be returned.

IOPAR_NETRMTNETAS = 0x0506 - Network remote network address (send)

This parameter returns or sets the remote node address for a network device which is used when sending data. This parameter is normally used when sending datagrams to specify the destination network address of individual datagrams. The value must be numeric (REP_HEXV) with a length of at least 4 bytes.

IOPAR_NETRMTNETAR = 0x0507 - Network remote network address (receive)

This parameter returns the remote network address associated with a network device. This can be either a connection oriented or a datagram device. This is particularly useful when the node was originally specified using a domain name and the actual matching network address is needed. The value must be numeric (REP_HEXV) with a length of at least 4 bytes.

IOPAR_NETRMTPORTS = 0x0508 - Network remote port number (send)

This parameter gets or sets the remote port number used when sending data for a network device which uses port numbers. It is used when establishing a connection to a “public” remote port to specify the number of the public port. It is also be used when sending datagrams to specify the remote port. It can also be used to obtain the remote port number being used. The value must be numeric (REP_HEXV) with a length of at least 2 bytes.

IOPAR_NETRMTPORTR = 0x0509 - Network remote port number (receive)

This parameter returns the remove network port number associated with a network device. This can be either a connection oriented or a datagram device. The value must be numeric (REP_HEXV) with a length of at least 2 bytes.

IOPAR_NETDSTNAME = 0x050A - Network destination name

This parameter returns the actual destination name which was used to specify as a remote network address. If a destination name was not used, a null string is returned. The value must be a string (REP_STR). The length should in general be at least 128 bytes to allow for all possible name formats, although a shorter length can be used if it is known that the name will be shorter than this.

IOPAR_NETSMODE = 0x050B - Set network mode bits

This parameter sets the specified network mode bits. The value specified, which must be numeric with a length of 4 bytes, is ored with the current mode bits value. The table below summerizes the network mode bits.

Name	Value	Description
IDLEMARK	0x00004000	Send mark when idle
SENDSYNC	0x00002000	Send sync pattern before data packets
BRDCST	0x00000800	Broadcast next packet
PUSH	0x00000020	Do push after TCP output
CONP	0x00000010	Push is conditional (Nagel algorithm

To form the C language symbol for each bit, prefix the name with NMODE_. TO form the assembler symbol for each bit, prefix the name with NMODE$. The value returned is the current value of all the network mode bits.

IOPAR_NETCMODE = 0x050C - Clear network mode bits

This parameter clears the specified network mode bits. The value specified, which must numeric with a length of 4 bytes, is complemented and anded with the current mode bits value. See the IOPAR_NETSMODE above for a description of the network mode bits. The value returned is the current value of all the network mode bits.

IOPAR_NETRCVWIN - 0x050D - Network receive window size

This parameter set or returns the receive window size. This value is either the maximum number of bytes or frames in the receive window, depending on the protocol type. For protocols which negotiate a window size, this value can only be set when the network device is opened and then only represents a target value. Negotiation may result in a different value, depending on the protocol type. Reading this parameter always returns the actual (negotiated) value.

Interprocess Message Parameters

The following parameter is specific to IPM (inter-process message) devices.

IOPAR_IPMRMTPID = 0x0601 - IPM remote PID

This parameter returns the PID of the sender or receiver of an IPM message. It is only valid when specified with a datagram input or output function for an IPM device. The value must be numeric (REP_HEXV) with a length of at least 4 bytes.

Datagram Parameters

The following parameters are used by devices which send and receive datagrams. A datagram is a message which is independently routed.

IOPAR_DGLCLADDR = 0x0701 - Datagram local address

This parameter returns the local address associated with a datagram device. The value must be a string (REP_STR) with a length of at least 128 bytes.

IOPAR_DGRMTADDR = 0x0702 - Datagram remote address (send)

This parameter returns or sets the remote address used when sending a datagram. The value must be a string (REP_STR) with a length of at least 128 bytes when getting the value.

IOPAR_DGRMTADDRR = 0x0703 - Datagram remote address (receive)

This parameter returns the remote address from which a datagram was received. The value must be a string (REP_STR) with a length of at least 128 bytes.

svcIoRun Parameters

The following parameters are used with the svcIoRun and svcSchSpawn system calls.

IOPAR_RUNCMDTAIL = 0x1001 - Command tail (argument list)

This parameter specifies a string which is passed to the program being loaded as the command tail. If a DOS program is being loaded, the first item in the command tail (which is assumed to be the program name) is removed before it is stored in the DOS program segment prefix. Also, the default FCBs in the program segment prefix are initialized from the second and third items from this parameter unless the IOPAR_FCB1 or IOPAR_FCB2 parameters (which override the respective items from the command tail) are specified. PAR$SET must be set and PAR$GET must be clear. The value must be a string (REP_STR).

IOPAR_RUNDEVLIST = 0x1002 - Device list

This parameter specifies a device list which controls which devices are passed to the program being loaded in a child process. It is ignored if the program is being loaded in the same process. PAR$SET must be set and PAR$GET must be clear.

The device list is a list of 8 byte items. The first item specifies the device for handle 0 in the child process; the second specifies the device for handle 1, etc. The first four bytes of each item specify the handle of the device in the process issuing the system call or a special code. A value of -1 indicates the end of the list and a value of -2 indicates that no device is to be passed to the child process for that handle. If the value is not -1 or -2, the low 30 bits specify the handle. If bit 31 is set, the device is transferred to the child process. If it is clear, it is duplicated in the child process. Setting bit 31 has the same effect as closing the device after it is passed to the child process. If bit 30 is set, ownership of the device is passed to the child process. Note that this is implied if bit 31 is set. It is only meaningful if the process issuing the system call is the owner of the device.

If a device is specified, the next four bytes specify the open command bits for the device. A value of all 0s indicates that the current open commands associated with the device should be used. If the first four bytes contain -1, the next 4 bytes are not used. If the first four bytes contain -2, the next 4 bytes are not used but must be present and should contain 0.

Note that the use of a -1 value to indicate the end of the list is optional. The end may also be indicated by the value of the string length field in the parameter. The value must be a string (REP_STR).

IOPAR_RUNENVLIST = 0x1003 - Additional environment data

This parameter specifies a list of environment strings to be defined for the new process. It consists of a list of null terminated strings, each of which has the format:

ENVNAME=environment string value

The end of the list is indicated by another null character or by the value of the string length field in the parameter. The value must be a string (REP_STR).

IOPAR_RUNDEBUGBFR = 0x1004 - Buffer for debug data

This parameter specifies the location of a buffer to receive information about values used to relocate data in the program being loaded and initial register values. This information is useful to debuggers which use separate symbol table files and must relocate symbol values to match the loaded program. PAR$GET must be set and PAR$SET must be clear. The format of the data returned is as follows:

Offset	Size	Description
0	4	Initial value for EAX
4	4	Initial value for ECX
8	4	Initial value for EDX
12	4	Initial value for EBX
16	4	Initial value for ESP
20	4	Initial value for EBP
24	4	Initial value for ESI
28	4	Initial value for EDI
32	4	Initial value for EPC
36	4	Initial value for EFR
40	4	Initial value for DS
44	4	Initial value for ES
48	4	Initial value for FS
52	4	Initial value for GS
56	4	Initial value for CS
60	4	Initial value for SS
64	4	Total size of loaded program
68	4	Number of address spaces

Offset	Size	Description
70	2	Number of msects
72		Start of relocation data item

The relocation data consists of a list of segment selectors, one for each segment (stored in 4 bytes each) followed by a list of 16 byte msect description blocks, one for each msect. Each msect description block has the following format:

Offset	Size	Description
0	4	Segment number (starting with 1)
4	4	Base offset for msect
8	4	Size of msect (in bytes)
12	4	Not used

The value must be a string (REP_STR). It is suggested that the length be at least 172 bytes. This will allow for 4 segments and 4 msects. A length of about 300 bytes should allow for data for the most complex memory image which can be loaded.

IOPAR_RUNADDRESS = 0x1005 - Load address

This parameter returns or sets the load address for an overlay. The value must be numeric (REP_HEXV) with a length of 4 bytes.

IOPAR_RUNRELOCVAL = 0x1006 - Relocation value

This parameter returns or sets the relocation value for an overlay. The value must be numeric (REP_HEXV) with a length of 4 bytes.

IOPAR_RUN_FCB1 = 0x1007 - First DOS FCB

This parameter specifies the address of a buffer containing an image of the first FCB to be set up for a DOS program. This parameter is ignored if the program being loaded is not a DOS program. If this parameter is not present and a DOS program is being loaded, the first FCB is initialized for the second item in the IOPAR_ARGUMENT parameter. The value must be a string (REP_STR) with a string length of 12 bytes.

IOPAR_RUNFCB2 = 0x1008 - Second DOS FCB

This parameter specifies the address of a buffer containing an image of the second FCB to be set up for a DOS program. This parameter is ignored if the program being loaded is not a DOS program. If this parameter is not present and a DOS program is being loaded, the second FCB is initialized

for the third item in the IOPAR_ARGUMENT parameter. The value must be a string (REP_STR) with a string length of 12 bytes.

IOPAR_RUNACTPRIV = 0x1009 - Active privileges for child process

This parameter specifies the process privileges which are to be passed to the child process as its initial active privileges. The privileges are specified as an ASCII string with the same format as used by the ACTPRIV and AVLPRIV characteristics for the process class (see Chapter 12). Any privileges which are not available to the parent or child process are ignored. The value must be a string (REP_STR).

IOPAR_RUNAVLPRIV = 0x100A - Available privileges for child process

This parameter specifies the process privileges which are to be available to the child process. The privileges are specified as an ASCII string with the same format as used by the ACTPRIV and AVLPRIV characteristics for the process class (see Chapter 12). Any privileges which are not available to the parent process are ignored. The value must be a string (REP_STR).

IOPAR_RUNWSLIMIT = 0x100C - Working set size limit for child

This parameter specifies the initial working set limit value for the child process. If the value specified is greater than the maximum value allowed for the child process (see IOPAR_WSALLOW), the maximum value allowed is used. The value must be numeric (REP_DECV) with a maximum useful length of 4 bytes.

IOPAR_RUNWSALLOW = 0x100D - Working set size allowed for child

This parameter specifies the maximum working set size allowed for the child process. If the value is greater than the maximum value allowed for the parent process, then that maximum value is used. The value must be numeric (REP_DECV) with a maximum useful length of 4 bytes.

IOPAR_RUNTMLIMIT = 0x100E - Total user memory limit for child

This parameter specifies the initial total user memory limit value for the child process. If the value is greater than the maximum value allowed (see IOPAR_TMALLOW), then the maximum value allowed is used. The value must be numeric (REP_DECV) with a maximum useful length of 4 bytes.

IOPAR_RUNTMALLOW = 0x100F - Total user memory allowed for child

This parameter specifies the maximum total user memory size allowed for the child process. If the value is greater than the maximum value allowed

for the parent process, then that maximum value is used. The value must be numeric (REP_DECV) with a maximum useful length of 4 bytes.

IOPAR_RUNPMLIMIT = 0x1010 - Protected mode memory limit for child

This parameter specifies the initial protected mode memory limit value for the child process. If the value is greater than the maximum value allowed (see IOPAR_PMALLOW), then the maximum value allowed is used. The value must be numeric (REP_DECV) with a maximum useful length of 4 bytes.

IOPAR_RUNPMALLOW = 0x1011 - Protected mode memory allowed for child

This parameter specifies the maximum protected mode memory size allowed for the child process. If the value is greater than the maximum value allowed for the parent process, then that maximum value is used. The value must be numeric (REP_DECV) with a maximum useful length of 4 bytes.

IOPAR_RUNRMLIMIT = 0x1012 - Real mode memory limit for child

This parameter specifies the initial real mode memory limit value for the child process. If the value is greater than the maximum value allowed (see IOPAR_RMALLOW), then the maximum value allowed is used. The value must be numeric (REP_DECV) with a maximum useful length of 4 bytes.

IOPAR_RUNRMALLOW = 0x1013 - Real mode memory allowed for child

This parameter specifies the maximum real mode memory size allowed for the child process. If the value is greater than the maximum value allowed for the parent process, then that maximum value is used. The value must be numeric (REP_DECV) with a maximum useful length of 4 bytes.

IOPAR_RUNOMLIMIT = 0x1014 - Overhead memory limit for child

This parameter specifies the initial overhead memory limit value for the child process. If the value is greater than the maximum value allowed (see IOPAR_OMALLOW), then the maximum value allowed is used. The value must be numeric (REP_DECV) with a maximum useful length of 4 bytes.

IOPAR_RUNOMALLOW = 0x1015 - Overhead memory allowed for child

This parameter specifies the maximum overhead memory size allowed for the child process. If the value is greater than the maximum value allowed

for the parent process, then that maximum value is used. The value must be numeric (REP_DECV) with a maximum useful length of 4 bytes.

Device Class Parameter

IOPAR_CLASS = 0x8000 - Device class

This parameter is used to get or verify the device class. When PAR$GET is set, the name of the class of the device is returned. When PAR$SET is set, the name specified is compared to the name of the class of the device. If it is different, an ER_PARMV error is returned. If it is the same, an internal flag is set indicating that the class name has been verified. This flag must be set before any private device parameters can be used. This is done to insure that the action of any private parameters will be as expected, since the same parameter index may represent completely different private parameters for devices in different classes. The value must be text (REP_TEXT) with a length of 8 bytes.

The private parameters are described in the sections which describe the specific device classes to which they apply.

Chapter 12

Class Characteristics

This chapter describes the class characteristics for all standard device classes in the system. A class characteristic is a named item which specifies a value associated with a device class. Class characteristics are the primary mechanism by which the user obtains and sets the values of the various items which specify the state and operation of the XOS operating system. They provide a standardized interface which apply to all device classes.

A device class is the collection of all devices of a specific type. For example, the TRM device class includes all terminal-like devices. The DISK device class includes all disk devices, that is, all devices which support local file systems. The definition of device class is actually more abstract and flexible than these two examples would indicate. Two of the most significant device classes do not include any actual devices at all. These are the SYSTEM and PROCESS device classes. The SYSTEM class is provided to allow the class characteristic mechanism to be used to manipulate various values associated with the system as a whole (such as the amount of memory available or the response of the system to CTL-ALT-DEL). Likewise, the PROCESS class allows access to values associated with individual processes.

Class characteristics are specified using a class characteristic list. This is a data structure consisting of a sequence of class characteristic items followed by a byte containing 0. Each class characteristic item consists of a 10 byte header followed by a 0 to 32 byte value.

The first header byte specifies the function to perform and the value format. It is described in detail in Figure 12.1.

Figure 12.1 First - Class Characteristic Header Byte

Bit	Name	Set by	Meaning		
7	SET	User	1 if should set value of characteristic		
6	GET	User	1 if should get value of characteristic		
5	ERROR	System	1 if error associated with this characteristic		
4			Not used		
3-0	FORMAT	User	Value format as follows:		
			Value	Name	Meaning
			0		Not used
			1	DECV	Decimal value
			2	HEXV	Hex value
			3	OCTV	Octal value
			4	BINV	Binary value
			5	DECB	Decimal bytes
			6	HEXB	Hex bytes
			7	OCTB	Octal bytes
			8	VERN	Version number
			9	TIME	Time value
			10	DATE	Date value
			11	DT	Date/time value
			12	DATAB	Data bytes
			13	DATAS	Data string
			14	TEXT	Test bytes
			15	STR	String value

The value format field provides a fairly detailed specification of the type of value associated with the parameter. The current version of XOS does not differentiate between types 1 through 11, but simply treats them as general numeric values. Any of these values can be specified whenever a numeric value is called for. It is suggested that some effort be made to match the type with the specific type of value actually used, however, since future versions of XOS may use this information. Value types of 1 through 12 and 14 indicate a value which is stored in the value field which immediately follows the header. Value types of 13 and 15 indicate a value which is stored in a buffer which is specified by a buffer descriptor which immediately follows the header.

The C language symbols for the function bits are generated by prefixing PAR_ to the names given in the table above. The assembler names are generated by prefixing PAR$. The C and assembler names for the value formats are generated by prefixing REP_ to the name given above.

The second header byte contains the length of the value, in bytes. If a shorter value is specified than is needed when setting a characteristic value, the value is zero extended to the necessary length. If a shorter value field than expected is specified when getting a value and the actual value to be returned will fit in the field specified without truncation, the value is stored. If the value would be truncated, an ER_CHARS error is returned for the I/O operation and the PAR$ERROR bit is set in the characteristic. When the value type is FMT_STR, the second byte is not used and should contain 0. In this case, the value field consists of an 8 byte far pointer (high 2 bytes of the selector part are not used) which points to a buffer containing the string value (if PAR$SET set) or to receive the string value (if PAR$GET set), followed by two 2 byte values. The first of these values specifies the length of the buffer and the second specifies the length of the string in the buffer. The buffer length value must be set by the caller before issuing the system call. The string length value must be set by the caller if PAR$SET is specified. It is set by the system to indicate the actual length of the string returned if PAR$GET is specified.

The third through tenth bytes of the header contain the ASCII name of the characteristic. It is not case sensitive. If the name is shorter than 8 characters, the field is filled with NULL characters.

Class characteristics are accessed using the QFNC_CLSCHAR function of the svcIoQueue system call, which is described in Chapter 15. A CLSCHAR command is provided (which is described in the XOS User's Guide), which provides access to the class characteristics values from the command line.

This chapter contains a number of tables which summarize the class characteristics for the various devices. These tables all have the same format. The first column specifies the name of the characteristic (1 to 8 characters, not case sensitive but listed by convention as upper case). The second column specifies the format of the value. This corresponds to the PAR$FORMAT value listed in Figure 12.1, above, without the FMT_ prefix. The third column specifies the minimum value length which can contain all possible values of the characteristic. The fourth column contains one or more of the letters g, s, or v. The letter g indicates that a get operation (PAR$GET set in the first byte of the characteristic) is allowed. The letter s indicates that a set operation (PAR$SET set in the first byte of the characteristic) is allowed. The letter v also indicates that a "set" operation is allowed but that it will verify, rather than change the value.

It should be noted that the interpretation of the format is exactly the same for device characteristics (see Chapter 13) , and is the same as for device parameters (see Chapter 16), except that the type of numeric format is significant here. A sub-function of the QFNC_CLASS function allows a program to obtain the format of a characteris-

tic, which can then be used to format the value for output. This feature is used by the CLSCHAR command to display each value in the proper format.

This chapter lists the class characteristics by device class. Technically, there is no relationship between characteristics for different classes which have the same name, but in practice all characteristics with the same names represent roughly equivalent values for their respective classes. Programmers implementing new device classes are strongly encouraged to use the established names but only in the same way they are used for the standard device classes.

SYSTEM Class Characteristics

The SYSTEM device class is a dummy device class which exists only to provide a mechanism for accessing and changing various values associated with the XOS operating system as a whole. The SYSTEM class characteristics are summarized in Table 12.1.

Table 12.1 - SYSTEM Class Characteristics

Name	Fnc	Format	Size	Description
AVAILMEM	G	DECV	4	Available memory (in KB)
DEBUG	G	TEXT	4	Exec debugger present
DOSVER	GS	VERN	4	Default DOS emulator version number
HIGHDMA	GS	HEXV	4	Highest physical address for DMA
INITIAL	GS	TEXT	4	Run initial command SHELL on startup
KBRESET	GS	TEXT	4	Can reset system from console keyboard
LOGIN	GS	TEXT	4	User login required
NUMFLPY	G	DECV	4	Number of floppy disk units in system
NUMHARD	G	DECV	4	Number of hard disk units in system
NUMPAR	G	DECV	4	Number of parallel ports in system
NUMSER	G	DECV	4	Number of serial ports in system
PROINUSE	G	DECV	4	Number of processes/shared sections
PROLIMIT	GS	DECV	4	Limit of number of processes/shared sections
SELINUSE	G	DECV	4	Number of global selectors in use
SELNUM	G	DECV	4	Number of global selectors created
SERNUM	G	DECV	4	Kernel serial number
SPEED	G	DECV	4	Processor speed factor
STATE	GS	HEXV	4	System state
SYSNAME	G	STR	32	Name of system
TOTALMEM	G	DECV	4	Total memory in system (in KB)
USERMEM	G	DECV	4	User memory in system (in KB)
V86BASE	GS	DECV	4	Base address for virtual-86 image (in KB)
V86SIZE	GS	DECV	4	Default size of virtual-86 image (in KB)
XFFINUSE	G	DECV	4	Number of extended fork frames in use
XFFLIMIT	G	DECV	4	Limit of number of extended fork frames
XFFMAX	GS	DECV	4	Maximum extended fork frames in use
XFFNUM	G	DECV	4	Current number of extended fork frames

Table 12.1 - SYSTEM Class Characteristics

Name	Fnc	Format	Size	Description
XMBAMAX	G	DECV	28	Maximum number of available exec buffers
XMBAVAIL	G	DECV	28	Number of available exec buffers
XMBINUSE	G	DECV	28	Number of exec buffers in use
XMBMAX	G	DECV	28	Maximum number of exec buffers in use
XOSVER	G	VERN	4	XOS version number

The following sections describe each of the SYSTEM class characteristics in detail.

AVAILMEM - Available memory

This read-only class characteristic returns the amount of memory (in KB) currently available for allocation. The value must be numeric in decimal format with a length of at least 4 bytes.

DEBUG - Exec debugger present

This read-only class characteristic returns a value of YES if the exec debugger (XDT) is loaded with the kernel or a value of NO if it is not loaded. The value must be text with a length of at least 4 bytes.

DOSVER - DOS emulator version number

This class characteristic sets or returns the default DOS emulator version number. This is the number returned to a DOS program which issues the get version number DOS system call. Currently, it has no other effect. Future versions of XOS may use this value to customize the behavior of the DOS emulator to match specific versions of DOS. The value must be numeric in hex format with a length of at least 2 bytes.

HIGHDMA - Highest physical address for DMA

This class characteristic specifies the highest physical address for DMA transfers. The initial value of this class characteristic is 0xFFFFFFFF, which is the highest possible physical address. This value allows DMA transfers to any memory on the system. This is correct for most 386/486 machines, but a few have hardware limitations such that DMA transfers to memory above a certain address will not work correctly. Some machines will not support DMA transfers to what is referred to as reserved or shadow memory. This is a small amount of memory (between 128KB and 386KB) which is usually physically mapped just below 16MB. If there is a problem with DMA transfers on a given machine, setting this value to

0xF00000 (15MB) will usually correct the problem. Some machines may require a different value.

INITIAL - Run initial command shell on startup

This class characteristic specifies if the command shell should be run on screen 1 on the system's console at startup. This value is only meaningful at startup time. It can be set using a CLSCHAR or SYSCHAR command in the STARTUP.BAT file. The value of this characteristic immediately after STARTUP.BAT is executed is used to determine if a command shell is started. Valid values are YES and NO. The initial value is NO. The value must be text with a length of at least 4 bytes.

KBRESET - Can reset system from console keyboard

This class characteristic specifies if the CTL-ALT-DEL key sequence from the console keyboard causes a system reboot. Valid values are YES and NO. The initial value is YES, which enables the CTL-ALT-DEL key combination to re-boot the system. Changing this value requires the ADMIN privilege. The value must be text with a length of at least 4 bytes.

LOGIN - User login required

This class characteristic specifies if the LOGIN feature will be used. A value of YES indicates that users must specify a user name and password before using the system. A value of NO indicates DOS style operation with no formal login required. The initial value is NO. Setting this value to YES with a command in the STARTUP.BAT file guarantees that no one will be able to access the system without logging in. The value must be text with a length of at least 4 bytes.

NUMFLPY - Number of floppy disk units in system

This read-only class characteristic returns the number of floppy disk in the system. The value must be numeric in decimal format with a length of at least 1 byte.

NUMHARD - Number of hard disk units in system

This read-only class characteristic returns the number of hard disks in the system. The value must be numeric in decimal format with a length of at least 1 byte.

NUMPAR - Number of parallel ports in system

This read-only class characteristic returns the number of parallel ports in the system. The value must be numeric in decimal format with a length of at least 1 byte.

NUMSER - Number of serial ports in system

This read-only class characteristic returns the number of serial ports in the system. The value must be numeric in decimal format with a length of at least 1 byte.

PROINUSE - Number of processes/shared sections

This read-only class characteristic returns the number of processes and shared memory sections currently active in the system. The value must be numeric in decimal format with a length of at least 4 bytes.

PROLIMIT - Limit of number of processes/shared sections

This class characteristic specifies the maximum number of processes and shared memory sections which can be active in the system. Making this value very large does not consume any system resources. It simply serves as a limit to prevent overloading the system with too many processes or shared memory sections. The initial value is 10,000, which is effectively no limit. The value must be numeric in decimal format with a length of at least 4 bytes.

SELINUSE - Number of global selectors in use

This read-only class characteristic returns the number of global selectors currently in use by the system. Global selectors are used by the system when addressing memory and represent a finite, but large, resource. The value must be numeric in decimal format with a length of at least 4 bytes.

SELNUM - Number of global selectors created

This read-only class characteristic returns the number of global selectors which have been created by the system. This value will generally be slightly larger than the SELINUSE value. The value must be numeric in decimal format with a length of at least 4 bytes.

SERNUM - Kernel serial number

This class characteristic returns the kernel serial number for the copy of XOS being used. Each copy of XOS is serialized with a unique serial number. This number is used to verify that multiple systems on a network

are not running from the same copy of XOS. Any attempt to modify this serial number will result in a fatal system error. The value must be numeric in decimal format with a length of at least 4 bytes.

SPEED - Processor speed factor

This read-only class characteristic returns a number which is roughly proportional to processor speed. This value is not intended to be a comprehensive measure of processor speed, but is just a simple measure of the speed of a simple timing loop. It is used internally by the system to calibrate very short time delays. A 16MHz 386DX machine should give a value of about 10. Other speed machines should be roughly proportional, with machines with a cache giving somewhat higher than expected values. The value must be numeric in decimal format with a length of at least 4 bytes.

STATE - System state

This class characteristic sets or returns a value which represents the current state of the system. The value is bit encoded as described in Table 12.2.

Table 12.2 System state values

Bit	Name	Meaning
0	SS$STRTCOMP	System startup complete

Setting or getting this value requires the ADMIN privilege. The value must be numeric in decimal format with a length of at least 4 bytes.

SYSNAME - System name

This read-only class characteristic returns the name of the system. This is a text string which specifies the name and version of the operating system. The value must be a string with a buffer length of at least 32 bytes.

TOTALMEM - Total memory in system

This read-only class characteristic returns the total amount of memory (in KB) in the system. The value must be numeric in decimal format with a length of at least 4 bytes.

USERMEM - User memory in system

This read-only class characteristic returns the total amount of user memory (in KB) in the system. User memory is memory which is not allocated to

the kernel. The value must be numeric in decimal format with a length of at least 4 bytes.

V86BASE - Default base address for virtual-86 image

This class characteristic specifies the default base address for virtual-86 images (in KB). This is the lowest virtual-86 address at which memory is allocated (excluding memory between 0 and 0xFFF, which is always available). This value serves two purposes. First it allows a lower limit to be placed on the memory used by DOS programs. A few DOS programs do not work correctly if loaded too low in memory. Second, it can be used to prevent the use of virtual-86 memory between 0x1000 and 0xFFFF. Use of memory in this area can cause conflicts with I/O devices which use I/O registers between 0x1000 and 0xFFFF on some early steppings of the 80386 (due to a bug in the processor). The initial value of this characteristic is 0. The value must be numeric in decimal format with a length of at least 4 bytes.

V86SIZE - Default maximum size of virtual-86 image

This class characteristic specifies the default maximum size of a virtual-86 image (in KB). This effectively specifies the amount of base memory available to a DOS program running under XOS. The initial value of this characteristic is 640. The value must be numeric in decimal format with a length of at least 4 bytes.

XFFINUSE - Number of extended fork frames in use

This read-only class characteristic returns the number of extended fork frames currently in use by the system. The value must be numeric in decimal format with a length of at least 4 bytes.

XFFLIMIT - Limit of number of extended fork frames

This class characteristic specifies a maximum limit for the number of extended fork frames used by the system. Setting this limit to a very large value does not consume any system resources. This value provides a way to limit the maximum amount of I/O activity in the system. Normally there is no need to use this limit. It is initially set to 10,000, which is effectively no limit. The value must be numeric in decimal format with a length of at least 4 bytes.

XFFMAX - Maximum number of extended fork frames in use

This class characteristic records the maximum number of extended fork frames that have been in use at any one time. The value must be numeric in decimal format with a length of at least 4 bytes.

XFFNUM - Current number of extended fork frames

This read-only class characteristic returns the number of extended fork frames currently in use by the system. The value must be numeric in decimal format with a length of at least 4 bytes.

XMBAMAX - Maximum number of available exec buffers

This read-only class characteristic returns the maximum number of exec buffers which have been available on each of the exec buffer free lists. There are seven exec buffer free lists, corresponding to buffer sizes of 64, 128, 256, 512, 1024, 2048, and 4096 bytes. Note that each of the maximum values is maintained independently. Each number is returned in 4 bytes. The value must be numeric in decimal format with a length of at least 28 bytes.

XMBAVAIL - Number of available exec buffers

This read-only class characteristic returns the number of exec buffers currently on each of the exec buffer free lists. There are seven exec buffer free lists, corresponding to buffer sizes of 64, 128, 256, 512, 1024, 2048, and 4096 bytes. Each number is returned in 4 bytes. The value must be numeric in decimal format with a length of at least 28 bytes.

XMBINUSE - Number of exec buffers in use

This read-only class characteristic returns the number of exec buffers currently in use. Seven numbers are returned, corresponding to exec buffer sizes of 64, 128, 256, 512, 1024, 2048, and 4096 bytes. Each number is returned in 4 bytes. The value must be numeric with decimal format in a length of at least 28 bytes.

XMBMAX - Maximum number of exec buffers in use

This read-only class characteristic returns the maximum number of exec buffers which have been in use. Seven numbers are returned, corresponding to buffer sizes of 64, 128, 256, 512, 1024, 2048, and 4096 bytes. Note that each of the maximum values is maintained independently. Each number is returned in 4 bytes. The value must be numeric in decimal format with a length of at least 28 bytes.

XOSVER - XOS version number

This class characteristic is used to obtain the version number of the system. This is the same version number returned for the SYSNAME but it is returned here in numeric format. The value must be numeric in version number format with a length of at least 4 bytes.

PROCESS Class Characteristics

The PROCESS device class is a dummy device class which exists only to provide a mechanism for accessing and changing various values associated with individual processes in the system. The PROCESS class characteristics are summarized in Table 12.3.

Table 12.3 - PROCESS Class Characteristics

Name	Fnc	Format	Size	Description
CONTRM	GS	TEXT	8	Controlling terminal
NAME	GS	STR	12	Process name
NUM	G	DECV	4	Process number
PRIV	GS	STR	200	Current process privileges
PRIVAVL	GS	STR	200	Available process privileges
SEQ	GV	DECV	4	Sequence number

The PROCESS class is somewhat unusual in that it provides access to data about all processes in the system. Certain of the class characteristics are used to select the process to which any following characteristic applies. At the beginning of class characteristic processing, the process issuing the QFNC_CLASS function is selected.

The following sections describe each of the PROCESS class characteristics in detail.

CONTRM - Controlling terminal

This class characteristic returns the name of the currently selected process's controlling terminal or selects a process based on the terminal name specified. When the value of this characteristic is set, the process which is lowest in the process tree which has the specified terminal as its controlling terminal is selected for further class characteristic processing. If there is more than one process at this level, which one is selected is not defined. The terminal name must be a physical device name specified without a trailing colon. The value must be text with a length of at least 8 bytes.

NAME - Process name

This class characteristic returns or sets the name of the currently selected process. The value must be a string with a buffer length of at least 12 bytes.

NUM - Process number

This class characteristic returns the number of the currently selected process or selects a process based on process number. When the value of this characteristic is set, the process in the system with a matching process number is selected for further class characteristic processing. The process number is a 32-bit value. The high order 16-bits of the value contain the process sequence number and the low order 16-bits contain the process index. If the process sequence number is specified as 0, any process with the specified index is selected. If the process sequence number is specified as a non-0 value, an ER_CHARV error is returned if it does not match the current sequence number of the process whose index was specified. The value must be numeric in decimal format with a length of at least 4 bytes.

PRIV - Current process privileges

This class characteristic returns or sets the current process privileges for the currently selected process. Note that this characteristic does not change process selection. Refer to Chapter 4 for a description of the privilege string. The value must be a string. If getting the current privileges, the buffer length should be at least 200 bytes.

PRIVAVL - Available process privileges

This class characteristic returns or sets the current available process privileges for the currently selected process. Note that this characteristic does not change process selection. Refer to Chapter 4 for a description of the privilege string. Note that privileges cannot be added to the current value. The value must be a string. If getting the current available privileges, the buffer length should be at least 200 bytes.

SEQ - Sequence number

This class characteristic returns the sequence number of the currently selected process or verifies that the sequence number specified matches that of the currently selected process. If it does not match, an ER_CHARV error is returned. The value must be numeric with a length of at least 2 bytes.

DISK Class Characteristics

The DISK device class includes all disk-like devices. These are any devices which support a local file system. Some systems (such as DOS) refer to these as block devices. The DISK class characteristics are summarized in Table 12.4.

Table 12.4 - DISK Class Characteristics

Name	Fnc	Format	Size	Description
AHEAD	GS	DECV	4	Maximum read-ahead blocks
LIMIT	GS	DEVC	4	Maximum number of disk devices allowed
MAXIMUM	GS	DECV	4	Maximum number of in use disk devices
NUMBER	GS	DECV	4	Number of in use disk devices
NUMDBUF	GS	DECV	4	Number of disk data buffers
NUMSBUF	GS	DECV	4	Number of disk system buffers

The following sections describe each of the DISK class characteristics in detail.

AHEAD - Maximum read-ahead blocks

This class characteristic returns or sets the maximum number of blocks which will be read ahead when accessing a disk. This value is a maximum for the system. Individual disks may set a lower read ahead limit. The value must be numeric in decimal format with a length of at least 4 bytes.

LIMIT - Maximum number of disk devices allowed

This class characteristic specifies the maximum number of disk devices which can be in use at any one time. This is effectively the maximum number of files which can be simultaneously open. Setting this characteristic to a very large value does not consume any system resources. It is initially set to 10,000, which is effectively no limit. The value must be numeric in decimal format with a length of at least 4 bytes.

MAXIMUM - Maximum number of in use disk devices

This class characteristic records the maximum number of disk devices in use at any one time. This is effectively the maximum number of simultaneously open files. The value must be numeric in decimal format with a length of at least 4 bytes.

NUMBER - Number of in use disk devices

This read-only class characteristic reports the number of disk devices currently in use. This is effectively the number of currently open files. The value must be numeric in decimal format with a length of at least 4 bytes.

NUMDBUF - Number of disk data buffers

This class characteristic returns the number of disk data buffers in the system or allocates additional disk data buffers. Disk data buffers are used to cache user data read from the disk and for read ahead of user data. If the value is specified, it is first rounded up to a multiple of 7. If this value is larger than the current number of disk data buffers in the system, additional disk data buffers are allocated to increase the total number to the value specified. If this value is equal to or smaller than the current number of disk data buffers in the system, nothing is done. Allocating additional buffers requires the ADMIN privilege. The value must be numeric in decimal format with a length of at least 4 bytes.

NUMSBUF - Number of disk system buffers

This class characteristic returns the number of disk system buffers in the system or allocates additional disk data buffers. Disk system buffers are used to cache system accesses to the disk (such as directory and FAT reads) and to store information about open files. If the value is specified, it is first rounded up to a multiple of 7. If this value is larger than the current number of disk data buffers in the system, additional disk system buffers are allocated to increase the total number to the value specified. If this value is equal to or smaller than the current number of disk system buffers in the system, nothing is done. Allocating additional buffers requires the ADMIN privilege. The value must be numeric in decimal format with a length of at least 4 bytes.

SPL Class Characteristics

The SPL device class includes only the SPL spooled disk devices. There are no class characteristics associated with the SPL device class.

TAPE Class Characteristics

The TAPE device class is the generic class which supports various types of magnetic tape storage devices. The class characteristics for this class are those dealing with the number of devices allowed and the number in use. The NET class characteristics are summarized in Table 12.5.

Table 12.5 - TAPE Class Characteristics

Name	Fnc	Format	Size	Description
LIMIT	GS	DECV	4	Maximum number of TAPE devices allowed
NUMBER	G	DECV	4	Number of in use TAPE devices
MAXIMUM	GS	DECV	4	Maximum number of in use TAPE devices

The following sections describe each of the TAPE class characteristics in detail.

LIMIT - Maximum number of network devices allowed

This class characteristic specifies the maximum number of TAPE devices which can be in use at any one time. The value must be numeric in decimal format with a length of at least 4 bytes.

NUMBER - Number of in use TAPE devices

This read-only class characteristic returns the number of network devices currently in use in the system. This is the number of open NETWORK devices. The value must be numeric in decimal format with a length of at least 4 bytes.

MAXIMUM - Maximum number of in use TAPE devices

This class characteristic records the maximum number of TAPE devices in use at any one time. This is the maximum number of simultaneously open TAPE devices. The value must be numeric in decimal format with a length of at least 4 bytes.

TRM Class Characteristics

The TRM device class includes all terminal-like devices. These include the console/keyboard device and all serial port devices (except for the serial network driver). The TRM class characteristics are summarized in Table 12.6.

Table 12.6 - TRM Class Characteristics

Name	Fnc	Format	Size	Description
LIMIT	G	DECV	4	Maximum number of terminal devices allowed
MAXIMUM	GS	DECV	4	Maximum number of in use terminal devices
NUMBER	G	DECV	4	Number of in use terminal devices

The following sections describe each of the TRM class characteristics in detail.

LIMIT - Maximum number of terminal devices allowed

This class characteristic specifies the maximum number of terminal devices which can be in use at any one time. The value must be numeric in decimal format with a length of at least 4 bytes.

MAXIMUM - Maximum number of in use terminal devices

This class characteristic records the maximum number of terminal devices in use at any one time. This is the maximum number of simultaneously open terminal devices. The value must be numeric in decimal format with a length of at least 4 bytes.

NUMBER - Number of in use terminal devices

This read-only class characteristic returns the number of terminal devices currently in use in the system. This is the number of open terminal devices. The value must be numeric in decimal format with a length of at least 4 bytes.

PCN Class Characteristics

The PCN device class includes all serverside psuedoconsole devices. The PCN class characteristics are summarized in Table 12.7.

Table 12.7 - PCN Class Characteristics

Name	Fnc	Format	Size	Description
LIMIT	G	DECV	4	Maximum number of pseudo-console devices allowed
MAXIMUM	GS	DECV	4	Maximum number of in use pseudo-console devices
NUMBER	G	DECV	4	Number of in use pseudo-console devices

The following sections describe each of the PCN class characteristics in detail.

LIMIT - Maximum number of pseudo-console devices allowed

This class characteristic specifies the maximum number of pseudo-console devices which can be in use at any one time. The value must be numeric in decimal format with a length of at least 4 bytes.

MAXIMUM - Maximum number of in use pseudo-console devices

This class characteristic records the maximum number of pseudo-console devices in use at any one time. This is the maximum number of simultaneously open pseudo-console devices. The value must be numeric in decimal format with a length of at least 4 bytes.

NUMBER - Number of in use pseudo-console devices

This read-only class characteristic returns the number of pseudo-console devices currently in use in the system. This is the number of open pseudo-console devices. The value must be numeric in decimal format with a length of at least 4 bytes.

IPM Class Characteristics

The IPM device class includes only the IPM (InterProcess Message) device. This device is used for sending messages (datagrams) between processes on the same system. Since this function is implemented as a device, it allows the use of the standard device oriented system calls for interprocess communication, eliminating the need for a special set of system calls for this purpose. The IPM class characteristics are summarized in Table 12.8.

Table 12.8 - IPM Class Characteristics

Name	Fnc	Format	Size	Description
LIMIT	GS	DECV	4	Maximum number of interprocess message devices allowed
NUMBER	G	DECV	4	Number of in use interprocess message devices
MAXIMUM	G	DECV	4	Maximum number of in use interprocess message devices

The following sections describe each of the IPM class characteristics in detail.

LIMIT - Maximum number of interprocess message devices allowed

This class characteristic specifies the maximum number of interprocess message devices which can be in use at any one time. The value must be numeric in decimal format with a length of at least 4 bytes.

NUMBER - Number of in use interprocess message devices

This read-only class characteristic returns the number of interprocess message devices currently in use in the system. This is the number of open interprocess message devices. The value must be numeric in decimal format with a length of at least 4 bytes.

MAXIMUM - Maximum number of in use interprocess message devices

This class characteristic records the maximum number of interprocess message devices in use at any one time. This is the maximum number of simultaneously open interprocess message devices. The value must be numeric in decimal format with a length of at least 4 bytes.

NULL Class Characteristics

The NULL device class includes only the NULL device. This is a dummy device which discards all output written to it and always indicates end of file on input. There are no class characteristics associated with the NULL device class.

PPR Class Characteristics

The PPR device class includes only the PPRn device. This is the generic parallel printer device which supports printers connected to a parallel port. There are no class characteristics associated with the PPR device class.

NET Class Characteristics

The NET device class is the generic network device class. NET devices can be used to directly access the network interfaces, but most actual use is through various other associated devices. The class characteristics for this class are those dealing with the number of devices allowed and the number in use. The NET class characteristics are summarized in Table 12.9.

Table 12.9 - NET Class Characteristics

Name	Fnc	Format	Size	Description
LIMIT	GS	DECV	4	Maximum number of network devices allowed
NUMBER	G	DECV	4	Number of in use network devices
MAXIMUM	GS	DECV	4	Maximum number of in use network devices

The following sections describe each of the NET class characteristics in detail.

LIMIT - Maximum number of network devices allowed

This class characteristic specifies the maximum number of network devices which can be in use at any one time. The value must be numeric in decimal format with a length of at least 4 bytes.

NUMBER - Number of in use network devices

This read-only class characteristic returns the number of network devices currently in use in the system. This is the number of open network devices. The value must be numeric in decimal format with a length of at least 4 bytes.

MAXIMUM - Maximum number of in use network devices

This class characteristic records the maximum number of network devices in use at any one time. This is the maximum number of simultaneously open network devices. The value must be numeric in decimal format with a length of at least 4 bytes.

SNAP Class Characteristics

The SNAP device class provides access to the link control level for the Internet protocol stack using the SNAP or Bluebook link level protocols. The class characteristics for this class are those dealing with the number of devices allowed and the number in use. The SNAP class characteristics are summarized in Table 12.10.

Table 12.10 - NET Class Characteristics

Name	Fnc	Format	Size	Description
LIMIT	GS	DECV	4	Maximum number of SNAP devices allowed
NUMBER	G	DECV	4	Number of in use SNAP devices
MAXIMUM	GS	DECV	4	Maximum number of in use SNAP devices

The following sections describe each of the SNAP class characteristics in detail.

LIMIT - Maximum number of SNAP devices allowed

This class characteristic specifies the maximum number of SNAP devices which can be in use at any one time. The value must be numeric in decimal format with a length of at least 4 bytes.

NUMBER - Number of in use SNAP devices

This read-only class characteristic returns the number of SNAP devices currently in use in the system. This is the number of open SNAP devices. The value must be numeric in decimal format with a length of at least 4 bytes.

MAXIMUM - Maximum number of in use SNAP devices

This class characteristic records the maximum number of SNAP devices in use at any one time. This is the maximum number of simultaneously open SNAP devices. The value must be numeric in decimal format with a length of at least 4 bytes.

ARP Class Characteristics

The ARP device class provides access to the ARP protocol level for the Internet protocol stack. The class characteristics for this class are those dealing with the number of devices allowed and the number in use. The ARP class characteristics are summarized in Table 12.11.

Table 12.11 - ARP Class Characteristics

Name	Fnc	Format	Size	Description
LIMIT	GS	DECV	4	Maximum number of ARP devices allowed
NUMBER	G	DECV	4	Number of in use ARP devices
MAXIMUM	GS	DECV	4	Maximum number of in use ARP devices

The following sections describe each of the ARP class characteristics in detail.

LIMIT - Maximum number of ARP devices allowed

This class characteristic specifies the maximum number of ARP devices which can be in use at any one time. The value must be numeric in decimal format with a length of at least 4 bytes.

NUMBER - Number of in use ARP devices

This read-only class characteistic returns the number of ARP devices currently in use in the system. This is the number of open ARP devices. The value must be numeric in decimal format with a length of at least 4 bytes.

MAXIMUM - Maximum number of in use ARP devices

This class characteristic records the maximum number of ARP devices in use at any one time. This is the maximum number of simultaneously open ARP devices. The value must be numeric in decimal format with a length of at least 4 bytes.

IPS Class Characteristics

The IPS device class includes only the IPSn devices. This is a network device which provides access to the IP protocol level. There are no class characteristics associated with the IPS device class.

The IPS device class provides access to the IP level of the Internet protocol stack. The class characteristics for this class are those dealing with the number of devices allowed and the number in use. The IPS class characteristics are summarized in Table 12.12.

Table 12.12 - IPS Class Characteristics

Name	Fnc	Format	Size	Description
LIMIT	GS	DECV	4	Maximum number of IPS devices allowed
NUMBER	G	DECV	4	Number of in use IPS devices
MAXIMUM	GS	DECV	4	Maximum number of in use IPS devices

The following sections describe each of the IPS class characteristics in detail.

LIMIT - Maximum number of IPS devices allowed

This class characteristic specifies the maximum number of IPS devices which can be in use at any one time. The value must be numeric in decimal format with a length of at least 4 bytes.

NUMBER - Number of in use IPS devices

This read-only class characteristic returns the number of IPS devices currently in use in the system. This is the number of open IPS devices. The value must be numeric in decimal format with a length of at least 4 bytes.

MAXIMUM - Maximum number of in use IPS devices

This class characteristic records the maximum number of IPS devices in use at any one time. This is the maximum number of simultaneously open IPS devices. The value must be numeric in decimal format with a length of at least 4 bytes.

UDP Class Characteristics

The UDP device class provides access to the UDP protocol level of the Internet protocol stack. The class characteristics for this class are those dealing with the number of devices allowed and the number in use. The UDP class characteristics are summarized in Table 12.13.

Table 12.13 - UDP Class Characteristics

Name	Fnc	Format	Size	Description
LIMIT	GS	DECV	4	Maximum number of UDP devices allowed
NUMBER	G	DECV	4	Number of in use UDP devices
MAXIMUM	GS	DECV	4	Maximum number of in use UDP devices

The following sections describe each of the UDP class characteristics in detail.

LIMIT - Maximum number of UDP devices allowed

This class characteristic specifies the maximum number of UDP devices which can be in use at any one time. The value must be numeric in decimal format with a length of at least 4 bytes.

NUMBER - Number of in use UDP devices

This read-only class characteristic returns the number of UDP devices currently in use in the system. This is the number of open UDP devices. The value must be numeric in decimal format with a length of at least 4 bytes.

MAXIMUM - Maximum number of in use UDP devices

This class characteristic records the maximum number of UDP devices in use at any one time. This is the maximum number of simultaneously open UDP devices. The value must be numeric in decimal format with a length of at least 4 bytes.

TCP Class Characteristics

The TCP device class provides access to the TCP protocol level of the Internet protocol stack. The class characteristics for this class are those dealing with the number of devices allowed and the number in use. The TCP class characteristics are summarized in Table 12.14.

Table 12.14 - TCP Class Characteristics

Name	Fnc	Format	Size	Description
LIMIT	GS	DECV	4	Maximum number of TCP devices allowed
NUMBER	G	DECV	4	Number of in use TCP devices
MAXIMUM	GS	DECV	4	Maximum number of in use TCP devices

The following sections describe each of the TCP class characteristics in detail.

LIMIT - Maximum number of TCP devices allowed

This class characteristic specifies the maximum number of TCP devices which can be in use at any one time. The value must be numeric in decimal format with a length of at least 4 bytes.

NUMBER - Number of in use TCP devices

This read-only class characteristic returns the number of TCP devices currently in use in the system. This is the number of open TCP devices. The value must be numeric in decimal format with a length of at least 4 bytes.

MAXIMUM - Maximum number of in use TCP devices

This class characteristic records the maximum number of TCP devices in use at any one time. This is the maximum number of simultaneously open TCP devices. The value must be numeric in decimal format with a length of at least 4 bytes.

TLN Class Characteristics

The TLN device class implements an embedded Telnet server.This device cannot do IO directly, but it can be used to read or set Telnet server parameters and characteristics. There are no class characteristics associated with the TLN device class.

RCP Class Characteristics

The RCP device class provides access to the RCP protocol level of the Internet protocol stack. The class characteristics for this class are those dealing with the number of devices allowed and the number in use. The RCP class characteristics are summarized in Table 12.15.

Table 12.15 - RCP Class Characteristics

Name	Fnc	Format	Size	Description
LIMIT	GS	DECV	4	Maximum number of RCP devices allowed
NUMBER	G	DECV	4	Number of in use RCP devices
MAXIMUM	GS	DECV	4	Maximum number of in use RCP devices

The following sections describe each of the RCP class characteristics in detail.

LIMIT - Maximum number of RCP devices allowed

This class characteristic specifies the maximum number of RCP devices which can be in use at any one time. The value must be numeric in decimal format with a length of at least 4 bytes.

NUMBER - Number of in use RCP devices

This read-only class characteristic returns the number of RCP devices currently in use in the system. This is the number of open RCP devices. The value must be numeric in decimal format with a length of at least 4 bytes.

MAXIMUM - Maximum number of in use RCP devices

This class characteristic records the maximum number of RCP devices in use at any one time. This is the maximum number of simultaneously open RCP devices. The value must be numeric in decimal format with a length of at least 4 bytes.

XFP Class Characteristics

The XFP device class implements a client which uses the XFP protocol to provide remote file access. The class characteristics for this class are those dealing with the number of devices allowed and the number in use. The XFP class characteristics are summarized in Table 12.16.

Table 12.16 - TCP Class Characteristics

Name	Fnc	Format	Size	Description
LIMIT	GS	DECV	4	Maximum number of XFP devices allowed
NUMBER	G	DECV	4	Number of in use XFP devices
MAXIMUM	GS	DECV	4	Maximum number of in use XFP devices

The following sections describe each of the XFP class characteristics in detail.

LIMIT - Maximum number of XFP devices allowed

This class characteristic specifies the maximum number of XFP devices which can be in use at any one time. The value must be numeric in decimal format with a length of at least 4 bytes.

NUMBER - Number of in use XFP devices

This read-only class characteristic returns the number of XFP devices currently in use in the system. This is the number of open XFP devices. The value must be numeric in decimal format with a length of at least 4 bytes.

MAXIMUM - Maximum number of in use XFP devices

This class characteristic records the maximum number of XFP devices in use at any one time. This is the maximum number of simultaneously open XFP devices. The value must be numeric in decimal format with a length of at least 4 bytes.

Chapter 13

Device Characteristics

This chapter describes the device characteristics for all standard devices in the system, included those implemented with loadable drivers. A device characteristic is a named item which specifies a value associated with a specific device. It is similar to a class characteristic except that a class characteristic represents a value associated with a device class as a whole, rather than with an individual device.

Device characteristics are mainly used to access permanent values, that is, items which retain their values when devices are opened and closed by programs. There are a few exceptions to this, where values are only effective until the next time the device is idle, and these are noted in the descriptions which follow.

Device characteristics provide a standardized interface which applies to all devices. It is a very flexible interface which is easily adaptable to virtually any new device type. Indeed, this degree of flexibility is such that it is difficult to summarize how device characteristics are used. This is best understood by reading the following descriptions of how individual XOS devices use device characteristics.

Device characteristics are accessed using the QFNC_DEVCHAR function of the svcIoQueue system call, which is described in Chapter 16. A DEVCHAR command is provided (which is described in the XOS User's Guide), which provides access to the device characteristics values from the command line.

This chapter lists the device characteristics by device class, and where necessary, by device type within a class. Technically, there is no relationship between characteristics for different devices which have the same name, but in practice all characteristics with the same names represent roughly equivalent values for their respective devices. There is one device characteristic that is defined for all devices and a small number that are defined for nearly all devices. Programmers implementing new devices are strongly encouraged to use the established names, but only in the same way they are used for the standard devices.

It should be noted in passing that the characteristics passed to the CLSF_ADDUNIT sub-function of the QFNC_CLASS svcIoQueue function bear a strong resemblance to device characteristics. While they have a different name-space from the device characteristics for a device, they often perform similar functions, initializing values when a device is first created. Generally, the add unit characteristics specify values that must be specified when the device is created and cannot be changed later. The standard XOS devices define device characteristics with the same names as these add unit characteristics (such as IOREG) which can be used to obtain the values used when the device was created. Programmers implementing new devices are strongly encouraged to continue this practice.

Device characteristics are specified using a device characteristic list. This is a data structure consisting of a sequence of device characteristic items followed by a byte containing 0. Each device characteristic item consists of a 10 byte header followed by a 0 to 32 byte value.

The first header byte specifies the function to perform and the value format. It is described in detail in figure 13.1.

Figure 13.1 First Characteristic Header Byte

Bit	Name	Set by	Meaning		
7	SET	user	1 if should set value of characteristic		
6	GET	user	1 if should get value of characteristic		
5	ERROR	system	1 if error associated with this characteristic		
4			Not used		
3-0	FORMAT	user	Value format as follows:		
			Value	Name	Meaning
				0	Not used
			1	DECV	Decimal value
			2	HEXV	Hex value
			3	OCTV	Octal value
			4	BINV	Binary value
			5	DECB	Decimal bytes
			6	HEXB	Hex bytes
			7	OCTB	Octal bytes
			8	VERN	Version number
			9	TIME	Time value
			10	DATE	Date value
			11	DT	Date/time value
			12	DATAB	Data bytes
			13	DATAS	Data string
			14	TEXT	Test bytes
			15	STR	String value

The value format field provides a fairly detailed specification of the type of value associated with the parameter. The current version of XOS does not differentiate between types 1 through 11, but simply treats them as general numeric values. Any of these values can be specified whenever a numeric value is called for. It is suggested that some effort be made to match the type with the specific type of value actually used, however, since future versions of XOS may use this information. Value types of 1 through 12 and 14 indicate a value which is stored in the value field which immediately follows the header. Value types of 13 and 15 indicate a value which is stored in a buffer which is specified by a buffer descriptor which immediately follows the header.

The C language symbols for the function bits are generated by prefixing PAR_ to the names given in the table above. The assembler names are generated by prefixing PAR$. The C and assembler names for the value formats are generated by prefixing REP_ to the name given above.

The second header byte contains the length of the value, in bytes. If a shorter value is specified than is needed when setting a characteristic value, the value is zero extended to the necessary length. If a shorter value field than is expected is specified when getting a value and the actual value to be returned will fit in the field specified without truncation, the value is stored. If the value would be truncated, an ER_CHARS error is returned for the I/O operation the PAR$ERROR bit is set in the characteristic. When the value type is FMT_STR, the second byte is not used and should contain 0. In this case, the value field consists of an 8 byte far pointer (high 2 bytes of the selector part are not used) which points to a buffer containing the string value (if PAR$SET set) or to receive the string value (if PAR$GET set), followed by two 2 byte values. The first of these values specifies the length of the buffer and the second specifies the length of the string in the buffer. The buffer length value must be set by the caller before issuing the system call. The string length value must be set by the caller if PAR$SET is specified. It is set by the system to indicate the actual length of the string returned if PAR$GET is specified.

The third through tenth bytes of the header contain the ASCII name of the characteristic. It is not case sensitive. If the name is shorter than 8 characters, the field is filled with NULL characters.

Device characteristics are accessed using the QFNC_DEVCHAR function of the svcIoQueue system call, which is described in chapter 15. A DEVCHAR command is provided (which is described in the XOS User's Guide), which provides access to the device characteristics values from the command line.

This chapter contains a number of tables which summarize the device characteristics for the various devices. These tables all have the same format. The first column specifies the name of the characteristic (1 to 8 characters, not case sensitive but listed by convention as upper case). The second column specifies the format of the value. This corresponds to the PAR$FORMAT value listed in Table 13-1, above, without the FMT_ prefix. The third column specifies the minimum value length which can contain all possible values of the characteristic. The fourth column may contain the letters S or V to indicate that the value of the characteristic may be set or verified (PAR$SET set in the first byte of the characteristic). S indicates that the value may be set. V indicates that a set operation is allowed but that it will verify, rather than change the value. Get operations are always valid.

It should be noted that the interpretation of the format is exactly the same as for class characteristics (see Chapter 12) and is the same as for device parameters (see Chapter 11), except that the type of numeric format is significant here. A sub-function of the QFNC_DEVCHAR function allows a program to obtain the format of a charac-

teristic, which can then be used to format the value for output. This feature is used by the DEVCHAR command to display each value in the proper format.

Table 13.2 summarizes the device characteristics which are common to most XOS devices. Not all devices use all of these characteristics, but those that do always use them as described here.

Table 13.2 - Common device characteristics

Name	Fnc	Format	Size	Description
CLASS	V	TEXT	8	Device class
INDEX	G	DECV	1	Index of unit on controller
IOREG	G	HEXV	2	Base I/O register number
TYPE	G	TEXT	4	Device type

The following describes these common device characteristics in detail.

CLASS - Device class

This device characteristic is used by every XOS device. It returns the name of the device class to which the device belongs. When a value is specified for this characteristic, that value is compared to the name of the device class for the device. If they are the same, no further action is taken. If they are different, an ER_CHARV error is returned. This allows a program to verify that a given device is really a member of the class expected. This is useful since many devices implement device dependent functions which only behave as expected for a specific class device. The value must be text with a length of at least 8 bytes.

INDEX - Index of unit on controller

This device characteristic is used by devices which support multiple device units connected to a single controller. This characteristic returns the index of a device on its controller. Since this value generally cannot be changed after the device is created, attempting to set the value of this characteristic results in an ER_CHARF error. The value must be numeric im decimal format with a length of at least one byte.

IOREG - Base I/O register number

This device characteristic is used by all devices which are associated with a particular set of hardware I/O registers. The value of this characteristic is the number of the base I/O register for the device. Devices which require specification of additional I/O register numbers define additional characteristics for this purpose. Since this value generally cannot be changed af-

ter the device is created, attempting to set the value of this characteristic results in an ER_CHARF error. The value must be numeric in hex format with a length of at least two bytes.

TYPE - Device type

This device characteristic is used by devices which use different drivers to support different types of hardware. Its value specifies the specific type of driver being used for the device. For example, XOS supports both the standard AT type floppy controller and the Compati-Card floppy controller. The value of this characteristic indicates which is being used for a specific floppy drive. Since this value generally cannot be changed after the device is created, attempting to set the value of this characteristic results in an ER_CHARF error. The value must be text with a length of at least four bytes.

The remainder of this chapter describes the device characteristics used by the standard XOS devices.

DISK Device Characteristics

Device characteristics for DISK class devices generally refer to the underlying physical disk or disk partition, even if the device is really a file on a file structured disk. There are a number of device characteristics which are used by all DISK class devices and there are a smaller number which are specific to individual types of disk (hard disks or floppy disks). The common characteristics for disks are summarized in table 13.3. NOTE: Not all disks have all of these characteristics but more than one type of disk can have these.

Table 13.3 - Device characteristics for DISK class devices

Name	Fnc	Format	Size	Description
AVAIL	G	DECV	4	Number of free clusters
BLOCKIN	GS	DECV	4	Number of blocks input
BLOCKOUT	GS	DECV	4	Number of blocks output
BYTEIN	GS	DECV	4	Number of bytes input
BYTEOUT	GS	DECV	4	Number of bytes output
CBLKSZ	G	DECV	2	Current block size in bytes
CBLOCKS	G	DECV	4	Current total number of blocks on disk
CCYLNS	GS	DECV	4	Current number of cylinders
CHEADS	GS	DECV	1	Current number of heads
CLASS	G	TEXT	8	Device class
CLSSZ	G	DECV	4	Cluster size
CLUSTERS	G	DECV	4	Total number of clusters
CSECTS	G	DECV	4	Current number of sectors per track
DOSNAMEn	GS	TEXT	16	DOS name (drive letter) for unit n
DTHLIMIT	G	DECV	4	Data transfer hardware limit
FATMODE	G	HEXV	1	DOS FAT mode byte
FSTYPE	G	TEXT	8	File structure type
HDATAERR	GS	DECV	4	Number of hard data errors
HDEVERR	GS	DECV	4	Number of hard device errors
HIDFERR	GS	DECV	4	Number of hard ID field errors
HOVRNERR	GS	DECV	4	Number of hard DMA overrun errors
HRNFERR	GS	DECV	4	Number of hard record not found errors
HSEEKERR	GS	DECV	4	Number of hard seek errors
HUNGERR	GS	DECV	4	Number of device time-outs
IBLKSZ	GS	DECV	2	Initial block size in bytes
IBLOCKS	G	DECV	4	Initial total number of blocks on disk
ICYLNS	GS	DECV	4	Initial number of cylinders

Table 13.3 - Device characteristics for DISK class devices

Name	Fnc	Format	Size	Description
IHEADS	GS	DECV	1	Initial number of heads
ISECTS	GS	DECV	4	Initial number of sectors per track
MODEL	G	STR	42	Device model description
MSENSOR	G	TEXT	4	Drive has media sensor
PARTN	G	HEXV	1	Partition table index
PARTOFF	G	DECV	4	Partition offset
PROTECT	GS	TEXT	4	File access protection
RAMAX	GS	DECV	4	Maximum read-ahead blocks
REMOVE	G	TEXT	4	Media is removable
REVISION	G	STR	10	Device revision number
ROOTBLK	G	DECV	4	First block for root directory
ROOTPROT	G	STR	100	Root directory protection
ROOTSIZE	G	DECV	4	Size of root directory
SERIALNO	G	STR	22	Device serial number or other data
SHRDELAY	GS	DECV	4	File sharing delay factor
SHRFAIL	GS	DECV	4	Number of file sharing failures
SHRRETRY	GS	DECV	4	Number of file sharing retries
TDATAERR	GS	DECV	4	Total number of data errors
TDEVERR	GS	DECV	4	Total number of device errors
TIDFERR	GS	DECV	4	Total number of ID field errors
TOVRNERR	GS	DECV	4	Total number of DMA overrun errors
TRNFERR	GS	DECV	4	Total number of record not found errors
TSEEKERR	GS	DECV	4	Total number of seek errors
UNITTYPE	G	TEXT	4	Unit type
VOLCDT	G	DT	8	Volume creation date and time
VOLEDT	G	DT	8	Volume effective date and time
VOLLABEL	G	STR	34	Volume label
VOLMDT	G	DT	8	Volume modification date and time
VOLNAME	GS	TEXT	16	Volume name
VOLXDT	G	DT	8	Volume expriation date and time
WTMAX	GS	DECV	4	Write transfer limit

The common disk device characteristics are described in detail below.

AVAIL - Number of clusters available

This read-only device characteristic returns the number of available clusters for a file structured disk. If the disk is not file structured, the value returned is always 0.

BLOCKIN - Number of blocks input

This read-write device characteristic returns the number of blocks written to the disk.

BLOCKOUT - Number of blocks output

This read-write device characteristic returns the number of blocks read from the disk.

BYTEIN - Number of bytes input

This read-write device characteristic returns the number of bytes written to the disk.

BYTEOUT - Number of bytes output

This read-write device characteristic returns the number of bytes read from the disk.

CBLKSZ - Current block size

This device characteristic specifies the current block or sector size, in bytes. File structured disks require a block size of 512 bytes, so this value cannot be changed for file structured disks or for disk accessed using the XOS disk cache. It can be changed, however, for non-file structured disks open in raw mode. This allows for access of non-standard physical disk formats. This is particularly useful when reading certain non-DOS floppy disks. There are some limitations imposed by the disk controllers on block size. Controllers for hard disk usually do not allow the block size to be changed at all. The standard floppy controllers require an even power of 2 equaling between 128 and 4096. A value specified for this characteristic is in effect until a mount operation is done, when the value reverts back to the value of the DBLKSZ characteristic.

CBLOCKS - Current toltal number of blocks on disk

This read-only device characteristic returns the number of available blocks for a file structured disk. If the disk is not file structured, the value returned is the total number of blocks for the physical drive.

CCYLNS - Current number of cylinders

This device characteristic specifies the current number of cylinders on the disk. This value is set from the parameters of the file structure for file structured disks and cannot be changed for such disks. It can be changed for non-file structured disks. This should be done with care, however,

since some disks can be physically damaged if repeatedly positioned beyond the last cylinder on the disk. A value specified for this characteristic is in effect until a mount operation is done, when the value reverts back to the value of the DCYLNS characteristic.

CHEADS - Current number of heads

This device characteristic specifies the current number of heads on the disk. This value is set from the parameters of the file structure for file structured disks and cannot be changed for such disks. It can be changed for non-file structured disks. This is particularly useful when attempting to read a single sided floppy disk in a normal double sided floppy drive. A value specified for this characteristic is in effect until a mount operation is done, when the value reverts back to the value of the DHEADS characteristic.

CLASS - Device class

This device characteristic returns the name of the device class to which the device belongs. When a value is specified for this characteristic, that value is compared to the name of the device class for the device. If they are the same, no further action is taken.

CLSSZ - Cluster size

This read-only device characteristic reports the cluster size (in blocks) for a file structured disk. A cluster is the smallest amount of space allocated in the file system. For DOS and XOS file systems, it is always a power of 2, usually 1 to 8 blocks. A value of 0 is reported for non-file structured disks.

CLUSTERS - Total number of clusters

This read-only device characteristic reports the total number of clusters on a file structured disk. A value of 0 is reported for non-file structured disks.

CSECTS - Current number of sectors

This device characteristic specifies the current number of sectors per track on the disk. This value is set from the parameters of the file structure for file structured disks and cannot be changed for such disks. It can be changed for non-file structured disks. A value specified for this characteristic is in effect until a mount operation is done, when the value reverts back to the value of the DSECTS characteristic.

DOSNAME - DOS disk name

This device characteristic specifies the DOS name for the disk. This name provides an alternate name for the disk. Normally this is used to specify the equivalent DOS disk name (a single letter) for DOS compatibility. When a program requests that the DOS name of a device be returned, this name is returned.

DOSNAMEn - DOS name (drive letter) for unit/partition n

This device characteristic specifies the DOS name for the disk unit or partition where n is the partion number. This name provides an alternate name for the disk. Normally this is used to specify the equivalent DOS disk name (a single letter) for DOS compatibility. When a program requests that the DOS name of a device be returned, this name is returned. This is used for disks that have removeable media, where there may be different numbers of partions on each removeable disk. There is no error reported for setting more partition names than the disk actualy has. Only if the drive letter is used will an error be reported.

DTHLIMIT - Data transfer hardware limit

This device characteristic specifies the harware limit in blocks that this disk device can transfer in one operation.

FATMODE - FAT mode byte (DOS file system)

This read-only device characteristic reports the value of the FAT mode byte for DOS file systems. A value of 0 is returned for disks which do not contain a DOS file system.

FSTYPE - File structure type

This read-only device characteristic reports the type of file system mounted on the device.

Possible values are:

Value	Meaning
XOS	XOS native file system
DOS12	DOS file system with 12 bit FAT entries
DOS16	DOS file system with 16 bit FAT entries
DOSEXT	DOS extended partition
DOSHP	DOS huge partition
DSS12	DOS SpeedStor compatible with 12 bit FAT entries
DSS12L	DOS speedStor extended with 12 bit FAT entries
DSS16	DOS SpeedStor compatible with 16 bit FAT entries
DSS16L	DOS SpeedStor extended with 16 bit FAT entires

HDATAERR - Number of hard data errors

This device characteristic specifies the number of hard (non revoverable) data read errors that have occured on this disk.

HDEVERR - Number of hard device errors

This device characteristic specifies the number of hard (non revoverable) device access errors that have occured on this disk.

HIDFERR - Number of hard ID field errors

This device characteristic specifies the number of hard (non revoverable) disk ID field read errors that have occured on this disk.

HOVRNERR - Number of hard DMA overrun errors

This device characteristic specifies the number of hard (non revoverable) DMA receive errors that have occured on this disk.

HRNFERR - Number of hard record not found errors

This device characteristic specifies the number of hard (non revoverable) record not found errors that have occured on this disk.

HSEEKSRR - Number of hard seek errors

This device characteristic specifies the number of hard (non revoverable) seek errors that have occured on this disk.

HUNGERR - Numner of device time-outs

This device characteristic specifies the number of hard (non revoverable) disk access timeout errors that have occured on this disk.

IBLKSZ - Default block size

This device characteristic specifies the default block or sector size, in bytes. This value is only used when a disk is mounted. At this time it is copied to the value of the CBLKSZ characteristic to provide a default initial value. Note that for file structured disks, this value is immediately replaced by the value supplied by the parameters of the file system. The value must be numeric in decimal format with a length of at least 2 bytes.

IBLOCKS - Initial total number of blocks on disk

This device characteristic specifies the initial total number of blocks for this disk.

ICYLNS - Default number of cylinders

This device characteristic specifies the default number of cylinders. This value is only used when a disk is mounted. At this time it is copied to the value of the CCYLNS characteristic to provide a default initial value. Note that for file structured disks, this value is immediately replaced by the value supplied by the parameters of the file system. The value must be numeric in decimal format with a length of at least 2 bytes.

IHEADS - Default number of heads

This device characteristic specifies the default number of heads. This value is only used when a disk is mounted. At this time it is copied to the value of the CHEADS characteristic to provide a default initial value. Note that for file structured disks, this value is immediately replaced by the value supplied by the parameters of the file system. The value must be numeric in decimal format with a length of at least 2 bytes.

ISECTS - Initial number of sectors per track

This device characteristic specifies the default number of sectors. This value is only used when a disk is mounted. At this time it is copied to the value of the CSECTS characteristic to provide a default initial value. Note that for file structured disks, this value is immediately replaced by the value supplied by the parameters of the file system. The value must be numeric in decimal format with a length of at least 4 bytes.

MODEL - Device model description

This is a text string returned by the disk giving the manufacturers description.

MSENSOR - Disk has media sensor

This read-only device characteristic reports if the disk has a media sensor. This is only meaningful for removable disks. Possible values are YES and NO. Fixed disks always return NO. The value must be text with a length of at least 4 bytes.

PARTN - Partition number

This read-only device characteristic reports the partition number for a hard disk partition. This is the index (1 based) for the partition table entry for the partition. If the partition is in an extended partition, the index for the entry of the extended partition ored with 0x80 is returned. If the device is not a partition, a value of 0 is returned. The value must be numeric in hex format, with a length of at least 1 byte.

PARTOFF - Partition table offset

This read-only device characteristic reports the block offset for a disk partition. This is the block number on the underlying disk for the first block of the partition. The value must be numeric in decimal format, with a length of at least 4 bytes.

PROTECT - File access protection

This device characteristic specifies YES or NO to indicated if file access protection is enabled.

RAMAX - Maximum read-ahead blocks

Specifies the maximum number of blocks to read ahead into the disk cache when a program requests a block of data not currently in the cache.

REMOVE - removable

This device characteristic specifies if a disk is removable or fixed. Possible values are YES (removable) and NO (fixed). The value of this characteristic cannot be changed for hard disks and is always NO. The value must be text with a length of at least 4 bytes.

REVISION - Device revision number

This is a text string returned by the disk giving the manufacturers revision number for the drive.

ROOTBLK - First block for root directory

This device characteristic specifies the starting block on disk for the root directory.

ROOTPROT - Root directory protection

This device characteristic specifies the root directory protection status.

ROOTSIZE - Size of root directory

This device characteristic specifies the total number of blocks used by the root directory.

SERIALNO - Device serial number or other data

This read-only device characteristic returns the hard disk controller's serial number. If the controller does not report this information, a null string is returned. The value must be a string with a buffer length of at least 22 bytes.

SHRDELAY - File sharing delay factor

This device characteristic specifies time interval for retrying access to a locked file or region.

SHRFAIL - Number of file sharing failures

This device characteristic specifies the number of failures in access to a locked file or region.

SHRRETRY - Number of file sharing retries

This device characteristic specifies the number of times to retry access to a locked file or region.

TDATAERR - Total number of data errors

This device characteristic specifies the total number of hard (non revoverable) data read errors that have occured on this disk.

TDEVERR - Total number of device errors

This device characteristic specifies the total number of hard (non revoverable) device access errors that have occured on this disk.

TIDFERR - Total number of ID field errors

This device characteristic specifies the total number of hard (non revoverable) disk ID field read errors that have occured on this disk.

TOVRNERR - Total number of DMA overrun errors

This device characteristic specifies the total number of hard (non revoverable) DMA receive errors that have occured on this disk.

TRNFERR - Total number of record not found errors

This device characteristic specifies the total number of hard (non revoverable) record not found errors that have occured on this disk.

TSEEKERR - Total number of seek errors

This device characteristic specifies the number of hard (non revoverable) seek errors that have occured on this disk.

UNITTYPE - Unit type

This device characteristic specifies the type of a disk unit. For hard disks, the value of this characteristic is always HARD and cannot be changed. Valid values for floppy disks are:

Value	Meaning
HARD	Hard disk
HD3	3.5" high density
DD3	3.5" double density
HD5	5.25" high density
DD5	5.25" double density
DD8	8" double density

The value must be text with a length of at least 4 bytes.

VOLCDT - Volume creation date and time

This device characteristic specifies the time and data this volume was created.

VOLEDT - Volume effective date and time

This device characteristic specifies the the volumes effective use date and time (ISO 9660 CD-ROM).

VOLLABEL - Volume label

This device characteristic specifies the label for this disk volume.

VOLMDT - Volume modification date and time

This device characteristic specifies the last time this disk volume was modified.

VOLNAME - Volume name for disk

This device characteristic specifies an alternate name for a disk.

VOLXDT - Volume expiration date and time

This device characteristic specifies the expiration date and time for this volume.

WTMAX - Write transfer limit

This device characteristic specifies the harware write limit in blocks that this disk device can transfer in one operation.

HDKA Type Disk Devices (PC-AT hard disk)

This section describes the device characteristics which are specific to the HDKA type disk device. This device supports the standard PC-AT hard disk controller. This includes the original ST-501 controller, ESDI controllers, and IDE controllers. It does not include the PS-2 disk controllers or most SCSI controllers. SCSI controllers which fully emulate the PC-AT hard disk controller registers in hardware can be used, but none of the extra features of these controllers are supported by this device.

Hard disks use all of the common DISK class device characteristics listed previously plus the characteristics described here. Table 13.4 summarizes the hard disk specific device characteristics.

Table 13.4 - Device characteristics for HDKA type disks

Name	Fnc	Format	Size	Description
BUFSIZE	GS	DECV	4	Size of internal disk buffer
CONFIG	GS	HEXV	4	Disk configuration bits
MCYLNS	G	DECV	4	Number of mapped cylinders reported by drive
MHEADS	G	DECV	4	Number of mapped heads reported by drive
MSECTS	G	DECV	4	Number of mapped sectors reported by drive
SECPINT	G	DECV	4	Maximum sectors per interrupt
UXINTERR	GS	DECV	4	Number of unexpected interrupts

The hard disk specific device characteristics are described in detail below.

BUFSIZE - Size of internal disk buffer

This read-only device characteristic returns the length of the hard disk controller's internal data buffer in bytes. If the controller is not buffered or does not report the length of its buffer, a value of 0 is returned. The value must be numeric in decimal format with a length of at least 4 bytes.

CONFIG - Disk configuration bits

Device bits returned by IDE controller

MCYLNS - Number of mapped cylinders reported by drive

MHEADS - Number of mapped heads reported by drive

MSECTS - Number of mapped sectors reported by drive

SECPINT - Maximum sectors per interrupt

This read-only device characteristic returns the maximum number of sectors which the hard disk controller can transfer with a single interrupt. If the controller is not capable of multi-sector per interrupt operation or it does not report this information, a value of 0 is returned. The value must be numeric in decimal format with a length of at least 4 bytes.

UXINTERR - Number of unexpected interrupts

SDSK Type Disk Devices (SCSI controllers)

This section describes the device characteristics which are specific to the SDSK type disk device. This device supports all disks which are connected to the system using a SCSI interface.

SDSK hard disks use all of the common DISK class device characteristics listed previously plus the characteristics described here. Table 13.6 summarizes the SDSK specific device characteristics.

Table 13.6 - Device characteristics for SDSK type disks

Name	Fnc	Format	Size	Description
SCSIDEV	G	TEXT	16	SCSI device
SCSILUN	G	DECV	1	SCSI logical unit number
SCSITAR	G	DECV	1	SCSI target ID

The following section describes these device characteristics in detail.

SCSIDEV - SCSI device

This read-only characteristic returns the name of the SCSI controller device associated with this disk device. The value must be text with a length of at least 16 bytes.

SCSILUN - SCSI logical unit number

This read-only characteristic returns the SCSI logical unit number of the SCSI disk. The value must be numeric with a length of 1 byte or more.

SCSITAR - SCSI target ID

This read-only characteristic returns the SCSI target number of the SCSI disk. The value must be numeric with a length of 1 byte or more.

FDKA Type Disk Devices (floppy disk)

This section describes the device characteristics which are specific to the FDKA type disk device. This device supports the standard PC-AT floppy disk controller (NEC 765/Intel 8272) and the CompatiCard-I add-in floppy disk controller made by Micro Solutions, Inc. This controller uses the same chip as the standard PC-AT controller but provides slightly more flexibility in supporting non-standard floppy types, especially 8" floppies.

Floppy disks use all of the common DISK class device characteristics listed previously plus the characteristics described here. Table 13.7 summarizes the floppy disk specific device characteristics.

Table 13.7 - Device characteristics for floppy disks

Name	Fnc	Format	Size	Description
CONDESP	G	TEXT	4	Controller description
DATADEN	GS	TEXT	8	Data density
HLTIME	GS	DECV	2	Head load time in milliseconds
HUTIME	GS	DECV	2	Head unload time in milliseconds
MOTIME	GS	DECV	1	Motor off time in seconds
MSTIME	GS	DECV	4	Motor start time in milliseconds
SRTIME	GS	DECV	1	Step rate timing in milliseconds
TRKDEN	GS	DECV	1	Track density
XGAPLEN	GS	DECV	2	Gap length for transfers

The floppy disk specific device characteristics are described in detail below.

CONDESP - Controller description

This read-only device characteristic returns the description of the floppy disk controller. Possible valid values are:

Value	Meaning
PCAT	Standard AT floppy controller
CMPT	CompatiCard floppy controller

The value must be text with a length of at least 4 bytes.

DATADEN - Data density

This device characteristic specifies the data density for the floppy disk drive. Valid values are:

Value	Meaning
SINGLE	Single density floppy disk
DOUBLE	Double density floppy disk
HIGH	High density floppy disk

The value must be text with a length of at least 8 bytes.

HLTIME - Head load time

This device characteristic specifies the head load time (in milliseconds) for the floppy disk. The value must be numeric in decimal format with a length of at least 2 bytes.

HUTIME - Head unload time

This device characteristic specifies the head unload time (in milliseconds) for the floppy disk. The value must be numeric in decimal format with a length of at least 2 bytes.

MOTIME - Motor off time

This device characteristic specifies the motor off time (in seconds) for the floppy disk. The value must be numeric in decimal format with a length of at least 1 byte.

MSTIME - Motor start time

This device characteristic specifies the motor start time (in milliseconds) for the floppy disk. The value must be numeric in decimal format with a length of at least 4 bytes.

SRTIME - Step rate timing

This device characteristic specifies the step rate (in milliseconds) for the floppy disk. The value must be numeric in decimal format with a length of at least 1 byte.

TRKDEN - Track density

This device characteristic specifies the track density for the floppy disk drive. For 3.5" disks this value is always 135 and cannot be changed. For 5.25" disks this value may be 48 or 96. The value must be numeric in decimal format with a length of at least 1 byte.

XGAPLEN - Gap length for transfers

This device characteristic specifies the inter record gap value used with data transfers for the floppy disk. The value must be numeric in decimal format with a length of at least 2 bytes.

SPL Device Characteristics

This section describes the device characteristics that are specific to SPL class devices. These are “spooled” devices that are a special kind of disk device that allow the automatic generation of file names and automatic closing (and reopening if necessary) of files after some period of inactivity. This is normally used to capture output that appears to be directed to a printer or other physical output device to a disk file for later transfer to the actual physical device. The automatic closing fearture is provided mainly to allow spooled printer output from DOS programs, which usually do not open or close a printer dvice, but simply write to it with a special DOS or BIOS function.

The SPL device characteristics are summerized in table 13.8 below.

Table 13.8 - Device characteristics for SPL class devices

Name	Fnc	Format	Size	Description
CLSNAME	S	STR	48	File name for closed file
CLSMSG	S	TEXT	16	Destination for close message
CLSTIME	S	DECV	4	Inactive close time-out value (seconds)
SEQNUM	S	DECV	4	File name sequence number
SPLSPEC	S	STR	256	File specification for spooled file

The following section describes the device characteristics for SPL class devices in detail.

CLSNAME - File name for closed file

When a spooled file is closed, it is automatically renamed to the name specified by this device characteristic. The name or extension must contain the # character, which indicates that the current value of the spooled name sequence number is to be inserted. The # must be followed by a single digit which specifies the number of digits to insert. For example, if the CLSNAME value is S01_#4.SPL and the SEQNUM value is 52, the final name for the closed file will be S01_0052.SPL. The value must be a string with a buffer length of at least 48 bytes.

CLSMSG - Destination for close message

When a spooled file is closed, an IPM (Interprocess Message Device) message is sent to the destination specified by the value of this device characteristic. This will normally be the IPM name used by the

UNSPOOL symbiont which is unspooling the spooled device. The value must be text with a length of no more than 16 bytes.

CLSTIME - Inactive close time-out value (seconds)

This device characteristic specifies the maximum idle time (in seconds) for a spooled file. A spooled file is closed automatically if no output is done to the file for this interval. A value of 0 disables the automatic close feature. The value must be numeric in decimal format with a length of 4 bytes.

SEQNUM - File name sequence number

This device characteristic specifies the current spool name sequence number. This number is used when generating both the initial spooled file name and final closed spooled name. It is incremented by 1 whenever it is used. The value must be numeric in hex format with a length of 4 bytes.

SPLSPEC - File specification for spooled file

This device characteristic specifies the name which is used when opening a spooled file. Its value must be a complete file specification including a device name, directory path, name, and extension. The device specified must be a mass storage device. It can be either local or remote. The name or extension must contain the # character, which indicates that the current value of the spooled name sequence number is to be inserted. The # must be followed by a single digit which specifies the number of digits to insert. For example, if the SPLSPEC value is C:\SPOOL\S01_#4.TMP and the SEQNUM value is 193, the initial spooled file will be C:\SPOOL\S01_0193.TMP. The value must be a string with a buffer length of at least 256 bytes.

TAPE Device Characteristics

There are a number of device characteristics which are used by all TAPE class devices and there are a smaller number which are specific to individual types of TAPE controllers. The common characteristics for tapes are summarized in table 13.9

Table 13.9 - Device characteristics for TAPE class devices

Name	Fnc	Format	Size	Description
BYTEIN	GS	DECV	4	Total number of bytes input
BYTEOUT	GS	DECV	4	Total number of bytes output
DTHLIMIT	GS	DECV	4	Data transfer hardware limit
HDATAERR	GS	DECV	4	Number of hard data errors
HDEVERR	GS	DECV	4	Number of hard device errors
HOVRNERR	GS	DECV	4	Number of hard overrun errors
HUNGERR	GS	DECV	4	Number of hung device errors
MODEL	G	STR	44	Device model description
RECIN	GS	DECV	4	Total number of records input
RECMAX	GS	DECV	4	Maximum allowed record length
RECMIN	GS	DECV	4	Minimum allowed record length
RECOUT	GS	DECV	4	Total number of records output
REVISION	G	STR	12	Device revision number
SERIALNO	G	STR	24	Device serial number or other data
TDATAERR	GS	DECV	4	Total number of data errors
TDEVERR	GS	DECV	4	Total number of device errors
TOVRNERR	GS	DECV	4	Total number of overrun errors

The following section describes the device characteristics for TAPE class devices in detail.

BYTEIN - Number of bytes input

This read-write device characteristic returns the number of bytes written to the disk.

BYTEOUT - Number of bytes output

This read-write device characteristic returns the number of bytes read from the disk.

DTHLIMIT - Data transfer hardware limit

This device characteristic specifies the harware limit in records that this tape device can transfer in one operation.

HDATAERR - Number of hard data errors

This device characteristic specifies the number of hard (non revoverable) data read errors that have occured on this tape drive.

HDEVERR - Number of hard device errors

This device characteristic specifies the number of hard (non revoverable) device access errors that have occured on this tape drive.

HIDFERR - Number of hard ID field errors

This device characteristic specifies the number of hard (non revoverable) disk ID field read errors that have occured on this tape drive.

HOVRNERR - Number of hard DMA overrun errors

This device characteristic specifies the number of hard (non revoverable) DMA receive errors that have occured on this tape drive.

HUNGERR - Numner of device time-outs

This device characteristic specifies the number of hard (non revoverable) disk access timeout errors that have occured on this tape drive.

MODEL - Device model description

This is a text string returned by the tape drive giving the manufacturers description.

RECIN - Number of records input

This read-write device characteristic returns the number of records written to the tape.

RECMAX - Maximum allowed record length

This read-write device characteristics specifies the maximum allowed record length.

RECMIN - Minimum allowed record length

This read-write device characteristics specifies the minimum allowed record length.

RECOUT - Number of records output

This read-write device characteristic returns the number of records read from the tape.

REVISION - Device revision number

This is a text string returned by the disk giving the manufacturers revision number for the tape drive.

SERIALNO - Device serial number or other data

This read-only device characteristic returns the tape controller's serial number. If the controller does not report this information, a null string is returned. The value must be a string with a buffer length of at least 22 bytes.

TDATAERR - Total number of data errors

This device characteristic specifies the total number of hard (non revoverable) data read errors that have occured on this tape drive.

TDEVERR - Total number of device errors

This device characteristic specifies the total number of hard (non revoverable) device access errors that have occured on this tape drive.

TOVRNERR - Total number of overrun errors

This device characteristic specifies the total number of hard (non revoverable) overrun errors that have occured on this tape drive.

STAP Type Tape Devices (SCSI controllers)

This section describes the device characteristics which are specific to the STAP type tape device. This device supports all tape drives which are connected to the system using a SCSI interface.

STAPE tape drives use all of the common TAPE class device characteristics listed previously plus the characteristics described here. Table 13.10 summarizes the SDSK specific device characteristics.

Table 13.10 - Device characteristics for STAPE type tape drivess

Name	Fnc	Format	Size	Description
SCSIDEV	G	TEXT	16	SCSI device
SCSILUN	G	DECV	1	SCSI logical unit number
SCSITAR	G	DECV	1	SCSI target ID

The following section describes these device characteristics in detail.

SCSIDEV - SCSI device

This read-only characteristic returns the name of the SCSI controller device associated with this tape device. The value must be text with a length of at least 16 bytes.

SCSILUN - SCSI logical unit number

This read-only characteristic returns the SCSI logical unit number of the SCSI tape drive. The value must be numeric with a length of 1 byte or more.

SCSITAR - SCSI target ID

This read-only characteristic returns the SCSI target number of the SCSI tape drive. The value must be numeric with a length of 1 byte or more.

TRM Device Characteristics

This section describes the device characterictics which are specific to the various types of TRM class devices. These are serial port, console, psuedo-console, and telnet TRM devices.

TRM (Serial Port) Devices

This section describes the device characteristics which are specific to TRM class serial port devices. This includes serial ports using the SERA driver (standard AT serial ports) and ports using the SERB driver (DigiBoard multi-port serial interface). Table 13.11 summarizes the TRM class serial port device characteristics.

Table 13.11 - Device characteristics for TRM class serial ports

Name	Fnc	Format	Size	Description
ACCESS	GS	TEXT	4	System access class
CHARIN	GS	DECV	4	Number of characters input
CHAROUT	GS	DECV	4	Number of characters output
DBITS	GS	DECV	1	Current number of data bits
IDBITS	GS	DECV	1	Initial number of data bits
IINFLOW	GS	TEXT	8	Initial input flow control
IINRATE	GS	DECV	4	Initial input baud rate
IMODEM	GS	TEXT	4	Initial modem control
INFLOW	GS	TEXT	8	Current input flow control
INLBS	GS	DECV	2	Output ring buffer size
INRATE	GS	DECV	4	Current input baud rate (not used)
INRBHELD	GS	DECV	4	Input ring buffer held count
INRBLOST	GS	DECV	4	Input ring buffer lost count
INRBPL	GS	DECV	2	Input ring buffer panic level
INRBS	G	DECV	2	Input ring buffer size
INRBSL	GS	DECV	2	Input ring buffer stop level
INT	G	DECV	1	Interrupt level
INTRBS	G	DECV	2	Interrupt ring buffer size
INTRHELD	GS	DECV	4	Interrupt ring buffer held count
INTRLOST	GS	DECV	4	Interrupt ring buffer lost count
INTRPL	GS	DECV	2	Interrupt ring buffer panic level
INTRS	G	DECV	2	Interrupt ring buffer size

Table 13.11 - Device characteristics for TRM class serial ports

Name	Fnc	Format	Size	Description
INTRSL	GS	DECV	2	Interrupt ring buffer stop level
IOUTFLOW	GS	TEXT	8	Initial output flow control
IOUTRATE	GS	DECV	4	Initial output baud rate
IPARITY	GS	TEXT	8	Initial parity handling
IRATE	GS	DECV	4	Initial baud rate
ISBITS	GS	DECV	1	Initial number of stop bits
KBCHAR	GS	DECV	4	Keyboard characters
MODEM	GS	TEXT	4	Current modem control
MSGDST	GS	STR	16	Destination for initial message
OUTFLOW	GS	TEXT	8	Current output flow control
OUTRATE	GS	DECV	4	Current output baud rate
OUTRS	G	DECV	2	Output ring buffer size
PARITY	GS	TEXT	8	Current parity handling
PASSWORD	GS	STR	8	Password
PROGRAM	GS	STR	16	Initial program to run
RATE	GS	DECV	4	Current baud rate
RATEDET	GS	DECV	1	Baud rate detect type
SBITS	GS	DECV	1	Current number of stop bits
SESSION	GS	TEXT	8	Allow user session on port
STSREG	GS	HEXV	4	Status I/O register number (SERB only)

The following section describes the device characteristics for TRM class serial port devices in detail.

DBITS - Current number of data bits

This device characteristic specifies the current number of data bits in each character sent or received on the serial port. A value specified for this characteristic takes effect immediately and is in effect until the device is opened after being idle. At this time the value of this characteristic is initialized from the value of the IDBITS characteristic. The value must be between 5 and 8. Values less than 5 are taken as 5 and those greater than 8 are taken as 8. The value must be numeric in decimal format with a length of at least one byte.

IDBITS - Initial number of data bits

This device characteristic specifies the initial number of data bits in each character sent or received on the serial port. A value specified for this characteristic is not used directly but is used to initialize the value of the DBITS characteristic when the device is opened after being idle. The

value must be between 5 and 8. Values less than 5 are taken as 5 and those greater than 8 are taken as 8. The value must be numeric in decimal format with a length of at least one byte.

IINFLOW - Initial input flow control

This device characteristic specifies the initial input flow control handling. A value specified for this characteristic is not used directly but is used to initialize the value of the INFLOW characteristic when the device is opened after being idle. The valid values for this characteristic are:

Value	Meaning
DSRDTR	Hardware flow control using DSR and DTR
DSR	Same as DSRDTR
CTSRTS	Hardware flow control using CTS and RTS
CTS	Same as CTSRTS
REVCTS	Same as CTSRTS but with reversed sense
REV	Same REVCTS
XONXOFF	Flow control using XON and XOFF characters
XON	Same as XONXOFF
NONE	No flow control

The value must be text with a length of at least 8 bytes.

IINRATE - Initial input baud rate

This device characteristic specifies the initial input baud rate for the serial port. Since SERA and SERB devices do not support separate input and output baud rate specifications, setting the value of this characteristic has no effect. The value returned is the value of the IRATE characteristic.

IMODEM - Initial modem control

This device characteristic specifies the initial modem control handling state for the serial port. A value specified for this characteristic is not used directly but is used to initialize the value of the MODEM characteristic when the device is opened after being idle. Valid values for this characteristic are YES (port uses modem control features) and NO (port does not use modem control features). The value must be text with a length of at least 4 bytes.

INFLOW - Current input flow control

This device characteristic specifies the current input flow control handling state. The value specified for this parameter takes effect immediately and

is in effect until the next time the device is opened while idle. At this time, the value of this characteristic is initialized to be the same as the value of the IINFLOW characteristic. The valid values for this characteristic are:

Value	Meaning
DSRDTR	Hardware flow control using DSR and DTR
DSR	Same as DSRDTR
CTSRTS	Hardware flow control using CTS and RTS
CTS	Same as CTSRTS
REVCTS	Same as CTSRTS but with reversed sense
REV	Same REVCTS
XONXOFF	Flow control using XON and XOFF characters
XON	Same as XONXOFF
NONE	No flow control

The value must be text with a length of at least 8 bytes.

INLBS - Output ring buffer size

This read-only characteristic returns the size of the output ring buffer for the serial port. This buffer is allocated when the device is created and is used to buffer all data output to the port. The value must be numeric in decimal format with a length of at least 2 bytes.

INRATE - Current input baud rate

This device characteristic specifies the current input baud rate for the serial port. Since SERA and SERB devices do not support separate input and output baud rate specifications, setting the value of this characteristic has no effect. The value returned is the value of the RATE characteristic.

INRBS - Input ring buffer size

This read-only characteristic returns the size of the input ring buffer for the serial port. This buffer is allocated when the device is created and is used to type ahead data. The value must be numeric in decimal format with a length of at least 2 bytes.

INT - Interrupt level

This read-only characteristic returns the interrupt being used by the serial port. The value must be numeric in decimal format with a length of at least 1 byte.

INTRBS - Interrupt ring buffer size

This read-only characteristic returns the size of the interrupt ring buffer for the serial port. This buffer is allocated when the device is created and is used to buffer input data and output done indications at interrupt level. The value must be numeric in decimal format with a length of at least 2 bytes.

IOUTFLOW - Initial output flow control

This device characteristic specifies the initial output flow control handling. A value specified for this characteristic is not used directly but is used to initialize the value of the OUTFLOW characteristic when the device is opened after being idle. The valid values for this characteristic are:

Value	Meaning
DSRDTR	Hardware flow control using DSR and DTR
DSR	Same as DSRDTR
CTSRTS	Hardware flow control using CTS and RTS
CTS	Same as CTSRTS
REVCTS	Same as CTSRTS but with reversed sense
REV	Same REVCTS
XONXOFF	Flow control using XON and XOFF characters
XON	Same as XONXOFF
NONE	No flow control

The value must be text with a length of at least 8 bytes.

IOUTRATE - Initial output baud rate

This device characteristic specifies the initial output baud rate for the serial port. Since SERA and SERB devices do not support different input and output baud rates, this characteristic is equivalent to the IRATE characteristic. The value must be numeric in decimal format with a length of at least 4 bytes.

IPARITY - Initial parity handling

This device characteristic specifies the initial output parity handling state for the serial port. A value specified for this characteristic is not used directly but is used to initialize the value of the PARITY characteristic when the device is opened after being idle. Valid values for this characteristic are:

Value	Meaning
NONE	No parity bit is added
MARK	A marking bit is always added
SPACE	A spacing bit is always added
ODD	A bit is added to generate odd parity
EVEN	A bit is added to generate even parity

The value must be text with a length of at least 4 bytes.

IRATE - Initial baud rate

This device characteristic specifies the initial baud rate for the serial port. A value specified for this characteristic is not used directly but is used to initialize the value of the RATE characteristic when the device is opened after being idle. The baud rate value may be any positive number. If the value is less than the lowest baud rate supported by the interface, that rate is used; otherwise the system selects the highest available baud rate which is not greater than the value specified. The value must be numeric in decimal format with a length of at least 2 bytes.

ISBITS - Initial number of stop bits

This device characteristic specifies the initial number of stop bits for the serial port. A value specified for this characteristic is not used directly but is used to initialize the value of the SBITS characteristic when the device is opened after being idle. The value must be 1 or 2. Values less than 1 are taken as 1 and those greater than 2 are taken as 2. The value must be numeric in decimal format with a length of at least one byte.

MODEM - Current modem control

This device characteristic specifies the current modem control handling state for the serial port. A value specified for this characteristic takes effect immediately and stays in effect until the next time the device is opened while idle. At this time the value of this characteristic is initialized from the value of the IMODEM characteristic. Valid values for this characteristic are YES (port uses modem control features) and NO (port does not use modem control features). The value must be text with a length of at least 4 bytes.

OUTFLOW - Current output flow control

This device characteristic specifies the current output flow control handling state. The value specified for this parameter takes effect immediately and is in effect until the next time the device is opened while idle. At this

time, the value of this characteristic is initialized to be the same as the value of the IOUTFLOW characteristic. The valid values for this characteristic are:

Value	Meaning
DSRDTR	Hardware flow control using DSR and DTR
DSR	Same as DSRDTR
CTSRTS	Hardware flow control using CTS and RTS
CTS	Same as CTSRTS
REVCTS	Same as CTSRTS but with reversed sense
REV	Same REVCTS
XONXOFF	Flow control using XON and XOFF characters
XON	Same as XONXOFF
NONE	No flow control

The value must be text with a length of at least 8 bytes.

OUTRATE - Current output baud rate

This device characteristic specifies the current output baud rate for the serial port. Since SERA and SERB devices do not support different input and output baud rates, this characteristic is equivalent to the RATE characteristic. The value must be numeric in decimal format with a length of at least 4 bytes.

PARITY - Current parity handling

This device characteristic specifies the current output parity handling state for the serial port. A value specified for this characteristic takes effect immediately and stays in effect until the next time the device is opened while idle. At this time the value of this characteristic is initialized from the value of the IPARITY characteristic. Valid values for this characteristic are:

Value	Meaning
NONE	No parity bit is added
MARK	A marking bit is always added
SPACE	A spacing bit is always added
ODD	A bit is added to generate odd parity
EVEN	A bit is added to generate even parity

The value must be text with a length of at least 4 bytes.

RATE - Current baud rate

This device characteristic specifies the current baud rate for the serial port. A value specified for this characteristic takes effect immediately and stays in effect until the next time the device is opened while idle. At this time the value of this characteristic is initialized from the value of the IRATE characteristic. The baud rate value may be any number. If the value is less than the lowest baud rate supported by the interface, that rate is used; otherwise the system selects the highest available baud rate which is not greater than the value specified. The value must be numeric in decimal format with a length of at least 2 bytes.

RATEDET - Baud rate detect type

This parameter specifies the baud rate detect method to be used for automatic baud rate determination. This feature is not fully implemented in the current version of XOS.

SBITS - Current number of stop bits

This device characteristic specifies the current number of stop bits for the serial port. A value specified for this characteristic takes effect immediately and stays in effect until the next time the device is opened while idle. The value of this characteristic is initialized from the value of the ISBITS characteristic. The value must be 1 or 2. Values less than 1 are taken as 1 and those greater than 2 are taken as 2. The value must be numeric in decimal format with a length of at least 1 byte.

SESSION - Allow user session on port

This characteristic specifies if the port can be used to control a user session. Normally, this characteristic will have the value YES or NO. If the value is YES, any input from the serial port while the port is idle causes the system to create a new process and run a command shell with the port as the controlling terminal. If the value is NO, input from the serial port while it is idle is ignored. The value may also be set to any sequence of characters beginning with an underscore character. In this case, the system will send an IPM message to the IPM name formed by removing the underscore from the characteristic value whenever there is input from the serial port and it is idle. The user must have a program running which has opened an IPM device with this name and is prepared to receive the messages. This is intended to provide a method of implementing non-standard terminal based systems. The value must be text with a length of at least 8 bytes.

STSREG - Status I/O register number (SERB only)

This read-only characteristic returns the I/O register number of the DigiBoard status register. It is only valid for SERB type devices. Each DigiBoard has a single status register, regardless of the number of ports which it supports. The number of this status register is used by the system to identify the board (as opposed to the individual ports). The value must be numeric in hex format with a length of at least 2 bytes.

TRM (Console) Devices

This section describes the device characteristics which are specific to TRM class console devices. This includes consoles using the MGAA (Hercules compatible monochrome graphics adapter), EGAA (Enhanced Graphics Adapter and compatibles), and VGAA (Virtual Graphics Array and compatibles) type devices. Table 13.12 summarizes the TRM class console device characteristics.

Table 13.12 - Device characteristics for TRM class consoles

Name	Fnc	Format	Size	Description
BELLFREQ	GS	DECV	2	Bell tone frequency
BELLLEN	GS	DECV	2	Bell tone length
CHARIN	GS	DECV	4	Number of characters input
CHAROUT	GS	DECV	4	Number of characters output
CURFIX	GS	TEXT	4	Special cursor fix-up enable
INLBS	GS	DECV	4	Input line buffer size
INRBHELD	GS	DECV	4	Number of times input held for flow control
INRBLOST	GS	DECV	4	Number of input characters discarded
INRBPL	GS	DECV	4	Input ring buffer panic level
INRBS	GS	DECV	4	Input ring buffer size
INRBSL	GS	DECV	4	Input ring buffer stop level
IOUTFLOW	GS	TEXT	4	Initial output flow control
KBCHAR	GS	DECV	4	Number of keyboard scan codes input
KBTCHAR	GS	DECV	4	Total number of keyboard scan codes input
OUTFLOW	GS	TEXT	4	Current output flow control
PASSWORD	GS	STR	12	System level password for console
PROGRAM	GS	STR	4	Initial program to run
SESSION	GS	TEXT	4	Allow session on console
SCSVTIME	GS	DECV	4	Screen saver time (seconds)
SCSVTYPE	GS	TEXT	4	Screen saver type

The following section describes the device characteristics for TRM class console devices in detail.

BELLFREQ - Bell tone frequency

This device characteristic specifies the frequency (in Hertz) for the bell tone associated with the console. The value must be numeric with a length of at least 2 bytes.

BELLLEN - Bell tone length

This device characteristic specifies the length (in milliseconds) of the bell tone generated when a BELL (Cntl-G) character is output. The value must be numeric with a length of at least 2 bytes.

CHARIN - Number of characters input

This device characteristic records the number of characters input by the device. The value must be numeric in decimal format with a length of 4 bytes.

CHAROUT - Number of characters output

This device characteristic records the number of characters output by the device. The value must be numeric in decimal format with a length of 4 bytes.

CURFIX - Special cursor fix-up enable

This device characteristic specifies if the special cursor fix-up mode is enabled. Some display adapters do not correctly display a full block cursor. When the special fix up mode is enabled, full block cursors are reduced by one scan line. Valid values are YES (indicating that the special fix-up mode is enabled) and NO (indicating that the special fix-up mode is disabled). The value must be text with a length of at least 4 bytes.

INLBS

This device characteristic returns the size of the input line buffer for the device. This value is specified when the console device is created and cannot be subsequently changed. The value must be numeric in decimal format with a length of at least 4 bytes.

INRBHELD

This device characteristic records the number of times that input flow control has been held of input because the input ring buffer was nearly full (See description of the INRBSL device characteristic). The value must be numeric in decimal format with a length of at least 4 bytes.

INRBLOST

This device characteristic records the number of input characters which have been discarded because the input ring buffer was full. The value must be numeric in decimal format with a length of at least 4 bytes.

INRBPL

This device characteristic specifies the level at which "panic mode" is used to hold off input. This is only relevant when using XON/XOFF flow control. When the input buffer fills up to the level specified by the INRBSL characteristic a single XOFF character is output to attempt to stop input. No additional response is made until the number of character positions available is less than the level specified by the value of this parameter. At this point, an XOFF character is output for every input character received. A value of 0 for this characteristic disables the panic mode response. The value must be numeric in decimal format with a length of at least 4 bytes.

INRBS

This device characteristic returns the size of the input ring buffer for the device. This value is specified when the console device is created and cannot be changed after that. The input ring buffer is used to buffer type ahead. The value must be numeric in decimal format with a length of at least 4 bytes.

INRBSL - Input ring buffer stop level

This device characteristic specifies the level at which input is held off. The level is specified as the number of character positions available at which point input is to be held off. The value must be numeric in decimal format with a length of at least 4 bytes.

IOUTFLOW - Initial output flow control

This device characteristic specifies the current output flow control handling state. The value specified for this parameter takes effect immediately and is in effect until the next time the device is opened while idle. At this time, the value of this characteristic is initialized to be the same as the value of the IOUTFLOW characteristic. The valid values for this characteristic are:

Value	Meaning
XONXOFF	Flow control using XON and XOFF characters
XON	Same as XONXOFF
NONE	No flow control

The value must be text with a length of at least 8 bytes.

KBCHAR - Number of keyboard scan codes input

This device characteristic records the number of scan codes generated by the console keyboard for this device. This includes both make and break codes. The value must be numeric in decimal format with a length of at least 4 bytes.

KBTCHAR - Total number of keyboard scan codes input

This device characteristic records the total number of scan codes generated by the console keyboard for all devices which can be associated with it. This includes both make and break codes. The value must be numeric in decimal format with a length of at least 4 bytes.

OUTFLOW - Current output flow control

This device characteristic specifies the current output flow control handling state. The value specified for this parameter takes effect immediately and is in effect until the next time the device is opened while idle. At this time, the value of this characteristic is initialized to be the same as the value of the IOUTFLOW characteristic. The valid values for this characteristic are:

Value	Meaning
XONXOFF	Flow control using XON and XOFF characters
XON	Same as XONXOFF
NONE	No flow control

The value must be text with a length of at least 8 bytes.

PASSWORD - System level password for console

This device characteristic specifies the system level password associated with the terminal device. If a password is specified, it must be entered whenever a session is started on the terminal. The value must be a string with a buffer length of at least 12 bytes.

PROGRAM - Initial program to run

This device characteristic specifies the name of the program to run initially when a session is started on the terminal. If no name is specified, either SHELL.IMG or LOGIN.IMG is run, depending on whether user logins are enabled for the system. Only a name can be specified. The program must be in the directory specified by the XOSSYS: logical name. The value must be a string with a buffer length of at least 16 bytes.

SESSION - Allow session on console

This characteristic specifies if the port can be used to control a user session. Normally, this characteristic will have the value YES or NO. If the value is YES, any input from the serial port while the port is idle causes the system to create a new process and run a command shell with the port as the controlling terminal. If the value is NO, input from the serial port while it is idle is ignored. The value may also be set to any sequence of characters beginning with an underscore character. In this case, the system will send an IPM message to the IPM name formed by removing the underscore from the characteristic value whenever there is input from the serial port and it is idle. The user must have a program running which has opened an IPM device with this name and is prepared to receive the messages. This is intended to provide a method of implementing non-standard terminal based systems. The value must be text with a length of at least 8 bytes.

SCSVTIME - Screen saver time

This device characteristic specifies the time (in seconds) for the screen saver function. The screen is blanked if there is no keyboard activity for this time interval. It is unblanked as soon as any keyboard activity occurs. A value of 0 disables the screen saver function. The value must be numeric with a length of at least 2 bytes.

SCSVTYPE - Screen saver type

This device characteristic specifies the behavior of the screen saver function. A value of K or KEY specifies that only keyboard activity will be considered when blanking or unblanking the screen. A value of F or FULL specifies that program output to the screen will also be considered. The value must be text with a length of at least 4 bytes.

TRM (Pseudo-Console) Devices

This section describes the device characteristics which are specific to TRM class devices which are the client side of a PCN (pseudo-console) device. These devices are true TRM class devices although they do not correspond to a physical terminal device. They are used by various servers (specifically by the Telnet server) to emulate a real terminal. Table 13.13 summarizes the TRM class console device characteristics.

Table 13.13 - Device characteristics for TRM class consoles

Name	Fnc	Format	Size	Description
INLBS	G	DECV	4	Input line buffer size
INRBS	G	DECV	4	Input ring buffer size
PASSWORD	GS	STR	12	System level password for terminal
PROGRAM	GS	STR	16	Initial program to run
SESSION	GS	TEXT	4	Allow session on console

The following section describes the device characteristics for TRM class pseudo-console devices in detail.

INLBS - Input line buffer size

This device characteristic returns the size of the input line buffer for the device. This value is specified when the console device is created and cannot be changed after that. The value must be numeric in decimal format with a length of at least 4 bytes.

INRBS - Input ring buffer size

This device characteristic returns the size of the input ring buffer for the device. This value is specified when the console device is created and cannot be subsequently changed. The input ring buffer is used to buffer type ahead. The value must be numeric in decimal format with a length of at least 4 bytes.

PASSWORD - System level password for terminal

This device characteristic specifies the system level password associated with the terminal device. If a password is specified, it must be entered whenever a session is started on the terminal. The value must be a string with a buffer length of at least 12 bytes.

PROGRAM - Initial program to run

This device characteristic specifies the name of the program to run initially when a session is started on the terminal. If no name is specified, either SHELL.IMG or LOGIN.IMG is run, depending on whether user logins are enabled for the system. Only a name can be specified. The program must be in the directory specified by the XOSSYS: logical name. The value must be a string with a buffer length of at least 16 bytes.

SESSION - Allow session on console

This device characteristic specifies if the port can be used to control a user session. Normally, this characteristic will have the value YES or NO. If the value is YES, any input from the serial port while the port is idle causes the system to create a new process and run a command shell (or the login program if user login is enabled for the system) with the port as the controlling terminal. If the value is NO, input from the serial port while it is idle is ignored. The value may also be set to any sequence of characters beginning with an underscore character. In this case, the system will send an IPM message to the IPM name formed by removing the underscore from the characteristic value whenever there is input from the serial port and it is idle. The user must have a program running which has opened an IPM device with this name and is prepared to receive the messages. This is intended to provide a method of implementing non-standard terminal based systems. The value must be text with a length of at least 4 bytes.

TRM (Telnet) Devices

This section describes the device characteristics which are specific to TRM class devices which are the client side of a PCN (pseudo-console) device. These devices are true TRM class devices although they do not correspond to a physical terminal device. They are used by various servers (specifically by the Telnet server) to emulate a real terminal. Table 13.14 summarizes the TRM class console device characteristics.

Table 13.14 - Device characteristics for TRM class consoles

Name	Fnc	Format	Size	Description
INLBS	G	DECV	4	Input line buffer size
INRBS	G	DECV	4	Input ring buffer size
PASSWORD	GS	STR	12	System level password for terminal
PROGRAM	GS	STR	16	Initial program to run
SESSION	GS	TEXT	4	Allow session on console

The following section describes the device characteristics for TRM class pseudo-console devices in detail.

INLBS - Input line buffer size

This device characteristic returns the size of the input line buffer for the device. This value is specified when the console device is created and cannot be subsequently changed. The value must be numeric in decimal format with a length of at least 4 bytes.

INRBS - Input ring buffer size

This device characteristic returns the size of the input ring buffer for the device. This value is specified when the console device is created and cannot be changed after that. The input ring buffer is used to buffer type-ahead. The value must be numeric with a length of at least 4 bytes.

PASSWORD - System level password for terminal

This device characteristic specifies the system level password associated with the terminal device. If a password is specified, it must be entered whenever a session is started on the terminal. The value must be a sting with a buffer length of at least 12 bytes.

PROGRAM - Initial program to run

This device characteristic specifies the name of the program to run initially when a session is started on the terminal. If no name is specified, either SHELL.IMG or LOGIN.IMG is run, depending on whether user logins are enabled for the system. Only a name can be specified. The program must be in the directory specified by the XOSSYS: logical name. The value must be a string with a buffer length of at least 16 bytes.

SESSION - Allow session on terminal

This device characteristic specifies if the port can be used to control a user session. Normally, this characteristic will have the value YES or NO. If the value is YES, any input from the serial port while the port is idle causes the system to create a new process and run a command shell (or the login program if user login is enabled for the system) with the port as the controlling terminal. If the value is NO, input from the serial port while it is idle is ignored. The value may also be set to any sequence of characters beginning with an underscore character. In this case, the system will send an IPM message to the IPM name formed by removing the underscore from the characteristic value whenever there is input from the serial port and it is idle. The user must have a program running which has opened an IPM device with this name and is prepared to receive the messages. This is intended to provide a method of implementing non-standard terminal based systems. The value must be text with a length of at least 4 bytes.

PCN Device Characteristics

This section describes the device characteristics for PCN class devices. These are summarized in Table 13.15.

Table 13.15 - Device Characteristics for PCN Class Devices

Name	Fnc	Format	Size	Description
INLBS	GS	DECV	4	Input line buffer size
INRBS	GS	DECV	4	Input ring buffer size
PASSWORD	GS	STR	12	System level password
PROGRAM	GS	STR	16	Initial program to run
SESSION	GS	TEXT	4	Allow session on terminal

Following is a detailed description of the device characteristic defined for the PCN device class.

INLBS - Input line buffer size

This device characteristic specifies the size of the line buffer which is allocated when a PCN device is opened. This will be the value returned by the INLBS device characteristic for the terminal class device associated with the PCN device. The value must be numeric in decimal format with a length of at least 4 bytes.

INRBS - Input ring buffer size

This device characteristic specifies the size of the input ring buffer which is allocated when a PCN device is opened. This buffer is used to hold type ahead data. This will be the value returned by the INRBS device characteristic for the terminal class device associated with the PCN device. The value must be numeric in decimal format with a length of at least 4 bytes.

PASSWORD - System level password

This device characteristic specifies the initial value for the PASSWORD device characteristic for TRM class devices associated with the PCN device. The value must be a string with a buffer length of at least 12 bytes.

PROGRAM - Initial program to run

This device characteristic specifies the initial value for the PROGRAM device characteristic for TRM class devices associated with the PCN device. The value must be string with a buffer length of at least 16 bytes.

SESSION - Allow session on terminal

This device characteristic specifies the initial value for the SESSION device characteristic for TRM class devices associated with the PCN device. The value must be text with a length of at least 4 bytes.

IPM Device Characteristics

There are no device characteristics associated with the IPM class devices.

NULL Device Characteristics

The NULL class device is a dummy device which discards all output and always returns an end-of-file condition on input. There are no device characteristics associated with the NULL class devices.

PPR Device Characteristics

This section describes the device characteristics for PPR class devices. These are summarized in Table 13.16.

Table 13.16 - Device Characteristics for PPR Class Devices

Name	Fnc	Format	Size	Description
INT	G	DECV	1	Interrupt number
TIMEOUT	GS	HEXV	4	Default time-out value

Following is a detailed description of the device characteristic defined for the PPR device class.

INT - Interrupt number

This device characteristic returns the interrupt number for the device. This value cannot be changed after the device is created. The value must be numeric in decimal format with a length of at least 1 byte.

TIMEOUT - Default time-out value

This device characteristic specifies the default time out value in seconds. If no time out value is specified when doing output to the PPR device, this value is used to determine when to return “a no response” error. A value of 0 indicates no time out. The value must be numeric in decimal format with a length of at least 4 bytes.

NET Device Characteristics

This section describes the device characteristics which are specific to NET class devices. The NET devices are the low level network interfaces. These devices can be used to send and receive raw network data, but are most useful for obtaining statistics about the operation of the network using the device characteristics described here. Even though multiple NET DCBs are created as needed to allow multiple processes to access the network, the various error and usage counters accessed with these device characteristics reflect the total usage of the network interface.

This section first describes the device characteristics which are used by all NET class devices. This is followed by descriptions of the device characteristics which are specific to each type of NET device.

Table 13.17 summarizes the common device characteristics for the NET class devices.

Table 13.17 - Device characteristics for NET class devices

Name	Fnc	Format	Size	Description
BADPNT	GS	DECV	4	Number of packets discarded because of bad ring pointer
BCPKTIN	GS	DECV	4	Number of broadcast packets input
BYTEIN	GS	DECV	4	Number of bytes input
BYTEOUT	GS	DECV	4	Number of bytes output
ICRC	GS	DECV	4	Number of input CRC errors
IFRAME	GS	DECV	4	Number of input framing errors
ILOST	GS	DECV	4	Number of lost input packets
INT	G	DECV	1	Interrupt level
IOVRRN	GS	DECV	4	Number of input overrun errors
NETADDR	G	HEXB	6	Physical network address
NOBFR	GS	DECV	4	Number of packets discarded because no buffer available
NODST	GS	DECV	4	Number of packets discarded because no destination for E-N protocol
OCOL	GS	DECV	4	Number of output collisions
OCSEN	GS	DECV	4	Number of output carrier lost errors
OHTBT	GS	DECV	4	Number of output heartbeat errors
OHUNG	GS	DECV	4	Number of hung output errors
OOWC	GS	DECV	4	Number of output out of window collisions
OUNDRN	GS	DECV	4	Number of output underrun errors
OXCOL	GS	DECV	4	Number of excessive output collisions

Table 13.17 - Device characteristics for NET class devices

Name	Fnc	Format	Size	Description
PKTIN	GS	DECV	4	Number of packets input
PKTOUT	GS	DECV	4	Number of packets output

Following is a detailed description of each common device characteristic for the NET class devices.

BADPNT - Number of packets discarded because of bad ring pointer

This device characteristic records the number of input packets discarded because of an invalid network interface buffer ring pointer. This is an internal network interface error which should not occur if the hardware and driver software are working correctly. The value must be numeric in decimal format with a length of at least 4 bytes.

BCPKTIN - Number of broadcast packets input

This device characteristic records the total number of input broadcast packets received. The value must be numeric in decimal format with a length of at least 4 bytes.

BYTEIN - Number of bytes input

This device characteristic records the total number of bytes input. The value must be numeric in decimal format with a length of at least 4 bytes.

BYTEOUT - Number of bytes output

This device characteristic records the total number of bytes output. The value must be numeric in decimal format with a length of at least 4 bytes.

ICRC - Number of input CRC errors

This device characteristic records the total number of input CRC errors reported by the network interface. Packets with bad CRCs are discarded. The value must be numeric in decimal format with a length of at least 4 bytes.

IFRAME - Number of input framing errors

This device characteristic records the total number of input framing errors reported by the network interface. Packets with framing errors are discarded. The value must be numeric in decimal format with a length of at least 4 bytes.

ILOST - Number of lost input packets

This device characteristic records the total number of lost input packets reported by the network interface. A lost input packet is one that was discarded because there was no space for it in the network interface's internal buffer. The value must be numeric in decimal format with a length of at least 4 bytes.

INT - Interrupt level

This read-only device characteristic returns the interrupt level used by the network interface. The value must be numeric in decimal format with a length of at least 1 byte.

IOVRRN - Number of input overrun errors

This device characteristic records the total number of input overrun errors reported by the network interface. The value must be numeric in decimal format with a length of at least 4 bytes.

NETADDR - Physical network address

This read-only device characteristic returns the physical network address of the network interface. XOS always uses the network address belonging to the interface. It cannot be changed. The value must be a byte list in hex format with a length of at least 6 bytes.

NOBFR - Number of packets discarded because no buffer available

This device characteristic records the total number of input packets discarded because there was no system buffer available for the packet. The value must be numeric in decimal format with a length of at least 4 bytes.

NODST - Number of packets discarded because no destination for E-N protocol

This device characteristic records the total number of input packets discarded because there was no destination available for the ethernet protocol type of the packet; i.e., no process or device driver was registered to handle the protocol type and there was no default protocol handler registered. The value must be numeric in decimal format with a length of at least 4 bytes.

OCOL - Number of output collisions

This device characteristic records the total number of output collisions reported by the network interface. This includes output collisions which are

retried by the interface. The value must be numeric in decimal format with a length of at least 4 bytes.

OCSEN - Number of output carrier lost errors

This device characteristic records the total number of output carrier lost errors reported by the network interface. The value must be numeric in decimal format with a length of at least 4 bytes.

OHTBT - Number of output heartbeat errors

This device characteristic records the total number of output heartbeat errors reported by the network interface. The value must be numeric in decimal format with a length of at least 4 bytes.

OHUNG - Number of hung output errors

This device characteristic records the total number of times that an expected output done interrupt did not occur. The value must be numeric in decimal format with a length of at least 4 bytes.

OOWC - Number of output out of window collisions

This device characteristic records the total number of out of window collisions reported by the network interface. The value must be numeric in decimal format with a length of at least 4 bytes.

OUNDRN - Number of output underrun errors

This device characteristic records the total number of output underrun errors reported by the network interface. The value must be numeric in decimal format with a length of at least 4 bytes.

OXCOL - Number of excessive output collisions

This device characteristic records the total number of excessive output collisions reported by the network interface. An excessive output collision occurs when the interface exceeds its maximum retry amount without being able to output a packet without a collision. The value must be numeric in decimal format with a length of at least 4 bytes.

PKTIN - Number of packets input

This device characteristic records the total number of packets input. The value must be numeric in decimal format with a length of at least 4 bytes.

PKTOUT - Number of packets output

This device characteristic records the total number of packets output. The value must be numeric in decimal format with a length of at least 4 bytes.

EWDA Network Interface Devices

This sections describes the device characteristics specific to the EWDA type network interface. The EWDA interface uses these device characteristics in addition to the common network interface device characteristics described previously.

Table 13.18 summarizes the device characteristics which are specific to EWDA type devices.

Table 13.18 - Device characteristics for EWDA network interface devices

Name	Fnc	Format	Size	Description
MEM	G	HEXV	4	Shared memory address

The device characteristic specific to the ENWA type network interface is described in detail below.

MEM - Shared memory address

This read-only device characteristic returns the physical address of the network interface's shared memory area. This address is specified when the device is created and must match the value for which the board is configured. The value must be numeric in hex format with a length of at least 4 bytes.

ENEA Network Interface Devices

This sections describes the device characteristics specific to the ENEA type network interface. The ENEA interface uses these device characteristics in addition to the common network interface device characteristics described previously.

Table 13.19 summarizes the device characteristics which are specific to the ENEA type device.

Table 13.19 - Device characteristics for ENEA network interface devices

Name	Fnc	Format	Size	Description
BOARD	G	TEXT	12	Board type

Note that the ENEA type network interface does not use shared memory, so there is no MEM device characteristic for this interface. The device characteristic specific to the ENEA type network interface is described in detail below.

BOARD - Board type

This read-only device characteristic returns the type of interface board in use. The ENEA type interface driver actually supports three types of boards. The type is dynamically determined by the interface driver. This characteristic reports the result of this determination. Possible values are:

Value	Meaning
NE1000	Novell NE1000 or compatible
NE2000-8	Novell NE2000 or compatible in 8-bit slot
NE2000-16	Novell lNE2000 or compatible in 16-bit slot

The value must be text with a length of at least 12 bytes.

E3CA Network Interface Devices

This sections describes the device characteristics specific to the E3CA type network interface. The E3CA interface uses these device characteristics in addition to the common network interface device characteristics described previously.

Table 13.20 summarizes the device characteristics which are specific to the E3CA type device.

Table 13.20 - Device characteristics for E3CA network interface devices

Name	Fnc	Format	Size	Description
MEM	G	HEXV	4	Shared memory address
THICK	G	TEXT	4	Interface uses thick-wire cable

The device characteristics specific to the E3CA type network interface are described in detail below.

MEM - Shared memory address

This read-only device characteristic returns the physical address of the network interface's shared memory area. This address is read from the board's configuration when the device is created. The value must be numeric in hex format with a length of at least 4 bytes.

THICK - Interface uses thick-wire cable

This device characteristic controls the operation of the network interface's on board thin wire transceiver. Unlike most network interfaces, this interface provides for software control of this feature. A value of YES disables the on board transceiver and enables thick wire operation. A value of NO disables thick wire operation and enables the on board transceiver. The value must be text with a length of at least 4 bytes.

SNAP Device Characteristics

This section describes the device characteristics for SNAP class devices. SNAP class devices provide access to the link control level for the Internet protocol stack using the SNAP or Bluebook link level protocols. These device characteristics are summarized in Table 13.21.

Table 13.21 - Device Characteristics for SNAP Class Devices

Name	Fnc	Format	Size	Description
BADPDU	GS	DECV	4	Number of physical data units with illegal format
BYTEIN	GS	DECV	4	Number of bytes input from the network
BYTEOUT	GS	DECV	4	Number of bytes output to the network
NETDEV	G	TEXT	8	Name of associated network interface device
NODST	G	DECV	4	Number of packets discarded because no destination available
PKTIN	GS	DECV	4	Number of packets input from the network
PKTOUT	GS	DECV	4	Number of packets output to the network
SAP	GS	HEXB	2	Service access point address

Following is a detailed description of the device characteristic defined for the SNAP device class.

BADPDU - Number of physical data units with illegal format

This device characteristic records the number of input physical data units (link level packets) which have been discarded because of an illegal format. The value must be numeric in decimal format with a length of at least 4 bytes.

BYTEIN - Number of bytes input from the network

This device characteristic records the total number of bytes input from the network. The value must be numeric in decimal format with a length of at least 4 bytes.

BYTEOUT - Number of bytes output from the network

This device characteristic records the total number of bytes output to the network. The value must be numeric in decimal format with a length of at least 4 bytes.

NETDEV - Name of associated network interface device

This device characteristic reports the name of the underlying network interface device. This will always be the name of a NET class device. The value must be text with a length of at least 8 bytes.

NODST - Number of packets discarded because no destination available

This device characteristic reports the number of input packets that have been discarded because no destination was available for the packet. In the case of IP packets, this means that there was no routine established to handle the IP protocol type that was specified in the packet. The value must be numeric in decimal format with a length of at least 4 bytes.

PKTIN - Number of packets input from the network

This device characteristic reports the total number of input packets received from the network. The value must be numeric in decimal format with a length of at least 4 bytes.

PKTOUT - Number of packets output to the network

This device characteristic reports the total number of output packets output to the network. The value must be numeric in decimal format with a length of at least 4 bytes.

SAP - Service access point address

This device characteristic specifies the IEEE-802.2 service access point address associated with this SNAP device. This value is usually specified as two hex bytes. A value of 0FF-FF indicated that the SNAP device is handling the Ethernet-II/Bluebook link level protocol instead of the 802.2 protocol. Any other value indicates an 802.2 protocol. A value of 0AA-AA is the standard SAP value for the 802.2 SNAP protocol. The value must be numeric in hex byte format with a length of at least 2 bytes.

ARP Device Characteristics

This section describes the device characteristics for ARP class devices. This device provides access to the ARP protocol level in the Internet protocol stack. These device characteristics are summarized in Table 13.22.

Table 13.22 - Device Characteristics for ARP Class Devices

Name	Fnc	Format	Size	Description
BADHDR	GS	DECV	4	Number of packets discarded because of a bad header
BYTEIN	GS	DECV	4	Number of ARP bytes input
BYTEOUT	GS	DECV	4	Number of ARP bytes output
ETYPE	GS	HEXB	2	Ethertype value
PKTIN	GS	DECV	4	Number of ARP packets input
PKTOUT	GS	DECV	4	Number of ARP packets output
SNAPDEV	GS	TEXT	8	Name of underlying SNAP device

Following is a detailed description of the device characteristic defined for the ARP device class.

BADHDR - Number of packets discarded because of a bad header

This device characteristic records the number of ARP input packets that have been discarded because of a bad header format. The value must be numeric in decimal format with a length of at least 4 bytes.

BYTEIN - Number of ARP bytes input

This device characteristic records the total number of ARP bytes input. The value must be numeric in decimal format with a length of at least 4 bytes.

BYTEOUT - Number of ARP bytes output

This device characteristic records the total number of ARP bytes output. The value must be numeric in decimal format with a length of at least 4 bytes.

ETYPE - Ethertype value

This device characteristic specifies the Ethertype value for the protocol suite with which the ARP device is associated. The value is usually specified as two hex bytes. When used with the IP protocol, the value is nor-

mally 08-06. The value must be numeric in hex format with a length of at least 2 bytes.

PKTIN - Number of ARP packets input

This device characteristic records the total number of ARP packets input. The value must be numeric in decimal format with a length of at least 4 bytes.

PKTOUT - Number of ARP packets output

This device characteristic records the total number of ARP packets output. The value must be numeric in decimal format with a length of at least 4 bytes.

SNAPDEV - Name of underlying SNAP device

This device characteristic reports the name of the underlying SNAP (link level) device in the network protocol stack. The value must be text with a length of at least 8 bytes.

IPS Device Characteristics

This section describes the device characteristics which are specific to IPS class devices. The IPS devices are the lowest level network protocol devices which provide access to the IP level of the Internet Protocol Suite. These devices can be used to send and receive raw network data, but are most useful for obtaining statistics about the operation of the network using the device characteristics described here. Even though multiple IPS DCBs are created as needed to allow multiple processes to access the network, the various error and usage counters accessed with these device characteristics reflect the total usage at the IP protocol level.

Table 13.23 summarizes the device characteristics for the IPS class devices.

Table 13.23 - Device characteristics for the IPS device class

Name	Fnc	Format	Size	Description
ADJADDR	G	DECB	4	Adjacent node IP address
ARPDEV	G	TEXT	16	ARP device
BADHDR	GS	DECV	4	Discarded - bad header
BYTEIN	GS	DEVV	4	Number of bytes input
BYTEOUT	GS	DECV	4	Number of bytes output
CHKSUM	GS	DECV	4	Discarded - bad IP header checksum
CHKSUMH	GS	TEXT	8	IP checksum handling
CLASS	GV	TEST	16	Device class
DLLTHL	GS	DECV	4	Discarded - data header length
DOMAIN	GS	STR	64	Domain Name for system
DRT1ADDR	GS	DECB	4	Default router IP address
ETYPE	GS	DECB	2	Ethertype value
HOSTDOWN	GS	TEXT	4	Primary host system is down
IPADDR	GS	DECB	4	IP address
NAMESRVR	GS	TEXT	4	Domain Name Server on this system
NETMASK	G	DECB	4	IP network address mask
NODST	GS	DECV	4	Discarded - no destination for IP
NOMERGE	GS	DECV	1	Do not merge or split packets
NUMSNAP	G	DECV	1	Number of SNAP devices
PKTIN	GS	DECV	4	Number of packets input
PKTOUT	GS	DECV	4	Number of packets output
PSLTDL	GS	DECV	4	Discarded - packet data length
PSLTMN	GS	DECV	4	Discarded - packet minimum
RMTADDR	GS	DECB	4	Default remote IP address
RTPURGE	GS	DECV	4	IP routing table purge request

Table 13.23 - Device characteristics for the IPS device class

Name	Fnc	Format	Size	Description
RTREMOVE	GS	HEXV	1	IP routing table remove request
RTSIZE	G	DECV	4	IP routing table size
RTUSE	G	DECV	4	IP routing table usage
SNAPDEV	G	TEXT	16	SNAP device
SUBMASK	GS	HEXV	4	IP subnet address mask

Following is a detailed description of each device characteristic for IPS class devices.

BADHDR - Number of packets discarded because of bad header

This device characteristic records the total number of IP packets discarded because of an invalid header format. The value must be numeric in decimal format with a length of at least 4 bytes.

BYTEIN - Number of bytes input

This device characteristic records the total number of bytes input for the IP protocol. The value must be numeric in decimal format with a length of at least 4 bytes.

BYTEOUT - Number of bytes output

This device characteristic records the total number of bytes output for the IP protocol. The value must be numeric in decimal format with a length of at least 4 bytes.

CHKSUM - Number of packets discarded because of bad IP header checksum

This device characteristic records the total number of packets discarded because of bad IP header checksum value. The value must be numeric in decimal format with a length of at least 4 bytes.

CHKSUMH - IP checksum handling

This device characteristic specifies how IP checksums are handled by the system. Valid values are:

Value	Meaning
NONE	No IP checksums are generated or checked
INPUT	IP checksums are checked on input but not generated on output
OUTPUT	IP checksums are generated on output but not checked on input
FULL	IP checksums are checked on input and are generated on output

Note that this device characteristic also controls how TCP and UDP checksums are handled. The value must be text with a length of at least 8 bytes.

DLLTHL - Number of packets discarded because of bad packet header length

This device characteristic records the number of IP input packets discarded because the IP header length was invalid; either less than 20 bytes or greater than the length of the packet. The value must be numeric in decimal format with a length of at least 4 bytes.

DNSADDR - Primary Domain Name Server IP address

This device characteristic specifies the IP address of the primary Domain Name Server for this system. If a value is not specified for this device characteristic, the standard XOS Domain Name Server IP address is used. This is node 1 on the same sub-net as the system sending the request. The XOS ARP routines treat this address specially, allowing automatic routing to a name server on the same sub-net. This is the usual case for XOS only networks. If the system is to use a Domain Name Server running on a foreign system (such as Unix), this device characteristic is used to specify that system. The value must be a byte list in decimal format with a length of at least 4 bytes.

DOMAIN - Domain Name for system

This device characteristic specifies the Domain Name for the system. A Domain Name must be specified if the system is to use the Domain Name System for addressing other system or if it is to be addressable by Domain Name by other systems. The name is specified as a list of names separated by periods, just as Domain Names are normally written. The value must be a string with a buffer length of at least 64 bytes.

HOSTDOWN - Primary host system is down

This device characteristic specifies the state of the primary host for this system. The primary host is the remote system specified by the default remote IP address (RMTADDR) device characteristic. Possible values for this device characteristic are YES and NO. If the value of this characteristic is YES, any attempt to establish a TCP connection to the default remote IP address will fail with an ER_NHSNA error. Also, any attempt to send a UDP datagram to that address will fail if the UDP device was opened with the O$FNR bit (fail if not ready) set. The network drivers never change

the value of this device characteristic. It is intended to be set and cleared by a user mode program which is keeping track of the availability of the primary host system. This allows requests to access that system to fail immediately when it is not available, instead of requiring a time out of 10 to 20 seconds or more. The initial value of this device characteristic is NO, which means that it has no effect on the operation of the network interface unless the value is changed by a program. The value must be text with a length of at least 4 bytes.

INTRFACE - Network interface device

This read-only device characteristic returns the name of the NET class device which the IPS class device is using as a network interface. The name is returned without a trailing colon. The value must be text with a length of at least 8 bytes.

IPPROT - IP protocol

This device characteristic specifies the network protocol to be used for IP packets. This protocol is normally not specified, since the system defaults to the correct standard IP protocol value. The value must be a byte list in hex format with a length of at least 2 bytes.

IPADDR - IP address

This device characteristic specifies the IP address for this system. This must be specified if this system is to be part of a network. The value can be changed at any time, although changing after the system has been connected to a network can cause undefined network behavior. This is not recommended. The value must be a byte list in decimal format with a length of at least 4 bytes.

NAMESRVR - Domain Name Server on this system

This device characteristic specifies if a Domain Name Server is running on this system. Valid values are YES and NO. If the value is YES, the ARP routines will respond to requests for node 1 on the same sub-net as this system; if it is NO, it will not respond to this address. The initial value for this characteristic is NO. The Domain Name Server sets it to YES during initialization. The value must be text with a length of at least 4 bytes.

NODST - Number of packets discarded because no destination for protocol

This device characteristic records the total of IP input packets discarded because there is no destination for the IP protocol, i.e., that there is no process or device driver that has registered to handle the IP protocol specified

in the packet. The value must be numeric in decimal format with a length of at least 4 bytes.

PKTIN - Number of packets input

This device characteristic records the total number of IP packets input. The value must be numeric in decimal format with a length of at least 4 bytes.

PKTOUT - Number of packets output

This device characteristic records the total number of IP packets output. The value must be numeric in decimal format with a length of at least 4 bytes.

PSLTDL - Number of packets discarded because of bad data length

This device characteristic records the total number of IP input packets discarded because the data length in the IP header specified a length longer than the packet. The value must be numeric in decimal format with a length of at least 4 bytes.

PSLTMN - Number of packets discarded because packet less than minimum

This device characteristic records the total number of IP input packets discarded because the data length in the IP header specified a length too short to include the IP header. The value must be numeric in decimal format with a length of at least 4 bytes.

RMTADDR - Default remote IP address

This device characteristic specifies the default remote IP address. This address is used whenever a request to send data over the network does not specify an IP address. It also interacts with the HOSTDOWN device characteristic as described in the description of that device characteristic. The value of this device characteristic is initially 0, in which case the IP address must always be specified. The value must be a byte list in hex format with a length of at least 4 bytes.

RTPURGE - IP routing cache purge request

This device characteristic is used to manage the Internet Protocol routing cache. Its behavior is somewhat unique compared to most other device characteristics. The value returned is only meaningful when a value is also specified. When a value is specified, it is taken as an IP address. If that address is currently in the IP address cache, it is removed from the cache and a value of 1 is returned. If it is not in the cache, a value of 0 is returned. If

a value of 0 is specified, the IP routing cache is cleared and a value of 1 is returned. The value must be numeric in hex format with a length of at least 4 bytes.

RTSIZE - IP routing cache size

This read-only device characteristic returns the size of the IP routing cache. The IP routing cache is allocated when the IPS device is created and cannot be changed. The value must be numeric in decimal format with a length of at least 4 bytes.

RTUSE - IP routing cache usage

This read-only device characteristic returns the number of entries currently in use in the IP routing cache. The value must be numeric in decimal format with a length of at least 4 bytes.

SUBNET - IP subnet address mask

This device characteristic specifies the IP sub-net address mask. This value defines which IP address bits specify a node on the local sub-net and which specify the sub-net or network. A 1 bit indicates sub-net or network specification and a 0 bit indicates node specification. For example, a value of 0FF.FF.FF.0 indicates that the fourth byte of the IP address specifies a node on the local sub-net and the first three bytes specify the sub-net or network. This is probably the most commonly used sub-net mask value. It corresponds to a class C Internet address. If this value is not specified, 0.0.0.0 is used, meaning that the entire address is used to specify nodes. The value must be a byte list in hex format with a length of at least 4 bytes.

UDP Device Characteristics

This section describes the device characteristics which are specific to UDP class devices. The UDP devices provide access to the UDP protocol. These devices are commonly used to send and receive datagrams. While multiple UDP DCBs are created as needed to allow multiple processes to access the network, the various error and usage counters accessed with these device characteristics reflect the total usage at the UDP protocol level.

Table 13.24 summarizes the device characteristics for the UDP class devices.

Table 13.24 - Device characteristics for UDP class devices

Name	Fnc	Format	Size	Description
IPSDEV	G			IPS device
IPPROT	GS			IP protocol value
RTREMOVE	GS			IP routing table remove request
HOSTDOWN	GS	TEXT	4	Primary host system is down
NAMESRVR	GS	TEXT	4	Domain Name Server on this system
PKTIN	GS	DECV	4	Number of packets input
BYTEIN	GS	DECV	4	Number of bytes input
PKTOUT	GS	DECV	4	Number of packets output
BYTEOUT	GS	DECV	4	Number of bytes output
CHKSUM	GS	DECV	4	Discarded - bad data checksum
NODST	GS	DECV	4	Discarded - no destination for port
IBLXCD	GS			Discarded - input buffer limit exceeded
BADHDR	GS	DECV	4	Discarded - bad header
PSLTMN	GS	DECV	4	Discarded - packet minimum
PSLTDL	GS	DECV	4	Discarded - packet data length

Following is a detailed description of each device characteristic for UDP class devices.

BADHDR - Number of packets discarded because of bad header

This device characteristic records the number of UDP input packets discarded because of an illegal header format. This value must be numeric in decimal format with a length of at least 4 bytes.

BYTEIN - Number of bytes input

This device characteristic records the total number of UDP bytes input. This value must be numeric in decimal format with a length of at least 4 bytes.

BYTEOUT - Number of bytes output

This device characteristic records the total number of UDP bytes output. This value must be numeric in decimal format with a length of at least 4 bytes.

CHKSUM - Number of packets discarded because of bad UDP checksum

This device characteristic records the total number of UDP input packets discarded because of a bad UDP data checksum. This value must be numeric in decimal format with a length of at least 4 bytes.

HOSTDOWN - Primary host system is down

This device characteristic exactly duplicates the HOSTDOWN device characteristic for IPS class devices. It is duplicated here to make it easier for a program to indicate the state of the primary host system when sending datagrams using a UDP device. If it were not available here, it would be necessary to open an IPS device just for this purpose. This value must be text with a length of at least 4 bytes.

NAMESRVR - Domain Name Server on this system

This device characteristic exactly duplicates the NAMESRVR device characteristic for IPS class devices. It is duplicated here to make it easier for the Domain Name Server program to set this value. This program uses UDP devices but does not normally use any IPS devices. Having this device characteristic available eliminates the need to open an IPS device just to set this value. This value must be text with a length of at least 4 bytes.

NODST - Number of packets discarded because no destination for UDP port

This device characteristic records the total number of UDP input packets because there is no destination for the UDP port. This occurs when there is no process with a pending input request for the UDP port specified as the destination port in a UDP input packet. This value must be numeric in decimal format with a length of at least 4 bytes.

PKTIN - Number of packets input

This device characteristic records the total number of UDP packets input. This value must be numeric in decimal format with a length of at least 4 bytes.

PKTOUT - Number of packets output

This device characteristic records the total number of UDP packets output. This value must be numeric in decimal format with a length of at least 4 bytes.

PSLTDL - Number of packets discarded because of bad data length

This device characteristic records the total number of UDP packets discarded because the data length specified in the UDP header was not consistent with the length of the IP packet. This value must be numeric in decimal format with a length of at least 4 bytes.

PSLTMN - Number of packets discarded because packet less than minimum

This device characteristic records the total number of UDP packets discarded because the IP packet did not contain at least 8 bytes, which is the length of the UDP header. This value must be numeric in decimal format with a length of at least 4 bytes.

RTPURGE - IP routing table purge request

This device characteristic exactly duplicates the RTPURGE device characteristic for IPS class devices. It is duplicated here to make it easier for a program to clear the IP routing table when a response is not received to a datagram sent using a UDP device. If it were not available here, it would be necessary to open an IPS device just for this purpose. This value must be numeric in decimal format with a length of at least 4 bytes.

TCP Device Characteristics

This section describes the device characteristics which are specific to TCP class devices. The TCP devices provide access to the TCP protocol. These devices are used by programs (such as servers) which need to directly access TCP virtual connections. While multiple TCP DCBs are created as needed to allow multiple processes to access the network, the various error and usage counters accessed with these device characteristics reflect the total usage at the TCP protocol level.

Table 13.25 summarizes the device characteristics for TCP class devices.

Table 13.25 - Device characteristics for TCP class devices

Name	Fnc	Format	Size	Description
BADHDR	GS	DECV	4	Number of packets discarded because of bad header
BYTEIN	GS	DECV	4	Number of bytes input
BYTEOUT	GS	DECV	4	Number of bytes output
CHKSUM	GS	DECV	4	Number of packets discarded because of bad TCP checksum
CLOST	GS	DECV	4	Number of lost connections
FLOWOVR	GS	DECV	4	Number of packets discarded because of flow control overrun
MERGED	GS	DEVC	4	Number of merged input packets
NOACK	GS	DECV	4	Number of packets discarded because ACK not indicated
NODST	GS	DECV	4	Number of packets discarded because no destination for TCP port
OOSMAX	GS	DECV	4	Maximum number of out-of-sequence input packets queued
OOSMRGD	GS	DECV	4	Number of out-of-sequence input packets merged
OOSNUM	GS	DECV	4	Number of out-of-sequence input packets currently queued
OUTSEQ	GS	DECV	4	Number of packets discarded because of out of sequence packet
OUTWIN	GS	DECV	4	Number of packets discarded because of out of window packet
PKTIN	GS	DECV	4	Number of packets input
PKTOUT	GS	DECV	4	Number of packets output
PSLTHL	GS	DECV	4	Number of packets discarded because of bad packet header length
PSLTMN	GS	DECV	4	Number of packets discarded because packet less than minimum
RETRY1	GS	DECV	4	First TCP re-transmission threshold

Table 13.25 - Device characteristics for TCP class devices

Name	Fnc	Format	Size	Description
RETRY2	GS	DECV	4	Second TCP re-transmission threshold
REXMIT	GS	DECV	4	Number of re-transmitted packets
RSTRCVD	GS	DECV	4	Number of resets received
RSTSENT	GS	DECV	4	Number of resets sent
UNXFIN	GS	DECV	4	Number of unexpected FIN packets received

Following is a detailed description of each device characteristic for TCP class devices.

BADHDR - Number of packets discarded because of bad header

This device characteristic records the total number of TCP input packets discarded because of a bad header format. The value must be numeric in decimal format with a length of at least 4 bytes.

BYTEIN - Number of bytes input

This device characteristic records the total number of TCP bytes input. The value must be numeric in decimal format with a length of at least 4 bytes.

BYTEOUT - Number of bytes output

This device characteristic records the total number of TCP bytes output. The value must be numeric in decimal format with a length of at least 4 bytes.

CHKSUM - Number of packets discarded because of bad TCP checksum

This device characteristic records the total number of TCP packets discarded because of a bad TCP data checksum value. The value must be numeric in decimal format with a length of at least 4 bytes.

CLOST - Number of lost connections

This device characteristic records the total number of lost TCP connections. A lost connections is one that is terminated because the remote system failed to respond in the required length of time. The value must be numeric in decimal format with a length of at least 4 bytes.

FLOWOVR - Number of packets discarded because of flow control overrun

This device characteristic records the number of input packets that have been discarded because of flow control overflow, that is, packets which

are beyond the receive window. The value must be numeric in decimal format with a length of at least 4 bytes.

MERGED - Number of merged input packets

This device characteristic records the number of input packets which have been merged. Input packets are merged to allow more efficient memory usage. This is especially important when a sequence of small packets are received. The value must be numeric in decimal format with a length of at least 4 bytes.

NOACK - Number of packets discarded because ACK not indicated

This device characteristic records the total number of TCP packets discarded because the ACK bit was not set in the TCP header when it should have been set. The value must be numeric in decimal format with a length of at least 4 bytes.

NODST - Number of packets discarded because no destination for TCP port

This device characteristic records the total number of TCP input packets discarded because there was no destination for the TCP port. This will occur if no process has opened a TCP device using the port or no device driver has registered to handle the port. The value must be numeric in decimal format with a length of at least 4 bytes.

OOSMAX - Maximum number of out-of-sequence input packets queued

This device characteristic records the maximum number of out of sequence input packets that have been queued at one time. The value must be numeric in decimal format with a length of at least 4 bytes.

OOSMRGD - Number of out-of-sequence input packets merged

This device characteristic records the number of out of sequence input packets that have been merged. The value must be numeric in decimal format with a length of at least 4 bytes.

OOSNUM - Number of out-of-sequence input packets currently queued

This device characteristic records the number of out of sequence input packets that are currently queued. The value must be numeric in decimal format with a length of at least 4 bytes.

OUTSEQ - Number of packets discarded because of out of sequence packet

This device characteristic records the total number of TCP input packets discarded because they were out of sequence. The value must be numeric in decimal format with a length of at least 4 bytes.

OUTWIN - Number of packets discarded because of out of window packet

This device characteristic records the total number of TCP input packets discarded because they were outside of the receive window. The value must be numeric in decimal format with a length of at least 4 bytes.

PKTIN - Number of packets input

This device characteristic records the total number of TCP packets input. The value must be numeric in decimal format with a length of at least 4 bytes.

PKTOUT - Number of packets output

This device characteristic records the total number of TCP packets output. The value must be numeric in decimal format with a length of at least 4 bytes.

PSLTHL - Number of packets discarded because of bad packet header length

This device characteristic records the total number of TCP input packets discarded because of an invalid TCP header length. The value must be numeric in decimal format with a length of at least 4 bytes.

PSLTMN - Number of packets discarded because packet less than minimum

This device characteristic records the total number of TCP input packets discarded because the IP packet was not large enough to contain a complete minimum TCP header (20 bytes). The value must be numeric in decimal format with a length of at least 4 bytes.

RETRY1 - First TCP re-transmission threshold

This device characteristic specifies the first TCP re-transmission threshold. This is the number of times a packet is retransmitted before re-routing is attempted. The default value is 2. The value must be numeric in decimal format with a length of at least 4 bytes.

RETRY2 - Second TCP re-transmission threshold

This device characteristic specifies the second TCP re-transmission threshold. This is the total number of times a packet is re-transmitted before the connection is terminated. This includes any re-transmissions before reaching the first re-transmission threshold. If this value is less than or equal to the RETRY1 value, re-routing will not be done. The default value is 5. The value must be numeric in decimal format with a length of at least 4 bytes.

REXMIT - Number of retransmitted packets

This device characteristic records the total number of TCP output packets which have been retransmitted. The value must be numeric in decimal format with a length of at least 4 bytes.

RSTRCVD - Number of resets received

This device characteristic records the total number of TCP reset packets input. The value must be numeric in decimal format with a length of at least 4 bytes.

RSTSENT - Number of resets sent

This device characteristic records the total number of TCP reset packets output. The value must be numeric in decimal format with a length of at least 4 bytes.

UNXFIN - Number of unexpected FIN packets received

This device characteristic records the total number of unexpected FIN packets that have been received. A FIN packet is considered unexpected when it is received on a port which does not have a connection. The value must be numeric in decimal format with a length of at least 4 bytes.

TLN Device Characteristics

This section describes the device characteristics for TLN class devices. The TLN device implements a basic Telnet server as part of the XOS kernel. It actually consists of a pair of devices: the TLN device and the corresponding TRM class device which provides the functionality of a serial port TRM device. The TLN device cannot do input or output directly, but is used only to access its device characteristics which are used to control the operation of the Telnet server. These device characteristics are summarized in Table 13.26.

Table 13.26 - Device Characteristics for TLN Class Devices

Name	Fnc	Format	Size	Description
BYTEIN	GS	DECV	4	Number of bytes input
BYTEOUT	GS	DECV	4	Number of bytes output
INLBS	GS	DECV	4	Input line buffer size
INRBS	GS	DECV	4	Input ring buffer size
OUTRBS	GS	DECV	4	Output ring buffer size
PASSWORD	GS	STR	12	System level password
PROTERR	GS	DECV	4	Number of protocol errors seen
PROGRAM	GS	TEXT	16	Initial program to run
RETRY1	GS	DECV	4	First TCP re-transmission threshold
RETRY2	GS	DECV	4	Second TCP re-transmission threshold
SESSION	GS	TEXT	4	Allow session on terminal
TLNPORT	GS	DECV	4	Telnet port number

Following is a detailed description of the device characteristics defined for the TLN device class.

BYTEIN - Number of bytes input

This device characteristic records the total number of Telnet bytes input. The value must be numeric in decimal format with a length of at least 4 bytes.

BYTEOUT - Number of bytes output

This device characteristic records the total number of Telnet bytes output. The value must be numeric in decimal format with a length of at least 4 bytes.

INLBS - Input line buffer size

This device characteristic specifies the size of the line buffer which is allocated when a connection is established to the Telnet server. This will be the value returned by the INLBS device characteristic for the terminal class device associated with the TLN device. The value must be numeric in decimal format with a length of at least 4 bytes.

INRBS - Input ring buffer size

This device characteristic specifies the size of the input ring buffer which is allocated when a connection is established to the Telnet server. This buffer is used to hold type ahead data. This will be the value returned by the INRBS device characteristic for the terminal class device associated with the TLN device. The value must be numeric in decimal format with a length of at least 4 bytes.

OUTRBS - Output ring buffer size

This device characteristic specifies the size of the output ring buffer which is allocated when a connection is established to the Telnet server. This buffer is used to hold output data. This will be the value returned by the OUTRBS device characteristic for the terminal class device associated with the TLN device. The value must be numeric in decimal format with a length of at least 4 bytes.

PASSWORD - System level password

This device characteristic specifies the system level password associated with the terminal device. If a password is specified, it must be entered whenever a session is started on the terminal, that is, whenever a connection is established to the Telnet server. The value must be a string with a buffer length of at least 12 bytes.

PROTERR - Number of protocol errors seen

This device characteristic reports the total number of Telnet protocol errors detected. The value must be numeric in decimal format with a length of at least 4 bytes.

PROGRAM - Initial program to run

This device characteristic specifies the name of the program to run initially when a session is started on the terminal. If no name is specified, either SHELL.IMG or LOGIN.IMG is run, depending on if user logins are enabled for the system. Only a name can be specified. The program must

be in the directory specified by the XOSSYS: logical name. The value must be a string with a buffer length of at least 16 bytes.

RETRY1 - First TCP re-transmission threshold

This device characteristic specifies the first TCP re-transmission threshold. This is the number of times a packet is re-transmitted before re-routing is attempted. The default value is 2. The value must be numeric in decimal format with a length of at least 4 bytes.

RETRY2 - Second TCP re-transmission threshold

This device characteristic specifies the second TCP re-transmission threshold. This is the total number of times a packet is re-transmitted before the connection is terminated. This includes any re-transmissions before reaching the first re-transmission threshold. If this value is less than or equal to the RETRY1 value, re-routing will not be done. The default value is 5. The value must be numeric in decimal format with a length of at least 4 bytes.

SESSION - Allow session on terminal

This device characteristic specifies if the associated TRM device can be used to control a user session. Normally, this characteristic will have the value YES or NO. If the value is YES, any input from the serial port while the port is idle causes the system to create a new process and run a command shell (or the login program if user login is enabled for the system) with the port as the controlling terminal. If the value is NO, input from the serial port while it is idle is ignored. The value may also be set to any sequence of characters beginning with an underscore character. In this case, the system will send an IPM message to the IPM name formed by removing the underscore from the characteristic value whenever there is input from the serial port and it is idle. The user must have a program running which has opened an IPM device with this name and is prepared to receive the messages. This is intended to provide a method of implementing non-standard terminal based systems. The value must be text with a length of at least 4 bytes.

TLNPORT - Telnet port number

This device characteristic specifies the port number on which the Telnet server listens for connections. The default value is 23, which is the standard Telnet public port. The value must be numeric in decimal format with a length of at least 4 bytes.

RCP Device Characteristics

This section describes the device characteristics which are specific to RCP class devices. The RCP devices provide access to the RCP protocol. These devices are used by programs (such as servers) which need to directly access RCP virtual connections. While multiple RCP DCBs are created as needed to allow multiple processes to access the network, the various error and usage counters accessed with these device characteristics reflect the total usage at the RCP protocol level.

The RCP protocol provides a reliable, packet oriented data path. It can be thought of as a packet oriented version of TCP. It is layered on top of UDP, mainly to make it ieasy to implement on systems which do not support direct access to the IP protocol level

Table 13.27 summarizes the device characteristics for RCP class devices.

Table 13.27 - Device characteristics for RCP class devices

Name	Fnc	Format	Size	Description
BADHDR	GS	DECV	4	Number of packets discarded because of bad header
BYTEIN	GS	DECV	4	Number of bytes input
BYTEOUT	GS	DECV	4	Number of bytes output
CHKSUM	GS	DECV	4	Number of packets discarded because of bad TCP checksum
CLOST	GS	DECV	4	Number of lost connections
FLOWOVR	GS	DECV	4	Number of packets discarded because of flow control overrun
MERGED	GS	DEVC	4	Number of merged input packets
NOACK	GS	DECV	4	Number of packets discarded because ACK not indicated
NODST	GS	DECV	4	Number of packets discarded because no destination for TCP port
OOSMAX	GS	DECV	4	Maximum number of out-of-sequence input packets queued
OOSMRGD	GS	DECV	4	Number of out-of-sequence input packets merged
OOSNUM	GS	DECV	4	Number of out-of-sequence input packets currently queued
OUTSEQ	GS	DECV	4	Number of packets discarded because of out of sequence packet
OUTWIN	GS	DECV	4	Number of packets discarded because of out of window packet
PKTIN	GS	DECV	4	Number of packets input

Table 13.27 - Device characteristics for RCP class devices

Name	Fnc	Format	Size	Description
PKTOUT	GS	DECV	4	Number of packets output
PSLTHL	GS	DECV	4	Number of packets discarded because of bad packet header length
PSLTMN	GS	DECV	4	Number of packets discarded because packet less than minimum
RETRY1	GS	DECV	4	First TCP re-transmission threshold
RETRY2	GS	DECV	4	Second TCP re-transmission threshold
REXMIT	GS	DECV	4	Number of re-transmitted packets
RSTRCVD	GS	DECV	4	Number of resets received
RSTSENT	GS	DECV	4	Number of resets sent
UNXFIN	GS	DECV	4	Number of unexpected FIN packets received

Following is a detailed description of each device characteristic for TCP class devices.

BADHDR - Number of packets discarded because of bad header

This device characteristic records the total number of TCP input packets discarded because of a bad header format. The value must be numeric in decimal format with a length of at least 4 bytes.

BYTEIN - Number of bytes input

This device characteristic records the total number of TCP bytes input. The value must be numeric in decimal format with a length of at least 4 bytes.

BYTEOUT - Number of bytes output

This device characteristic records the total number of TCP bytes output. The value must be numeric in decimal format with a length of at least 4 bytes.

CHKSUM - Number of packets discarded because of bad TCP checksum

This device characteristic records the total number of TCP packets discarded because of a bad TCP data checksum value. The value must be numeric in decimal format with a length of at least 4 bytes.

CLOST - Number of lost connections

This device characteristic records the total number of lost TCP connections. A lost connections is one that is terminated because the remote sys-

tem failed to respond in the required length of time. The value must be numeric in decimal format with a length of at least 4 bytes.

FLOWOVR - Number of packets discarded because of flow control overrun

This device characteristic records the number of input packets that have been discarded because of flow control overflow, that is, packets which are beyond the receive window. The value must be numeric in decimal format with a length of at least 4 bytes.

MERGED - Number of merged input packets

This device characteristic records the number of input packets which have been merged. Input packets are merged to allow more efficient memory usage. This is especially important when a sequence of small packets are received. The value must be numeric in decimal format with a length of at least 4 bytes.

NOACK - Number of packets discarded because ACK not indicated

This device characteristic records the total number of TCP packets discarded because the ACK bit was not set in the TCP header when it should have been set. The value must be numeric in decimal format with a length of at least 4 bytes.

NODST - Number of packets discarded because no destination for TCP port

This device characteristic records the total number of TCP input packets discarded because there was no destination for the TCP port. This will occur if no process has opened a TCP device using the port or no device driver has registered to handle the port. The value must be numeric in decimal format with a length of at least 4 bytes.

OOSMAX - Maximum number of out-of-sequence input packets queued

This device characteristic records the maximum number of out of sequence input packets that have been queued at one time. The value must be numeric in decimal format with a length of at least 4 bytes.

OOSMRGD - Number of out-of-sequence input packets merged

This device characteristic records the number of out of sequence input packets that have been merged. The value must be numeric in decimal format with a length of at least 4 bytes.

OOSNUM - Number of out-of-sequence input packets currently queued

This device characteristic records the number of out of sequence input packets that are currently queued. The value must be numeric in decimal format with a length of at least 4 bytes.

OUTSEQ - Number of packets discarded because of out of sequence packet

This device characteristic records the total number of TCP input packets discarded because they were out of sequence. The value must be numeric in decimal format with a length of at least 4 bytes.

OUTWIN - Number of packets discarded because of out of window packet

This device characteristic records the total number of TCP input packets discarded because they were outside of the receive window. The value must be numeric in decimal format with a length of at least 4 bytes.

PKTIN - Number of packets input

This device characteristic records the total number of TCP packets input. The value must be numeric in decimal format with a length of at least 4 bytes.

PKTOUT - Number of packets output

This device characteristic records the total number of TCP packets output. The value must be numeric in decimal format with a length of at least 4 bytes.

PSLTHL - Number of packets discarded because of bad packet header length

This device characteristic records the total number of TCP input packets discarded because of an invalid TCP header length. The value must be numeric in decimal format with a length of at least 4 bytes.

PSLTMN - Number of packets discarded because packet less than minimum

This device characteristic records the total number of TCP input packets discarded because the IP packet was not large enough to contain a complete minimum TCP header (20 bytes). The value must be numeric in decimal format with a length of at least 4 bytes.

RETRY1 - First TCP re-transmission threshold

This device characteristic specifies the first TCP re-transmission threshold. This is the number of times a packet is retransmitted before re-routing is attempted. The default value is 2. The value must be numeric in decimal format with a length of at least 4 bytes.

RETRY2 - Second TCP re-transmission threshold

This device characteristic specifies the second TCP re-transmission threshold. This is the total number of times a packet is re-transmitted before the connection is terminated. This includes any re-transmissions before reaching the first re-transmission threshold. If this value is less than or equal to the RETRY1 value, re-routing will not be done. The default value is 5. The value must be numeric in decimal format with a length of at least 4 bytes.

REXMIT - Number of retransmitted packets

This device characteristic records the total number of TCP output packets which have been retransmitted. The value must be numeric in decimal format with a length of at least 4 bytes.

RSTRCVD - Number of resets received

This device characteristic records the total number of TCP reset packets input. The value must be numeric in decimal format with a length of at least 4 bytes.

RSTSENT - Number of resets sent

This device characteristic records the total number of TCP reset packets output. The value must be numeric in decimal format with a length of at least 4 bytes.

UNXFIN - Number of unexpected FIN packets received

This device characteristic records the total number of unexpected FIN packets that have been received. A FIN packet is considered unexpected when it is received on a port which does not have a connection. The value must be numeric in decimal format with a length of at least 4 bytes.

XFP Device Characteristics

This section describes the device characteristics which are specific to XFP class devices. The XFP devices implement an XFP client which is used to transparently access remote file systems. To a program, an XFP device looks almost exactly like a local file structured disk. Normally, XFP devices are transparent to device characteristics, that is, device characteristics specified for an XFP device apply to the remote file system being accessed, not to the XFP device itself. If an XFP device is opened without a network address or file specification, any device characteristics referenced apply directly to the XFP device. These XFP device characteristics are described in this section. While multiple XFP DCBs are created as needed to allow multiple processes to access the network, the various error and usage counters accessed with these device characteristics reflect the total usage at the XFP protocol level.

Table 13.27 summarizes the device characteristics for XFP class devices.

Table 13.27 - Device characteristics for XFP class devices

Name	Fnc	Format	Size	Description
BYTEIN	S	DECV	4	Number of bytes input
BYTEOUT	S	DECV	4	Number of bytes output
PKTIN	S	DECV	4	Number of messages input
PKTOUT	S	DECV	4	Number of messages output
PROERR	S	DECV	4	Number of protocol errors
RCPDEV	S	TEXT	16	RCP device
RETRY1	S	DECV	1	First retransmission threshold
RETRY2	S	DECV	1	Second retransmission threshold
RMTPORT	S	DECV	4	Remote server port

Following is a detailed description of each device characteristic for XFP class devices.

BYTEIN - Number of bytes input

This device characteristic records the total number of XFP bytes input. The value must be numeric in decimal format with a length of at least 4 bytes.

BYTEOUT - Number of bytes output

This device characteristic records the total number of XFP bytes output. The value must be numeric in decimal format with a length of at least 4 bytes.

PKTIN - Number of messages input

This device characteristic records the total number of XFP messages input. The value must be numeric in decimal format with a length of at least 4 bytes.

PKTOUT - Number of messages output

This device characteristic records the total number of XFP messages output. The value must be numeric in decimal format with a length of at least 4 bytes.

PROERR - Number of protocol errors

This device characteristic records the total number of XFP protocol errors which have occurred. The value must be numeric in decimal format with a length of at least 4 bytes.

RCPDEV - RCP device

This device characteristic specifies the RCP device to use for connection to the remote XFP server.

RETRY1 - First retransmission threshold

This device characteristic specifies the first retransmission threshold for the XFP protocol.

RETRY2 - Second retransmission threshold

This device characteristic specifies the second retransmission threshold for the XFP protocol.

RMTPORT - Remote server port

This device characteristic specifies the remote RCP port used to contact an XFP server. This is a global value. When set, it is permanent and applies to all XFP connections from the device, not just the currently open XFP device. Normally, this value should not be set, since it defaults to the correct standard value. It may be changed if necessary to communicate with a

non-standard XFP server. The value must be numeric in decimal format with a length of at least 2 bytes.

Chapter 14

Add-Unit Characteristics

Add-unit characteristics have the same format as device characteristics. The are used to specify various characteristcis when adding new devices using the CF_ADDUNIT sub-function of the QFNC_CLASSFUNC function of the svcIoQueue system call. While the format is the same as that of device characteristics, only get PAR_SET function may be used. The PAR_GET function is not allowed and descriptions are not returned. Since these characteristics cannot be returned, the size specified is only advisory. Any size value field can be used which will contain the value being specified.

In most cases, there are devcie characteristics with the same names as the add-unit characteristics which generally allow the corresponding value to be read but not modified. There are a few exceptions to this, but generally, values which can be modified are set using device characteristics after a device has been added rather that with an add-unit characteristic.

Many add-unit characteristics are required when adding a device. These are indicated with a Y in the *Req* column in the following tables. Other characteristics are optional and a reasonal default value is used if the characteristic is not specified.

There are a number of add-unit characteristics that are global in the sense that they are used by many different devices and when used by a device, always have the same meaning. These characteristics are summerized in table 14.1 below.

Table 14.1 - Common add-unit characteristics				
Name	Req	Format	Size	Description
DMA	Y	DECV	1	DMA channel number
INDEX	Y	DECV	1	Index of unit on controller
INT	Y	DECV	1	Interrupt number
IOREG	Y	HEXV	2	Base I/O register number
MEM	Y	HEXV	4	Base memory address
TYPE	Y	TEXT	4	Device type
UNIT	Y	DECV	1	Unit number

These common characteristis are also listed in the tables for the individual device types where they are allowed or required, but are only defined in detail here.

DMA - DMA channel number

This characteristic is used to specify the ISA bus DMA channel used by DMA devices. It is only used by devices which use DMA and is generally required by all such devices. It can have a numeric value of 1 to 3 (for 8-bit DMA channels) or 5 to 7 (for 16-bit DMA channels).

INDEX - Index of unit on controller

This characteristic is used by devices which support multiple device units connected to a single controller and is required by all such devices. Generally, the first unit is specified by a value of 1, the second by a value of 2, etc. The value must be numeric.

INT - Interrupt number

This characteristic is used by devices which use an interrupt which can be specified and is required by all such devices. It should be noted that a small number of devices (such as the AT floppy disk controller) have a fixed interrupt number assignment and thus do not use the characteristic. It must have a numeric value when must be between 2 and 15. Some devices may not allow some values within this range.

IOREG - Base I/O register number

This characteristic is used by all devices which are associated with a particular set of hardware I/O registers. The value of this characteristic is the number of the base I/O register for the device. Devices which require specification of additional I/O register numbers define additional characteristics for this purpose. The value must be numeric. The value must never be greater than 0xFFFF and some devices restrict the value to a lower maximum.

TYPE - Device type

This device characteristic is used by device classes which use different device level drivers to support different types of hardware. Its value specifies the specific type of driver being used for the device. For example, XOS supports both the standard AT type floppy controller and the Compati-Card floppy controller. The value of this characteristic indicates which is being used for a specific floppy drive. This characteristic is not always required since in most cases there is a generally used default device type.

UNIT - Unit number

This add-unit characteristic does not directly correspond to a device characteristic. It is used to specify the unit number for the device unit being added to the class specified for the CF_ADDUNIT sub-function. It is required for all devices. Most device classes require that each device added to the class have a unique unit number, although some (such as the DISK class) create several naming spaces within the class and only require that the unit number be unique with the naming space. There is no requirement that unit numbers be contiguous or be assigned in any specific order. Many device classess restrict unit numbers to a value of 99 or less. Some device classes do not allow a unit number of 0 or reserve it for a special purpose.

The name of the device is generally created by prefixing the decimal representation of the unit number with the device name associated with the device class. This is often, but not necessarily the name of the class. A class is free to use any prefix or prefixes for the device name. For example, the DISK class uses F for floppy disks, D for IDE disks and S for SCSI disks. Nothing in XOS enforces this format however. A class driver can form a device name in any way, although it must ensure that the name is unique in the system.

DISK Add-Unit Characteristics

Due to the major differences in the architectures of the supported disk controllers, there are no additional common add-unit characteristics for disk class devices.

HDKA Type Disk Devices (PC-AT Hard Disk)

This section describes the add-unit characteristics which are specific to the HDKA type disk device driver. This device supports the standard PC-AT hard disk controller. This includes the original ST-501 controller, ESDI controllers, and IDE controllers. It does not include the PS-2 disk controllers or most SCSI controllers. SCSI controllers which fully emulate the PC-AT hard disk controller registers in hardware can be used, but none of the extra features of these controllers are supported by this device driver.

Table 14.2 summarizes the HDKA type specific add-unit characteristics.

Table 14.2 - Add-unit characteristics for HDKA type disks

Name	Req	Format	Size	Description
ICYLNS	*	DECV	4	Number of cylinders
IHEADS	*	DECV	4	Number of heads
INDEX	Y	DECV	1	Index on controller
INT	Y	DECV	1	Interrupt number
IOREG	Y	HEXV	2	Base IO register
ISECTS	*	DECV	4	Number of sectors
TYPE	Y	TEXT	4	Device type = HDKA
UNIT	Y	DECV	1	Unit number
WPCCYLN		DECV	4	Wriate pre-comp cylinder

The unique hard disk add-unit characteristics are described in detail below.

ICYLNS - Number of cylinders

This characteristic specifies the number of cylinders on the drive. This value is only used for disks with do not support the identify drive command. It is required for these disks unless there is a CMOS data entry describing the drive. The value must be numeric.

IHEADS - Number of heads

This characteristic specifies the number of heads on the drive. This value is only used for disks with do not support the identify drive command. It is required for these disks unless there is a CMOS data entry describing the drive. The value must be numeric.

ISECTS - Number of sectors

This characteristic specifies the number of sectors in each track on the drive. This value is only used for disks with do not support the identify drive command. It is required for these disks unless there is a CMOS data entry describing the drive. The value must be numeric.

WPCCYLN - Write pre-comp cylinder

This characteristic specifies the cylinder at which write pre-compensation begins. This value is ignored by almost all modern disk drives. It is required by a few very old drives for proper operation. If this characteristic is not specified and there is a CMOS data entry describing the drive the CMOS value will be used, thus there should very seldom be a need to use this characteristic. The value must be numeric.

SDSK Type Disk Devices (SCSI Controllers)

This section describes the add-unit characteristics which are specific to the SDSK type disk device driver. This device supports all disks which are connected to the system using a SCSI interface.

Table 14.3 summarizes the SDSK type disk add-unit characteristics.

Table 14.3 - Add-unit characteristics for SDSK type disks

Name	Req	Format	Size	Description
DOINQ		TEXT	4	Do inquiry function
SCSIDEV	Y	TEXT	16	SCSI device
SCSILUN		DECV	1	SCSI logical unit number
SCSITAR	Y	DECV	1	SCSI target ID
TYPE	Y	TEXT	4	Device type = SDSK
UNIT	Y	DECV	2	Unit number

DOINQ - Do inquiry function

This characteristic specifies if a SCSI inquiry function should be done to the drive to obtain its parameters. The default value is Y (Yes). This is correct for almost all SCSI disks. A few cases have been observed where a non-complient disk did not respond properly to an inquiry function and this characteristic has been included to handle this unusual case. The value must be text and should have a value of Y, YES, N, or NO.

SCSIDEV - SCSI device

This characteristic specifies the name of the scsi controller device for the disk. This will have the format SCSIn where n is the unit number of the SCSI controller device. The SCSI controller device must be added before the disk unit is added.

SCSILUN - SCSI logical unit number

This characteristic specifies the SCSI logical unit number for the disk. This characteristic is optional and has a default value of 0. The value must be numeric between 0 and 15 (0 and 7 for many SCSI controllers).

SCSITAR - SCSI target ID

This characteristic specifies the SCSI target number for the disk. The value must be numeric between 0 and 14 (0 and 6 for many SCSI controllers).

FDKA Type Disk Devices (Floppy Disk)

This section describes the add-unit characteristics for the FDKA type disk device. This device supports the standard PC-AT floppy disk controller (NEC 765/Intel 8272) and the CompatiCard-I add-in floppy disk controller made by Micro Solutions, Inc. This controller uses the same chip as the standard PC-AT controller but provides slightly more flexibility in supporting non-standard floppy types, especially 8" floppies.

The floppy disk geometery is not specified when adding a unit. It is determined automatically when a disk is mounted or can be specified explicitly with device characteristics.

Table 14.4 summarizes the floppy disk add-unit characteristics.

Table 14.4 - Device characteristics for floppy disks

Name	Req	Format	Size	Description
CONDESP		TEXT	4	Controller description
INDEX	Y	TEXT	8	Index on controller
IOREG	Y	DECV	2	Base IO register
TYPE	Y	DECV	2	Unit type = FKDA
UNIT	Y	DECV	1	Unit number
UNITTYPE	Y	TEXT	4	Unit type

The floppy disk add-unit characteristics are described in detail below.

CONDESP - Controller description

This characteristic specifies the type of floppy controller. Valid values are PCAT (standard PC/AT floppy controller) or CMPT (CompatiCard floppy controller). The default value is PCAT. The value must be text.

UNITTYPE - Unit type

This characteristic specifies the type of a floppy disk unit. While this value can be changed at any time using a device characteristic, it is settable here to ensure that the incorrect disk type will not be used when initially accessing the disk. If this characteristic is not specified, the CMOS value (if there is one) is used. If there is not CMOS value, HD5 is assumed. Valid values are:

Value	Meaning
HD3	3.5” high density
DD3	3.5" double density
HD5	5.25" high density
DD5	5.25" double density
DD8	8" double density

The value must be text with a length of at least 4 bytes.

SPL Add-Unit Characteristics

Other than the universally required UNIT characteristic there are not add-unit characteristics for SPL class devices. These devices are configured using device characteristics after the unit has been added.

TAPE Add-Unit Characteristics

Currently XOS only supports SCSI tape controllers. Additional types of controllers will probably be supported in future versions. The TAPE class driver is structured to support multiple low lever device drivers like the disk class driver.

STAP Type Tape Devices (SCSI Controllers)

This section describes the add-unit characteristics which are specific to the STAP type tape device driver. This device supports all tape drives which are connected to the system using a SCSI interface.

Table 14.5 summarizes the SDSK type disk add-unit characteristics.

Table 14.5 - Add-unit characteristics for STAP type tape drives

Name	Req	Format	Size	Description
SCSIDEV	Y	TEXT	16	SCSI device
SCSILUN		DECV	1	SCSI logical unit number
SCSITAR	Y	DECV	1	SCSI target ID
TYPE	Y	TEXT	4	Device type = SDSK
UNIT	Y	DECV	2	Unit number

SCSIDEV - SCSI device

This characteristic specifies the name of the scsi controller device for the tape drive. This will have the format SCSIn where n is the unit number of the SCSI controller device. The SCSI controller device must be added before the tape unit is added.

SCSILUN - SCSI logical unit number

This characteristic specifies the SCSI logical unit number for the tape drive. This characteristic is optional and has a default value of 0. The value must be numeric between 0 and 15 (0 and 7 for many SCSI controllers).

SCSITAR - SCSI target ID

This characteristic specifies the SCSI target number for the tape drive. The value must be numeric between 0 and 14 (0 and 6 for many SCSI controllers).

TRM Add-Unit Characteristics

The add-unit characteristics for the TRM class devices are different for each of the different types of low level devices supported. The section describes the add-unit characteristics for serial port SERA, serial port SERB/SERC, serial port SERD, and console (VGAA) devices. There are no add-unit characteistics for the psuedo-console and telnet terminal devices since these devices do not use the add-unit function.

TRM (Serial Port) Devices

XOS supports 4 different serial interface boards, each of which have somewhat different add-unit characterists. Each is listed seperately below.

All TRM class devices have names of the format TRMn regardless of the type of the low level device driver for the port. Thus the unit number for a TRM unit must be unique across all TRM unit types.

TRM (SERA) Devices

The SERA type serial device driver supports the standard PC architecture COM ports using the 16450/16550 and compatable chips.

Table 14.5 summarizes the TRM class serial port device characteristics.

Table 14.5 - Add-unit characteristics for TRM class serial ports

Name	Fnc	Format	Size	Description
INLBS		DECV	4	Input line buffer size
INRBS		DECV	4	Input ring buffer size
INT	Y	DECV	1	Interrupt number
INTRS		DECV	4	Interrupt level ring size
IOREG	Y	HEXV	4	Base IO register number
OUTRBS		DECV	4	Output ring buffer size
TYPE	Y	TEXT	8	Device type = SERA
UNIT	Y	DECV	2	Unit number

The following section describes the characteristics for TRM class SERA serial port devices in detail.

INLBS - Input line buffer size

This characteristic specifies the size of the input line buffer. The size of this buffer determines the maximum length line that can be input in line mode. If the chracteristic is not specified a value of 120 is used. This value is adequate for most cases. The value must be numeric.

INRBS - Input ring buffer size

This characteristic specifies the size of the input ring buffer for the serial port. This buffer is mainly used to hold typeahead input when in line mode. If the chracteristic is not specified a value of 120 is used. This value is adequate for most cases. The value must be numeric.

INTRBS - Interrupt ring buffer size

This characteristic specifies the size of the interrupt ring buffer for the serial port. This buffer is buffers all input until it can be processed. If the chracteristic is not specified a value of 100 is used. This value is adequate for most cases when the port is used to support an interactive terminal. If it is used for high speed data input, a larger value will probably be needed (depending on data rate and processor speed) to prevent data loss. The value must be numeric.

OUTRBS - Output ring buffer size

This characteristic specifies the size of the output ring buffer for the serial port. This buffer is used to internally buffer output to minimize process scheduling overhead when doing output. If the chracteristic is not specified a value of 100 is used. This value is adequate for for most cases. If heavy output is expected, it should be increased, especially if it is important that there be no pauses between output chracters The value must be numeric.

TRM (SERB, SERC) Devices

The SERB type serial device driver supports the DigiBoard multiport serial boards and the SERC type serial device driver support the Gtek multiport serial boards. These drivers are very similar and have the same add-unit characteristics. A TRM unit must be added for each port on a multiport board. The STSREG value identifies the board, since it is the one common register on each board. The INT value is also common to all ports on the board. It must be specified for the first unit added. It is optional for additional units on the same board, but if specified it must be the same as for the first unit. The IOREG value must match the value which the specified PORT is strapped for. Note that when 2 or 4 8-port DigiBoards are connected internally, the result appears to the system as a single board, with single STSREG and INT values.

Table 14.6 summarizes the TRM class serial port device characteristics.

Table 14.6 - Add-unit characteristics for TRM class SERB and SERC serial ports

Name	Fnc	Format	Size	Description
INLBS		DECV	4	Input line buffer size
INRBS		DECV	4	Input ring buffer size
INT	Y	DECV	1	Interrupt number
INTRS		DECV	4	Interrupt level ring size
IOREG	Y	HEXV	4	Base IO register number
OUTRBS		DECV	4	Output ring buffer size
PORT	Y	DECV	1	Port on board
STSREG	Y	TEXT	4	Status register number
TYPE	Y	TEXT	8	Device type = SERB or SERD
UNIT	Y	DECV	2	Unit number

The following section describes the characteristics for TRM class SERB and SERC serial port devices in detail.

INLBS - Input line buffer size

This characteristic specifies the size of the input line buffer. The size of this buffer determines the maximum length line that can be input in line mode. If the chracteristic is not specified a value of 120 is used. This value is adequate for most cases. The value must be numeric.

INRBS - Input ring buffer size

This characteristic specifies the size of the input ring buffer for the serial port. This buffer is mainly used to hold typeahead input when in line mode. If the chracteristic is not specified a value of 120 is used. This value is adequate for most cases. The value must be numeric.

INTRBS - Interrupt ring buffer size

This characteristic specifies the size of the interrupt ring buffer for the serial port. This buffer is buffers all input until it can be processed. If the chracteristic is not specified a value of 100 is used. This value is adequate for most cases when the port is used to support an interactive terminal. If it is used for high speed data input, a larger value will probably be needed (depending on data rate and processor speed) to prevent data loss. The value must be numeric.

OUTRBS - Output ring buffer size

This characteristic specifies the size of the output ring buffer for the serial port. This buffer is used to internally buffer output to minimize process scheduling overhead when doing output. If the chracteristic is not specified a value of 100 is used. This value is adequate for for most cases. If heavy output is expected, it should be increased, especially if it is important that there be no pauses between output chracters The value must be numeric.

PORT - Port on board

This characteristic specifies the port on the board. The first port is 1, the second 2, etc. The value must be numeric and must be less than or equal to the number of ports on the board.

STSREG - Status I/O register number (SERB only)

This characteristic specifies the I/O register number of the board’s status register. Each board has a single status register, regardless of the number of ports which it supports. The number of this status register is used by the system to identify the board (as opposed to the individual ports). The value must be numeric.

TRM (SERD) Devices

The SERD type serial device driver supports the RocketPort multiport serial interface boards. These boards are available in 8, 16, and 32 port versions.

Each board uses 2 or more seperate blocks of IO registers. All boards use a block of 4 board registers which an be assigned at any base address which is evenly divisable by 64 (0x40) and is 0xFC0 or less (BRDREG value). 8-port boards also use a single block of 64 registers which can also be assigned at any base address which is evenly divisable by 64 and is 0xFC0 or less (IOREG value). 16-port boards use two blocks of 64 registers. The first block can be assigned at any base address which is evenly divisable by 64 and is 0x7C0 or less (IOREG value). The second block of 64 registers is located at the address of the first block plus 0x400. 32-port boards use four blocks of 64 registers each. The first block can be assigned at any base address which is evenly divisable by 64 and is 0x3C0 or less (IOREG value). The second block of 64 registers is located at the address of first block plus 0x400, the third at the address of the first plus 0x800 and the fourth at the address of the first plus 0xC00.

Note that the above address limits are based on the number of ports initialized on a board, NOT on the physical size of the board.

All devices on a board must be specified with the same interrupt number, IO register address and board register address. The port number must be different for each port on a board and must be between 1 and 8 inclusive for an 8-port board, 1 and 16 inclusive for a 16-port board, and 1 and 32 inclusive for a 32 board board. Not all ports on a board need to be initialized and they can be initialized in any order.

Table 14.7 summarizes the TRM class SERD serial port device characteristics.

Table 14.7 - Add-unit characteristics for TRM class SERD serial ports

Name	Fnc	Format	Size	Description
BRDREG	Y	HEXV	4	Board register number
INLBS		DECV	4	Input line buffer size
INRBS		DECV	4	Input ring buffer size
INT	Y	DECV	1	Interrupt number
INTRS		DECV	4	Interrupt level ring size
IOREG	Y	HEXV	4	Base IO register number
OUTRBS		DECV	4	Output ring buffer size
PORT	Y	DECV	1	Port on board
TYPE	Y	TEXT	8	Device type = SERD

Table 14.7 - Add-unit characteristics for TRM class SERD serial ports				
Name	Fnc	Format	Size	Description
UNIT	Y	DECV	2	Unit number

The following section describes the characteristics for TRM class SERA serial port devices in detail.

BRDREG - Board register number

This characteristic specifies the register number for the board register which is described above. The value must be numeric.

INLBS - Input line buffer size

This characteristic specifies the size of the input line buffer. The size of this buffer determines the maximum length line that can be input in line mode. If the chracteristic is not specified a value of 120 is used. This value is adequate for most cases. The value must be numeric.

INRBS - Input ring buffer size

This characteristic specifies the size of the input ring buffer for the serial port. This buffer is mainly used to hold typeahead input when in line mode. If the chracteristic is not specified a value of 120 is used. This value is adequate for most cases. The value must be numeric.

INTRBS - Interrupt ring buffer size

This characteristic specifies the size of the interrupt ring buffer for the serial port. This buffer is buffers all input until it can be processed. If the chracteristic is not specified a value of 100 is used. This value is adequate for most cases when the port is used to support an interactive terminal. If it is used for high speed data input, a larger value may be needed (depending on data rate and processor speed) to prevent data loss. It should be remembered however, that the RocketPort provides a 1024 character input FIFO for each port. This usually eliminates the need for extra buffering in the system. The value must be numeric.

OUTRBS - Output ring buffer size

This characteristic specifies the size of the output ring buffer for the serial port. This buffer is used to internally buffer output to minimize process scheduling overhead when doing output. If the chracteristic is not specified a value of 100 is used. This value is adequate for for most cases. If heavy output is expected, it should be increased, especially if it is important that there be no pauses between output chracters It should be remem-

bered however, that the RocketPort provides a 256 character output FIFO for each port. This usually eliminates the need for extra buffering in the system The value must be numeric.

PORT - Port on board

This characteristic specifies the port on the board. The first port is 1, the second 2, etc. The value must be numeric and must be less than or equal to the number of ports on the board.

TRM (Console) Devices

This section describes the add-unit characteristics which are specific to TRM class console devices. Console devices are somewhat unusual in that they use two separate low level drivers, one for the display interface and one for the keyboard interface. The current of XOS only supports the VGA console display device (type is VGAA) and the standard PC/AT keyboard interface (type is KBDA).

Table 14.8 summarizes the TRM class console add-unit characteristics.

Table 14.8 - Add-unit characteristics for TRM class VGAA/KBDA consoles

Name	Req	Format	Size	Description
INLBS		DECV	4	Keyboard input line buffer size
INRBS		DECV	4	Keyboard input line buffer size
IOREG	Y	TEXT	4	Display base IO register number
KBINT	Y	DECV	4	Keyboard interrupt number
KBIOREG	Y	DECV	4	Keyboard base IO register number
KBTYPE	Y	TEXT	4	Keyboard unit type = KBDA
SCREEN	Y	STR	12	Display virtual screen number
TYPE	Y	STR	4	Display unit type = VGAA
UNIT	Y	TEXT	4	Unit number

The following section describes the dadd-unit characteristics for TRM class console devices in detail.

INLBS - Keyboard input line buffer size

This characteristic specifies the size of the keyboard input line buffer. The size of this buffer determines the maximum length line that can be input in line mode. If the chracteristic is not specified a value of 120 is used. This value is adequate for most cases. The value must be numeric.

INRBS - Keyboard input ring buffer size

This characteristic specifies the size of the keyboard input ring buffer for the serial port. This buffer is mainly used to hold typeahead input when in line mode. If the chracteristic is not specified a value of 120 is used. This value is adequate for most cases. The value must be numeric.

KBINT - Keyboard interrupt number

This characteristic specifies the keyboard interrupt number. It must be specified if this is the first virutal screen being defined. If it is not the first virtual screen this characteristic is optional, but if specified it must be the same as previously specified for the console. The value must be numeric.

KBIOREG - Keyboard base IO register number

This characteristic specifies the keyboard base IO register number.. It must be specified if this is the first virutal screen being defined. If it is not the first virtual screen this characteristic is optional, but if specified it must be the same as previously specified for the console. The value must be numeric.

KBTYPE - Keyboard device type

This characteristic specifies the unit type for the keyboard interface device. It must be specified if this is the first virutal screen being defined. If it is not the first virtual screen this characteristic is optional, but if specified it must be the same as previously specified for the console. Currently, the only valid value is KBDA. The value must be text.

SCREEN - Virtual screen number

This characteristic specifies the virtual screen number for the virtual screen being added. Virtual screens must be added in order, starting with virtual screen 1 and incrementing by 1 for each virtual screen added. The maximum virtual screen number is 32. The value must be numeric.

TRM (Pseudo-Console) Devices

There are no add-unit parameters associated with the client-side pseudo-console devices. These devices are created dynamically by the associated server-side pseudo-console device rather than by means of the CF_ADDUNIT sub-function.

TRM (Telnet) Devices

There are no add-unit parameters associated with the client-side Telnet devices. These devices are created dynamically by the associated server-side Telnet device rather than by means of the CF_ADDUNIT sub-function.

PCN Add-Unit Characteristics

Other than the universally required UNIT characteristic there are not add-unit characteristics for PCN class devices. These devices are configured using device characteristics after the unit has been added.

IPM Add-Unit Characteristics

The IPM device does not use the CF_ADDUNIT function so there are no add-unit characteristics associated with this device.

NULL Add-Unit Characteristics

The NULL device does not use the CF_ADDUNIT function so there are no add-unit characteristics associated with this device.

PPR Add-Unit Characteristics

PPR class devices use no unique add-unit characteristics. Only the INT, IOREG, and UNIT characteristics are used. All are required.

NET Add-Unit Characteristics

XOS supports two different network interface boards, each of which have slightly different add-unit characterists. Each is listed seperately below.

All NET class devices have names of the format NETn regardless of the type of the low level device driver for the interface. Thus the unit number for a NET unit must be unique across all NET unit types.

NET (EWDA) Devices

The EWDA type network device driver supports varieties of Western Digital/SMC Ethernet interface boards.

Table 14.9 summarizes the NET class EWDA add-unit characteristics.

Table 14.9 - Add-unit characteristics for NET class EWDA devices

Name	Fnc	Format	Size	Description
INT	Y	DECV	1	Interrupt number
IOREG	Y	HEXV	4	Base IO register number
MEM	Y	HEXV	4	Memory address
TYPE	Y	TEXT	8	Device type = EWDA
UNIT	Y	DECV	2	Unit number

These are all common add-unit characteristics which have been described previously. It should be noted that the MEM value must be in the range of 0xC0000 to 0xF0000 minus the amount of buffer memory on the card (8KB or 16KB). This value is NOT specified by the hardware set up, even though many WD/SMC compatable cards which use “soft” set up allow you to specify this address. The value associated with the hardware is not used. Thus the value of the MEM characteristic can specify any allowed address, independent of the card’s configuration. Of course, it must not conflict with other devices in the system.

NET (ENEA) Devices

The ENEA type network device driver supports the Novell NE1000/NE2000 and compatable Ethernet interface boards.

Table 14.10 summarizes the NET class ENEA add-unit characteristics.

Table 14.10 - Add-unit characteristics for NET class ENEA devices

Name	Fnc	Format	Size	Description
INT	Y	DECV	1	Interrupt number
IOREG	Y	HEXV	4	Base IO register number
TYPE	Y	TEXT	8	Device type = EWDA
UNIT	Y	DECV	2	Unit number

These are all common add-unit characteristics which have been described previously.

SNAP Add-Unit Characteristics

This section describes the add-unit characteristics for SNAP class devices. SNAP class devices provide access to the link control level for the Internet protocol stack using the SNAP or Bluebook link level protocols. These characteristics are summarized in Table 14.11.

Table 14.11 - Add-unit Characteristics for SNAP Class Devices

Name	Req	Format	Size	Description
NETDEV	Y	TEXT	16	Name of associated network device
UNIT	Y	DECV	4	Unit number

Most of the set up for the SNAP device is done using device characteristics after the unit is added. The only thing that must be specified when the device is added is the name of the network device (NET) that the SNAP device will be associated with. This name should always be of the format NETn, where n is a the unit number for the network device. Note that there is no requirement that the SNAP and NET devices have the same unit number, however, it is stongly recommended that they be the same to prevent confusion. Of course, this is not possible if more than one SNAP device is associated with the same NET device. This would be done to support multile link level protocols.

ARP Add-Unit Characteristics

This section describes the device characteristics for ARP class devices. This device provides access to the ARP protocol level in the Internet protocol stack. These characteristics are summarized in Table 14.12.

Table 14.12 - Add-unit Characteristics for ARP Class Devices

Name	Req	Format	Size	Description
SNAPDEV	Y	TEXT	16	Name of associated SNAP device
UNIT	Y	DECV	4	Unit number

Most of the set up for the ARP device is done using device characteristics after the unit is added. The only thing that must be specified when the device is added is the name of the SNAP device that the ARP device will be associated with. This name should always be of the format SNAPn, where n is a the unit number for the network device. Note that there is no requirement that the ARP and SNAP devices have the same unit number, however, it is stongly recommended that they be the same to prevent confusion. Of course, this is not possible if more than one ARP device is associated with the same SNAP device. While this is unlikely, it is possible.

IPS Add-Unit Characteristics

This section describes the add-unit characteristics for IPS class devices. The IPS devices are the lowest level network protocol devices which provide access to the IP level of the Internet Protocol Suite. These characteristics are summarized in Table 14.13.

Table 14.13 - Add-unit Characteristics for SNAP Class Devices

Name	Req	Format	Size	Description
SNAPDEV	Y	TEXT	16	Name of associated SNAP device
UNIT	Y	DECV	4	Unit number

Most of the set up for the IPS device is done using device characteristics after the unit is added. The only thing that must be specified when the device is added is the name of the SNAP device what the IPS device will be associated with. This name should always be of the format SNAPn, where n is a the unit number for the network device. Note that there is no requirement that the IPS and SNAP devices have the same unit number, however, it is stongly recommended that they be the same to prevent confusion. Of course, this is not possible if more than one IPS device is associated with the same SNAP device. This is unlikely, but possible.

UDP Add-Unit Characteristics

This section describes the add-unit characteristics for UDP class devices. The UDP devices provide access to the UDP protocol. These devices are commonly used to send and receive datagrams. These characteristics are summarized in Table 14.14.

Table 14.14 - Add-unit Characteristics for UDP Class Devices

Name	Req	Format	Size	Description
IPSDEV	Y	TEXT	16	Name of associated IPS device
UNIT	Y	DECV	4	Unit number

Most of the set up for the UDP device is done using device characteristics after the unit is added. The only thing that must be specified when the device is added is the name of the IPS device that the UDP device will be associated with. This name should always be of the format IPSn, where n is a the unit number for the network device. Note that there is no requirement that the UDP and IPS devices have the same unit number, however, it is stongly recommended that they be the same to prevent confusion. Of course, this is not possible if more than one UDP device is associated with the same IPS device. This is unlikely, but possible.

TCP Add-Unit Characteristics

This section describes the add-unit characteristics for TCP class devices. The TCP devices provide access to the TCP protocol. These devices are used by programs (such as servers) which need to directly access TCP virtual connections. These characteristics are summarized in Table 14.15.

Table 14.15 - Add-unit Characteristics for UDP Class Devices

Name	Req	Format	Size	Description
IPSDEV	Y	TEXT	16	Name of associated IPS device
UNIT	Y	DECV	4	Unit number

Most of the set up for the TCP device is done using device characteristics after the unit is added. The only thing that must be specified when the device is added is the name of the IPS device that the TCP device will be associated with. This name should always be of the format IPSn, where n is a the unit number for the network device. Note that there is no requirement that the TCP and IPS devices have the same unit number, however, it is stongly recommended that they be the same to prevent confusion. Of course, this is not possible if more than one TCP device is associated with the same IPS device. This is unlikely, but possible.

TLN Add-Unit Characteristics

This section describes the add-unit characteristics for TLN class devices. The TLN device implements a basic Telnet server as part of the XOS kernel. It actually consists of a pair of devices: the TLN device and the corresponding TRM class device which provides the functionality of a serial port TRM device. These characteristics are summarized in Table 14.16.

Table 14.16 - Add-unit Characteristics for TLN Class Devices

Name	Req	Format	Size	Description
PORT		DECV	4	TCP port for connections
TCPDEV	Y	TEXT	16	Name of associated TCP device
UNIT	Y	DECV	4	Unit number

Most of the set up for the TLN device is done using device characteristics after the unit is added. The following add-unit characteristics are used.

PORT - TCP port for connections

This characteristic specifies the TCP port used to listen for in-coming connections. If it is not specified, the standard Telnet port, 23 is used. The value must be numeric.

TCPDEV - Name of associated TCP device

This characteristic specifies the name of the TCP device that the TLN device will be associated with. This name should always be of the format TCPn, where n is a the unit number for the network device. Note that there is no requirement that the TLN and TCP devices have the same unit number, however, it is stongly recommended that they be the same to prevent confusion. Of course, this is not possible if more than one TLN device is associated with the same TCP device. This is unlikely, but possible.

RCP Add-Unit Characteristics

This section describes the add-unit characteristics for RCP class devices. The RCP devices provide access to the RCP protocol. These devices are used by programs (such as servers) which need to directly access RCP virtual connections. These characteristics are summarized in Table 14.17.

Table 14.17 - Add-unit Characteristics for RDP Class Devices

Name	Req	Format	Size	Description
UDPDEV	Y	TEXT	16	Name of associated UDP device
UNIT	Y	DECV	4	Unit number

Most of the set up for the RCP device is done using device characteristics after the unit is added. The only thing that must be specified when the device is added is the name of the UDP device that the RCP device will be associated with. This name should always be of the format UDPn, where n is a the unit number for the network device. Note that there is no requirement that the RCP and UDP devices have the same unit number, however, it is stongly recommended that they be the same to prevent confusion. Of course, this is not possible if more than one RCP device is associated with the same UDP device. This is unlikely, but possible.

XFP Add-Unit Characteristics

This section describes the add-unit characteristics for XFP class devices. The XFP devices implement a client for the XFP protocol. These devices are used to provide remote file access. These characteristics are summarized in Table 14.18.

Table 14.18 - Add-unit Characteristics for XFP Class Devices

Name	Req	Format	Size	Description
RCPDEV	Y	TEXT	16	Name of associated RCP device
UNIT	Y	DECV	4	Unit number

Most of the set up for the XFP device is done using device characteristics after the unit is added. The only thing that must be specified when the device is added is the name of the RCP device that the XFP device will be associated with. This name should always be of the format RCPn, where n is a the unit number for the network device. Note that there is no requirement that the XFP and RCP devices have the same unit number, however, it is stongly recommended that they be the same to prevent confusion. Of course, this is not possible if more than one XFP device is associated with the same RCP device. This is unlikely, but possible.

Chapter 15

svcIoQueue System Call

This chapter describes the svcIoQueue system call. This is the only XOS system call given its own chapter in this manual. This is by far the most complex system call. It implements virtually the entire XOS I/O interface. All major I/O functions eventually execute this system call. Most of the system calls that appear at first glance to be a rich set of other I/O system calls are in reality calls to user mode routines which set up arguments for the svcIoQueue call and then execute it. These routines are provided as a convenience when requesting simple operations, since setting up the argument block for the svcIoQueue call is more complex than pushing a couple of arguments onto the stack. Also, these user mode calls are more compatible with the DOS and Unix style APIs, making program conversion easier.

The svcIoQueue system call queues a request for an I/O operation.

CALLING SEQUENCE:

```
XOSSVC svcIoQueue(struct qab far *qab);
```

VALUE RETURNED:

The value returned is 0 unless an error occurred while queuing the request, in which case a negative error code is returned. Note that any errors which occur after the request is queued, which includes most errors associated with an I/O operation, are not indicated by the value returned by this call. Such errors cause a non-zero error code to be stored in the qab_error field in the QAB (see below). When the svcIoQueue call returns 0, several QAB fields are always filled to indicate the further progress of the I/O operation.

QAB Format

The svcIoQueue system call uses a single format argument (called a queued argument block or QAB) for all functions. The format of the QAB is summarized in Table 15.1.

Figure 15.1 - QAB Format

Name	Offset	Size	Description
qab_func	0	2	Function
qab_status	2	2	Returned status
qab_error	4	4	Returned error code
qab_amount	8	4	Returned amount
qab_handle	12	4	Device handle
qab_vector	16	1	Signal vector number
	17	3	Not used
qab_option	20	4	Option or command
qab_count	24	4	Count
qab_buffer1	28	8	Address of buffer 1
qab_buffer2	36	8	Address of buffer 2
qab_parmlist	44	8	Address of I/O parameter list

When the svcIoQueue system call is issued, all values set by the user, except for qab_vector, are copied to an internal system buffer; thus none of these values can be changed after the system call returns. These fields are explained in detail below.

qab_func - Function

This field specifies the I/O function to be queued by the svcIoQueue system call. Values for this field are summarized in Tables 15.3 and 15.4 and are described in detail in the following section. This field is 2 bytes long and must be filled in by the user before issuing the svcIoQueue call.

qab_status - Returned status

This field receives the status of the operation performed by the svcIoQueue call. After a blocking call, this field will always have the QSTS$DONE bit set. After a non-blocking call, this field can be polled to monitor the status of the I/O operation. The bits in this field are defined in Table 15.2. This field is 2 bytes long and is filled in by the system when the svcIoQueue call is issued.

Table 15.2 - Values for qab_status		
Bit	Name	Meaning
15	QSTS$DONE	Operation is complete
14	QSTS$ACTIVE	Operation is active
13	QSTS$WAIT	Need to wait (internal use only)
12	QSTS$REDO	Need to re-do (internal use only)
2	QSTSCANCEL	Operation canceled before started
1	QSTS$ABORT	Operation aborted while in progress
0	QSTS$TRUNC	Data truncated

The QSTS$ACTIVE bit is set when the operation is started and cleared when it is completed, at the same time the QSTS$DONE bit is set. The QSTS$WAIT and QSTS$REDO bits are used internally for scheduling and should be ignored by the user. The QSTS$CANCEL or QSTS$ABORT bit is set (one only) if the operation is terminated as a result of a svcIoCancel system call or as a result of process termination (although in this case it is generally not visible). The QSTS$TRUNC bit is set if not as much data was transferred as was requested due to an abnormal condition. This field will always be set for all svcIoQueue functions if the svcIoQueue returned a value of 0.

qab_error - Returned error code

This field receives an error code when the operation is complete. The error code is stored at the same time the QSTS$DONE bit is set. If the operation completed without any errors (other than truncation), a value of 0 is stored in this field. If there was an error, a negative error code is stored here. This field will always be set for all svcIoQueue functions if the svcIoQueue returned a value of 0.

qab_amount - Returned amount

This field receives the total amount transferred by the I/O operation. For data transfers, this will be the number of bytes transferred. For other functions (such as QFNC_OPEN, QFNC_DELETE, etc.), it is the number of items processed. This field is set to zero when the operation is started and may be updated incrementally as the operation progresses. In all cases, the final value will have been stored when the QSTS$DONE bit is set. This field will always be set for all svcIoQueue functions if the svcIoQueue returned a value of 0.

qab_handle - Device handle

This field contains the device handle for the I/O operation. This is the only QAB field which is stored by either the user or the system, depending on the function being performed. For the QFNC_OPEN function, the system stores the handle opened in this field (unless the O$FORCEH bit is set in qab_option). For all other functions, this field is either set by the user or is not used.

qab_vector - Signal vector number

This field specifies the signal vector number on which a signal is requested when the operation is complete. A value of 0 indicates that no signal is to be requested. This is the only QAB field that can be changed once the operation has started. This allows starting an operation which may finish quickly without a signal request and then setting a request once it is determined that the operation did not finish after some short period of time, thus often eliminating the overhead of an unnecessary signal.

qab_option - Option or command

This field specifies a value which modifies the function being performed. The format of this value depends on the function. It is described below with the descriptions of the individual functions. This field is set by the user and is 4 bytes long.

qab_count - Count

This field specifies the number of bytes or items to transfer or some other count associated with the function being performed This field is set by the user and is 4 bytes long.

qab_buffer1 - Address of buffer 1

This field specifies the address of the first buffer. For functions which require a device/file specification, this field specifies the address of that specification. For functions which transfer data, this field specifies the address of the data buffer. This field is set by the user and is 8 bytes long. The high order 2 bytes are not used.

qab_buffer2 - Address of buffer 2

This field specifies the address of the second buffer. For the QFNC_RENAME function, this field specifies the address of the new name. For the QFNC_OUTDATAGRAM function, this field specifies the address of the remote address string. For functions which use a characteristics list, this

field specifies the address of that list. This field is filled in by the user and is 8 bytes long. The high order 2 bytes are not used.

qab_parmlist - Address of I/O parameter list

This field specifies the address of a parameter list if one is used. This field is filled in by the user and is 8 bytes long. The high order 2 bytes are not used.

The qab_status, qab_error, and qab_amount fields are always filled in by the system; the user does not need to initialize them to any particular values. The qab_handle field is set by the system for some functions and must be set by the user for others. All other fields must be initialized by the user.

Summary of svcIoQueue Functions

Table 15.3 summarizes the values for the high order byte of the qab_func field. This value is bit encoded and is independent of the function code stored in the low order byte of this field.

Table 15.3 - High Order Byte of qab_func

Bit	Name	Meaning
15	QFNC$WAIT	Wait until complete
14	QFNC$DIO	Direct I/O
9	QFNC$CHILDTERM	Wait unit child process terminates
8	QFNC$SAMEPROC	Run program in same process

The meanings of these bits are described in detail below.

QFNC$WAIT (bit 15) = Wait until complete

When this bit is set, the svcIoQueue call blocks until the requested operation is complete. When this bit is not set, the call returns as soon as the request is queued.

QFNC$DIO (bit 14) = Direct I/O

When this bit is set, the requested operation is performed as a direct I/O operation (see Chapter 2 for a discussion of queued versus direct I/O) if the device supports direct I/O. If it does not support direct I/O, a simulated direct I/O operation is done using a queued request. This is done by disabling signals (if they were enabled) until the call returns and doing a normal queued I/O request. Note that this is only useful if the QFNC$WAIT bit is also set. The effect of this is to make the queued I/O operation non-interruptable. This feature is used by the DOS emulator to insure that DOS programs will not be confused by the XOS I/O queuing scheme.

When this bit is not set, the requested operation is performed as a queued I/O operation if the device supports queued I/O. If it does not support queued I/O, a direct I/O operation is performed.

QFNC$CHILDTERM (bit 9) = Wait until child process terminates

This bit is only used by the QFNC_RUN function. When it is set, the QFNC_RUN function does not complete until the child process created terminates. Also, qab_amount will receive the termination status of the child process instead of the PID of the process. If QFNC$WAIT is also

set, the call will not return until the child process terminates. If a termination signal is requested (qab_vector non-zero), the signal does not occur until the child process terminates. This bit is ignored if QFNC$SAMEPROC is also set.

QFNC$SAMEPROC (bit 8) = Run program in same process

This bit is only used by the QFNC_RUN function. When it is set, the program being run is run in the same process which issued the svcIoQueue call, replacing the current program. In this case, the svcIoQueue call will never return.

The low order byte of qab_func is value encoded and specifies the I/O function to be performed. Table 15.4 summarizes the values for this byte.

Table 15.4 - Low Order Byte of qab_func

Name	Value	Meaning
QFNC_OPEN	1	Open device/file
QFNC_DEVPARM	2	Device parameters
QFNC_DEVCHAR	4	Device characteristics
QFNC_DELETE	5	Delete file
QFNC_RENAME	6	Rename file
QFNC_PATH	8	Path functions
QFNC_CLASSFUNC	9	Class functions
QFNC_INBLOCK	12	Input block
QFNC_OUTBLOCK	14	Output block
QFNC_OUTSTRING	15	Output string
QFNC_SPECIAL	19	Special device functions
QFNC_LABEL	20	Read or write volume label
QFNC_COMMIT	21	Commit data to media
QFNC_CLOSE	22	Close file

The svcIoQueue functions are defined in detail in the following sections.

QFNC_OPEN - Open Device or File

QFNC_OPEN

QAB USAGE:

Field	Usage
qab_func	Function
qab_status	Returned status
qab_error	Returned error code
qab_amount	Returned number of items
qab_handle	Returned device handle
qab_vector	Signal vector number
qab_option	Open command bits
qab_count	Not used
qab_buffer1	Address of device/file specification
qab_buffer2	Not used
qab_parmlist	Address of I/O parameter list

VALUE RETURNED:

The value stored in qab_amount is 1 if the device was successfully opened or 0 if an error occurred.

Note that this is the only svcIoQueue function that modifies the qab_handle field. The handle assigned is returned in this field.

DESCRIPTION:

This function opens a device or file specified by name (pointed to by the qab_buffer1 field) and associates a device handle with it that is used to identify the device or file for other I/O operations.

The qab_option field contains the open command bits, which are summarized the Table 15.5.

Table 15.5 - Open Command Bits

Name	Bit	Description
O$REPEAT	31	Repeated operation
O$REQFILE	30	Require file structured device
O$NOMOUNT	29	Do not mount device if not mounted
O$ODF	28	Open directory as file
O$FAILEX	27	Fail if file exists

Table 15.5 - Open Command Bits

Name	Bit	Description
O$CREATE	26	Create new file if file does not exist
O$TRUNCA	25	Truncate existing file allocated length
O$TRUNCW	24	Truncate existing file written length
O$APPEND	23	Append to file
O$FHANDLE	22	Force handle
O$UNQNAME	21	Create unique name for file
O$NOWCL	20	Do not allow wild-card search lists
O$DFLTWILD	19	Default to wild-card extension
O$NORDAH	16	Do not read ahead
O$NODFWR	15	Do not defer writes
O$CONTIG	14	File allocation must be contiguous
O$CRIT	13	Do critical error processing
O$FAPPEND	12	Force append
O$SEQUENL	11	Sequential I/O only
O$PHYS	10	Physical I/O
O$RAW	9	Raw I/O
O$FNR	8	Fail if device is not ready
O$PARTIAL	7	Accept partial input
O$NOINH	6	Device cannot be passed to child
O$XWRITE	5	Exclusive write access
O$XREAD	4	Exclusive read access
O$DGOUT	3	Datagram output is allowed
O$DGIN	2	Datagram input is allowed
O$OUT	1	Output is allowed
O$IN	0	Input is allowed

The open command bits are described in detail below.

O$REPEAT (bit 31) = Repeated operation

This bit specifies that the operation should be repeated. This bit is only valid for the device parameter, delete, and rename functions. It is generally used in conjunction with a wildcard file specification to act upon multiple files with a single system call. If a file specification is returned (IOPAR_FILOPTN and IOPAR_FILSPEC specified), the operation will be repeated until the file specification buffer is full or until no more matching files are available. If no file specification is returned, the operation will be repeated until no more matching files are available. Any parameter values returned which reflect the length of a file will contain the sum of the values for all files found. If

the svcIoQueue call is used, the qab_count field will contain the number of files found. Bit 31 will be set if the operation stopped because the file specification buffer was full.

O$REQFILE (bit 30) = Require file structured device

If this bit is set, the operation will fail with an ER_NTFIL error if the device is not a file structured device.

O$NOMOUNT (bit 29) = Do not mount device if not mounted

If this bit is set and O$RAW or O$PHYS are also set, the automatic disk mount operation is suppressed. It has no effect if O$RAW or O$PHYS are not set or if the device is not a disk. If the device is a remote network disk, this bit affects the remote disk, not the network device.

O$ODF (bit 28) = Open directory as file

If this bit is set, the final directory in the file specification is opened. This directory can then be read as if it were a file using the normal I/O calls (this is not generally recommended, since the directory format varies between different file systems), or it may be efficiently searched by specifying the handle obtained here as the value of the IOPAR_DIRHNDL parameter. This is the recommended procedure for obtaining the contents of a directory since it is reasonably efficient and is independent of file system type. Note that directories can also be scanned using the IOPAR_DIROFFSET parameter without opening the directory, but this method is significantly less efficient since it requires the system to re-open the directory for each system call.

O$FAILEX (bit 27) = Fail if file exists

If this bit is set, the function will fail if the file specified exists. It has no effect if the file does not exist. This bit is commonly used to guarantee that an existing file is not overwritten when creating a new file.

O$CREATE (bit 26) = Create new file if file does not exist

If this bit is set, a new file is created if the file specified does not exist. It has no effect if the file exists.

O$TRUNCA (bit 25) = Truncate existing file allocated length

If this bit is set and the file specified exists, the allocated and written lengths of the file are reduced to zero . It has no effect if the file does not exist.

O$TRUNCW (bit 24) = Truncate existing file written length

If this bit is set and the file specified exists, the written length of the file is reduced to zero. The allocated length of the file is not changed, and no space is deallocated. If the file does not exist, this bit has no effect. This bit can be used effectively when a file is to be overwritten with new data. Keeping the existing allocation for the file significantly reduces the overhead involved in this operation. Any extra allocated space will normally be deallocated when the file is closed.

O$APPEND (bit 23) = Append to file

When this bit is set, the I/O position is set to the end of the file when the file is opened. It has no effect for operations which do not open a file.

O$FHANDLE (bit 22) = Force handle

When this bit is set, the value of qab_handle is used as the handle to use. The call will fail if this handle is not free. This bit should only be set when using the svcIoQueue system call, since this is the only call which provides access to the qab_handle field.

O$UNQNAME (bit 21) = Create unique name for file

When this bit is set, a unique name is created for the file. The name is created using the number of the process and a globally incremented value to insure complete uniqueness. The name created is concatenated to the file specification string given, which must provide at least 14 bytes for the name.

O$NOWCL (bit 20) = Do not allow wildcard search lists

When this bit is set, the characters { and } (which normally delimit wild-card search lists) are treated as normal file name characters, thereby disabling the wildcard search list feature. This option is needed to provide complete compatibility with the DOS file system calls since these are valid DOS file name characters, which are used by some DOS programs.

O$DFLTWILD (bit 19) = Default to wild-card extension

When this bit is set and no extension is specified (no period in name), a default wildcard extension (.*) is added to the file specification for file systems which use explicit extensions (DOS, XOS, and VMS). This bit is ignored for file systems which do not use explicit extensions (Unix) and for non-file structured devices.

The following bits are stored in the handle table and affect the operation of a device for as long as it is open.

O$NORDAH (bit 16) = Do not read ahead

When this bit is set, automatic read-ahead is disabled for accesses to the file or device. Normally the system will read ahead several blocks to attempt to keep the disk cache filled with data which will soon be accessed. It has no effect for non-disk devices or for disks which do not support read-ahead. When a disk is open in raw mode (O$RAW set), setting this bit disables multi-sector transfers.

O$NODFWR (bit 15) = Do not defer writes

When this bit is set, all data is written to the disk immediately. Normally, the system defers writing some data in an effort to optimize write accesses. This bit has no effect for non-disk devices or for disks which do not support deferred writes.

O$CONTIG (bit 14) = File allocation must be contiguous

When this bit is set, any attempt to allocate space to a file will fail if the space cannot be allocated contiguously with the rest of the space allocated to the file. The system normally allocates contiguous space whenever it can. Setting this bit requires such allocation.

O$CRIT (bit 13) = Do critical error processing

When this bit is set, certain device errors will cause a critical error signal to be requested. This allows compatibility with DOS style critical error handling.

O$FAPPEND (bit 12) = Force append

When this bit is set, the I/O position for the file cannot be set previous to the end of the file. This has the effect of forcing all output to the file to be appended to the end of the file. Data in the file cannot be read when this bit is set. This bit is ignored for non-file structured devices.

O$SEQUENL (bit 11) = Sequential I/O only

When this bit is set, the set I/O position operations are not allowed, forcing completely sequential access to the file. Setting this bit allows additional optimizations for network data transfers and can sometimes significantly increase network performance.

O$PHYS (bit 10) = Physical I/O

When this bit is set, the device is opened for physical I/O. The exact meaning of this option is device dependent. For terminals, it sets the terminal to image input and image output. For disks, it specifies that I/O is to be done directly between the device and the user buffer, bypassing the system's disk cache buffers. If a file name is specified for the disk, it is ignored. It also causes the device position to be interpreted as a raw disk address (sector, track, and cylinder) instead of as a disk block number.

O$RAW (bit 9) = Raw I/O

When this bit is set, it indicates that raw I/O transfers are to be done. The exact meaning of this is device dependent and is described in detail with the description of each device.

O$FNR (bit 8) = Fail if device is not ready

When this bit is set, any attempt to transfer data to or from a device which is not ready will fail with an ER_NTRDY error instead of waiting until the device is ready.

O$PARTIAL (bit 7) = Accept partial input

When this bit is set, certain I/O input operations for certain devices will complete when fewer than the requested number of bytes have been input. This applies mainly to image input operations from TRM class devices and for input operations from TCP class devices.

O$NOINH (bit 6) = Device should not be passed to child

When this bit is set, the device is not passed to a child process created with the QFNC_RUN function when the RUN$ALLPROC option is used. It will be passed to the child process if it is explicitly referenced in the IOPAR_DEVLIST I/O parameter, however.

O$XWRITE (bit 5) = Exclusive write access

When this bit is set, only one write access to the file is allowed. If the file is already open for write access and this bit is set, the open function will fail. Once the file has been opened with this bit set, any other attempt to open the file for write access (O$OUT set) will fail. This bit is ignored if the device is not file structured.

O$XREAD (bit 4) = Exclusive read access

When this bit is set, only one read access to the file is allowed. If the file is already open for read access and this bit is set, the attempt to

open the file will fail. Once the file has been opened with this bit set, any other attempt to open the file for read access (O$IN set) will fail. This bit is ignored if the device is not file structured.

O$DGOUT (bit 3) = Datagram output is allowed

When this bit is set, datagram output operations are allowed.

O$DGIN (bit 2) = Datagram input is allowed

When this bit is set, datagram input operations are allowed.

O$OUT (bit 1) = Output is allowed

When this bit is set, general output operations are allowed. This bit must also be set for any operation which changes the length of a file, even if no data is transferred.

O$IN (bit 0) = Input is allowed

When this bit is set, general input operations are allowed.

EXAMPLES:

QFNC_DEVPARM - Device Parameters

QFNC_DEVPARM

QAB USAGE:

Field	Usage
qab_func	Function
qab_status	Returned status
qab_error	Returned error code
qab_amount	Returned number of items
qab_handle	Not used
qab_vector	Signal vector number
qab_option	Open command bits
qab_count	Not used
qab_buffer1	Address of device/file specification
qab_buffer2	Not used
qab_parmlist	Address of I/O parameter list

VALUE RETURNED:

The value returned in the qab_amount field is the number of items processed.

DESCRIPTION:

This svcIoQueue function processes an I/O parameter list for a device or file specified by name without actually opening the device or file. In most cases, this functions is equivalent to an QFNC_OPEN function followed by a QFNC_CLOSE function. An exception is that the QFNC_DEVPARM function allows the O$REPEAT command bit while QFNC_OPEN function does not.

This svcIoQueue function is most often used to obtain information about a file, such as the creation date and time, or the written or allocated length. It is also used to obtain a list of files in a directory (with the O$REPEAT bit set). This is the standard method of obtaining the contents of a directory. XOS does not have a separate directory search function.

The contents of the qab_option field specifies the open command bits for the operation. See the description of the QFNC_OPEN function for a description of these bits. The qab_buffer1 field gives the address of the string which specifies the device or file. Wild card characters are allowed.

The qab_parmlist field specifies the address of the I/O parameter list. I/O parameter lists are described in detail in Chapter 11.

The value returned in the qab_amount field is the number of files processed. This will be 0 or 1 unless O$REPEAT was set in the qab_option field. This value is always available, even if an error occurs. In this case, it indicates the number of files successfully processed before the error occurred.

EXAMPLES:

QFNC_DEVCHAR - Device Characteristics Functions

QFNC_DEVCHAR

QAB USAGE:

Field	Usage
qab_func	Function (QFNC_CLASS)
qab_status	Returned status
qab_error	Returned error code
qab_amount	Returned number of items
qab_handle	Device handle
qab_vector	Signal vector number
qab_option	Sub-function and open command bits
qab_count	Not used
qab_buffer1	Not used
qab_buffer2	Address of class characteristic list
qab_parmlist	Address of I/O parameter list

VALUE RETURNED:

The value returned in the qab_amount field is the number of items processed.

DESCRIPTION:

This function provides a number of sub-functions relating to device characteristics.

All of these sub-functions use a device characteristics list, which is pointed to by the value of the qab_buffer2 field. This device characteristics list consists of a sequence of device characteristic items, followed by a byte containing 0. Each item consists of a 10 byte header, followed by a value between 0 and 32 bytes in length. A detailed description of the format of each item is given in Chapter 13. A discussion of how any why device characteristics are used is also found in Chapter 13.

The sub-functions for the QFNC_DEVCHAR function are summarized in Table 15.6.

Table 15.6 - Device Characteristics Functions		
Name	Value	Meaning
DCF_SIZE	1	Get size of complete characteristics list
DCF_ALL	3	Get complete characteristics list
DCF_TYPE	5	Get type and size of single characteristic
DCF_VALUES	7	Get or set characteristics values

These sub-functions are described in detail below.

DCF_SIZE = 1 - Get size of complete characteristics list

This functions returns, in the qab_amount field, the number of bytes required for a buffer to hold a complete device characteristics list for the device.

DCF_ALL = 3 - Get complete characteristics list

This function returns, in the buffer pointed to by the qab_buffer2 field, a skeleton device characteristics list containing all device characteristics defined for the device. Each entry has the PAR$SET and/or the PAR$GET bits set if they are valid for the characteristic. The length of each is sufficient to hold all valid values. The pointer part of each string value contains a null pointer. The buffer length part is filled in with the length buffer required for the longest possible string which can be returned by the characteristic. The value returned in the qab_amount field is the number of items in the characteristic list.

Before this characteristic list is used to obtain or set values, the user must scan through it and clear PAR$SET and/or PAR$GET bits as necessary. The user must also allocate string buffers and fill in the string pointers for characteristics with string values.

This function is intended to allow a program to determine all valid characteristics for a device and then to obtain or change their values. This function is used by the DEVCHAR command to list all characteristics for a device.

DCF_TYPE = 5 - Get format and size of single characteristic

This function returns the type and size of a single characteristic. The QAB field, qab_buffer2, must point to a characteristic containing a single entry. That entry must specify a characteristic name in the second through tenth bytes. The values of the first two bytes are ignored, as is anything following the characteristic name. There must be at least 12 bytes available following the characteristic in which data will be returned. The first 2 bytes will be filled in by this function to

indicate the valid functions, format, and size for the characteristic. If the characteristic has a string value, the buffer length field in the value will also be filled in. The value returned in the qab_amount field is always 1 unless an error occurred, in which case it is 0.

This function is intended to allow a program to display or change the value of any device characteristic without it having any advance knowledge of the format or size of the characteristic. It is used by the DEVCHAR command when individual characteristics are specified.

DCF_VALUES = 7 - Get or set characteristics values

This function returns or sets device characteristics values. Each device characteristic in the characteristic list pointed to by the qab_buffer2 field is processed. Values are obtained or set as specified by the PAR$GET and PAR$SET bits in the first byte of each characteristic. The value returned in the qab_amount field is the number of characteristics processed.

EXAMPLES:

QFNC_DELETE - Delete File

QFNC_DELETE

QAB USAGE:

Field	Usage
qab_func	Function
qab_status	Returned status
qab_error	Returned error code
qab_amount	Returned number of items
qab_handle	Not used
qab_vector	Signal vector number
qab_option	Open command bits
qab_count	Not used
qab_buffer1	Address of file specification
qab_buffer2	Not used
qab_parmlist	Address of I/O parameter list

VALUE RETURNED:

The value returned in the qab_amount field is the number of files deleted.

DESCRIPTION:

This function deletes the file(s) specified by the string whose addesss is given by the qab_buffer1 field. The I/O parameter list, whose address is given by the qab_parmlist field, is processed before any files are deleted. Wild-card characters are valid in the file name or extension. This function is only valid for file structured devices.

EXAMPLES:

QFNC_RENAME - Rename File

QFNC_RENAME

QAB USAGE:

Field	Usage
qab_func	Function
qab_status	Returned status
qab_error	Returned error code
qab_amount	Returned number of items
qab_handle	Not used
qab_vector	Signal vector number
qab_option	Open command bits
qab_count	Not used
qab_buffer1	Address of file specification
qab_buffer2	Address of new file name
qab_parmlist	Address of I/O parameter list

VALUE RETURNED:

The value returned in the qab_amount field is the number of files renamed.

DESCRIPTION:

This function renames the file(s) specified by the string whose address is given by the qab_buffer1 field to the new name(s) specified by the string whose address is given by the qab_buffer2 field. Wild card characters may be used in the file name and extension in the qab_buffer1 string and wild card substitution characters (see Chapter 2) may be used in the qab_buffer2 string. This function is only valid for file structured devices.

EXAMPLES:

QFNC_PATH - Path Functions

QFNC_PATH

QAB USAGE:

Field	Usage
qab_func	Function
qab_status	Returned status
qab_error	Returned error code
qab_amount	Returned 0 value
qab_handle	Not used
qab_vector	Signal vector number
qab_option	Open command bits
qab_count	Not used
qab_buffer1	Address of device/path specification
qab_buffer2	Not used
qab_parmlist	Address of I/O parameter list

VALUE RETURNED:

The value returned in qab_amount is always 0.

DESCRIPTION:

This function sets or gets the current directory path for a device. If the string pointed to by the qab_buffer1 field contains only a device name, the current directory path for that device is returned. The path is returned as the value of the IOPAR_FILSPEC I/O parameter which must be included in the parameter list pointed to by the qab_parmlist field if the value is to be obtained. The IOPAR_FILOPTN I/O parameter must also be present to specify which components of the path are to be returned. If a path is included in the qab_buffer1 string, the current directory path for the device is set to the path specified. All directories in the path must exist and be searchable by the process issuing the QFNC_PATH function or an error will be returned.

EXAMPLES:

QFNC_CLASSFUNC - Class Functions

QFNC_CLASSFUNC

QAB USAGE:

Field	Usage
qab_func	Function
qab_status	Returned status
qab_error	Returned error code
qab_amount	Returned number of items
qab_handle	Not used
qab_vector	Signal vector number
qab_option	Sub-function
qab_count	Not used
qab_buffer1	Address of class name
qab_buffer2	Address of class characteristics list
qab_parmlist	Not used

VALUE RETURNED:

The value returned in qab_amount is the number of items processed. The exact meaning of this value depends on the sub-function.

DESCRIPTION:

This function performs various functions involving device classes. This includes getting or setting the values of class characteristics, adding units to devices belonging to a class, and any additional class dependent functions defined for a specific class.

The sub-functions for the QFNC_DEVCHAR function are summarized in Table 15.7.

Table 15.7 - Device Characteristics Functions

Name	Value	Meaning
CF_SIZE	1	Get size of complete characteristics list
CF_ALL	3	Get complete characteristics list
CF_TYPE	5	Get type and size of single characteristic
CF_VALUES	7	Get or set characteristics values
CF_ADDUNIT	8	Add unit to device class
	9+	Class dependent functions

These sub-functions are described in detail below.

CF_SIZE = 1 - Get size of complete characteristics list

This functions returns, in the qab_amount field, the number of bytes required for a buffer to hold a complete class characteristics list for the class.

CF_ALL = 3 - Get complete characteristics list

This functions returns, in the buffer pointed to by the qab_buffer2 field, a skeleton class characteristics list containing all class characteristics defined for the class. Each entry has the PAR$SET and/or the PAR$GET bits set if they are valid for the characteristic. The length of each is sufficient to hold all valid values. The pointer part of each string value contains a null pointer. The buffer length part is filled in with the length buffer required for the longest possible string which can be returned by the characteristic. The value returned in the qab_amount field is the number of items in the characteristic list.

Before this characteristic list is used to obtain or set values, the user must scan through it and clear PAR$SET and/or PAR$GET bits as necessary. The user must also allocate string buffers and fill in the string pointers for characteristics with string values.

This function is intended to allow a program to determine all valid characteristics for a class, and then to obtain or change their values. This function is used by the CLSCHAR command to list all characteristics for a class.

CF_TYPE = 5 - Get format and size of single characteristic

This function returns the type and size of a single characteristic. The QAB field, qab_buffer2, must point to a characteristic containing a single entry. That entry must specify a characteristic name in the second through tenth bytes. The values of the first two bytes are ignored, as is anything following the characteristic name. There must be at least 12 bytes available following the characteristic in which data will be returned. The first 2 bytes will be filled in by this function to indicate the valid functions, format, and size for the characteristic. If the characteristic has a string value, the buffer length field in the value will also be filled in. The value returned in the qab_amount field is always 1 unless an error occurred, in which case it is 0.

This function is intended to allow a program to display or change the value of any class characteristic without it having any advance

knowledge of the format or size of the characteristic. It is used by the CLSCHAR command when individual characteristics are specified.

CF_VALUES = 7 - Get or set characteristics values

This function returns or sets device characteristics values. Each class characteristic in the characteristic list pointed to by the qab_buffer2 field is processed. Values are obtained or set as specified by the PAR$GET and PAR$SET bits in the first byte of each characteristic. The value returned in the qab_amount field is the number of characteristics processed.

CF_ADDUNIT = 8 - Add unit to device class

This sub-function adds a new device unit to the class. The qab_buffer2 field specifies the address of the add-unit characteristics list. This list has exactly the same format as a class characteristics list, but the names refer to add-unit characteristics. There is one common characteristic which must be included in the list. This is the UNIT characteristic, which has a 4 byte long numeric value. It specifies the unit number for the device unit being added to the class.

The value returned in the qab_amount field is 0 (if error) or 1 (if normal).

Additional class dependent functions can be defined for each device class. These are discussed in Chapter 19.

EXAMPLES:

QFNC_INBLOCK - Input Block

QFNC_INBLOCK

QAB USAGE:

Field	Usage
qab_func	Function
qab_status	Returned status
qab_error	Returned error code
qab_amount	Returned number of bytes input
qab_handle	Device handle
qab_vector	Signal vector number
qab_option	Not used
qab_count	Size of data buffer (bytes)
qab_buffer1	Address of data buffer
qab_buffer2	Not used
qab_parmlist	Address of I/O parameter list

VALUE RETURNED:

The value returned in the qab_amount is the actual number of bytes input.

DESCRIPTION:

This function reads the number of bytes specified in the qab_count field into the buffer pointed to by the qab_buffer1 field from the device specified by the device handle given in the qab_handle field.

If an I/O parameter list is specified in the qab_parmlist field, it is processed before the data transfer is done.

EXAMPLES:

QFNC_OUTBLOCK - Output Block

QFNC_OUTBLOCK

QAB USAGE:

Field	Usage
qab_func	Function
qab_status	Returned status
qab_error	Returned error code
qab_amount	Returned number of bytes output
qab_handle	Device handle
qab_vector	Signal vector number
qab_option	Not used
qab_count	Number of bytes to output
qab_buffer1	Address of data buffer
qab_buffer2	Not used
qab_parmlist	Address of I/O parameter list

VALUE RETURNED:

The value returned in the qab_amount field is the actual number of bytes output.

DESCRIPTION:

This system call outputs the number of bytes specified in the qab_count field from the buffer specified by the qab_buffer1 argument to the device specified by the device handle given in the qab_handle field.

EXAMPLES:

QFNC_OUTSTRING - Output String

QFNC_OUTSTRING

QAB USAGE:

Field	Usage
qab_func	Function
qab_status	Returned status
qab_error	Returned error code
qab_amount	Returned number of bytes output
qab_handle	Device handle
qab_vector	Signal vector number
qab_option	Not used
qab_count	Maximum number of bytes to output
qab_buffer1	Address of string
qab_buffer2	Not used
qab_parmlist	Address of I/O parameter list

VALUE RETURNED:

The value returned in the qab_amount field is the actual number of bytes output.

DESCRIPTION:

This function outputs a null terminated string from the buffer pointed to by the qab_buffer1 field to the device specified by the device handle given in the qab_handle field. The terminating NULL is not output. The field qab_count specifies the maximum number of bytes to output.

EXAMPLES:

QFNC_SPECIAL - Special Device Functions

QFNC_SPECIAL

QAB USAGE:

Field	Usage
qab_func	Function
qab_status	Returned status
qab_error	Returned error code
qab_amount	Returned value
qab_handle	Device handle
qab_vector	Signal vector number
qab_option	Sub-function
qab_count	Depends on sub-function
qab_buffer1	Depends on sub-function
qab_buffer2	Depends on sub-function
qab_parmlist	Address of I/O parameter list

VALUE RETURNED:

The value returned in qab_amount depends on the sub-function.

DESCRIPTION:

This function implements various device dependent sub-functions. The sub-function is specified in the low order byte of the qab_option field. The interpretation of the sub-function value and of all other QAB fields is device dependent. These sub-functions are discussed in Chapter 19.

An I/O parameter list must be specified and must contain at least the IOPAR_CLASS parameter with the PAR$SET bit set. The value specified must be the class of the device. If this parameter is not specified, an ER_PARMM error is returned. If the device class name is not correct, an ER_PARMV error is returned. The purpose of this is to verify that the device is of the expected class, since the same QAB field values can perform drastically different actions for different devices.

EXAMPLES:

QFNC_LABEL - Read or Write Volume Label

QFNC_LABEL

QAB USAGE:

Field	Usage
qab_func	Function
qab_status	Returned status
qab_error	Returned error code
qab_amount	Returned value
qab_handle	Device handle
qab_vector	Signal vector number
qab_option	Not used
qab_count	Buffer length
qab_buffer1	Address of data buffer
qab_buffer2	Not used
qab_parmlist	Not used

VALUE RETURNED:

Value returned is 0 in qab_amount is always 0

DESCRIPTION:

This function reads or writes the volume label on a mass storage device. The mass storage device must be open in raw mode. The label is written or read from the buffer specified by qab_buffer1 with length specified by qab_count.

EXAMPLES:

QFNC_COMMIT - Commit Data to Media

QFNC_COMMIT

QAB USAGE:

Field	Usage
qab_func	Function
qab_status	Returned status
qab_error	Returned error code
qab_amount	Returned 0
qab_handle	Device handle
qab_vector	Signal vector number
qab_option	Not used
qab_count	Not used
qab_buffer1	Not used
qab_buffer2	Not used
qab_parmlist	Not used

VALUE RETURNED:

The value returned in qab_amount is always 0.

DESCRIPTION:

This functions causes all data currently held in memory associated with a file or mass storate device to be written to the device. When issued for a file, it writes all file data and the file's directlry entry and attributes. When issued for a raw device, it writes all cached data for the device. This will not necessarily update directory entries and file attributes, but will write all file data. The function does not complete until all data has been written.

EXAMPLES:

QFNC_CLOSE - Close File

QFNC_CLOSE

QAB USAGE:

Field	Usage
qab_func	Function
qab_status	Returned status
qab_error	Returned error code
qab_amount	Returned value
qab_handle	Device handle
qab_vector	Signal vector number
qab_option	Close command bits
qab_count	Not used
qab_buffer1	Not used
qab_buffer2	Not used
qab_parmlist	Address of I/O parameter list

VALUE RETURNED:

The value returned in qab_amount is 0 or 1.

DESCRIPTION:

This function closes the device or file specified by the device handle in the qab_handle field. Any current I/O operations pending for the device or file are completed and the association between the handle and the device or file is removed. The handle is made idle and available for re-use. As long as a valid handle is specified, the handle will always be freed by this function, even is one or more errors occur while closing the device or file. The value returned is 1 if the handle was freed, or 0 if not (only possible if the handle was invalid).

EXAMPLES:

QFNC_IOCLOSE

Chapter 16

Input/Output System Calls

This section describes the XOS system calls which perform input/output operations and related functions, except for the svcIoQueue system call, which is described in Chapter 15.

XOS provides a single general purpose I/O call, svcIoQueue, which can be used to specify any I/O operation with any available option. It also provides a number of specialized calls for individual operations (such as svcIoOpen or svcIoInBlock), which are generally easier to use but do not provide access to all options. The most significant difference between svcIoQueue and the other I/O functions is that svcIoQueue allows for non-blockin or asynchronous I/O operations, which return control to the caller as soon as the operation is started. Completion of the I/O operation is then usually signaled with a signal.

Many of the system calls described in this chapter are actually implemented as small user mode functions which set up a QAB and then issue an svcIoQueue system call. Except when single-stepping, this is transparent to the user mode program.

svcIoCancel - Cancel I/O Request

svcIoCancel

CALLING SEQUENCE:

```
XOSSVC svcIoCancel(struct qab far *qab, long bits);
```

VALUE RETURNED:

The value returned is 0 if normal or a negative error code if an error occurred.

DESCRIPTION:

This system call cancels outstanding queued I/O requests. One or more outstanding I/O requests for the device specified by the qab_handle field of the QAB pointed to by the argument *qab* is canceled if it has not yet been started or aborted if it is currently active. The exact behavior is controlled by the value of the argument *bits* which is bit encoded and is summarized in Table 16.1.

Table 16.1 - svcIoCancel Request Bits

Name	Bit	Description
CAN$ALL	6	Cancel all requests for device
CAN$AFTER	5	Cancel this and all following requests
CAN$NOINT	4	Do not interrupt
CAN$OUTPUT	1	Cancel output requests
CAN$INPUT	0	Cancel input requests

The meanings of these bits are described in detail below.

CAN$ALL (bit 6) = Cancel all requests

When this bit is set, all outstanding requests (as specified by the CAN$INPUT and CAN$OUTPUT bits) for the device indicated by the qab_handle field of the QAB pointed to by the *qab* argument are canceled.

CAN$AFTER (bit 5) = Cancel this and all following requests

When this bit is set, the request associated with the QAB pointed to by the *qab* argument and all following requests for the device (as specified by the CAN$INPUT and CAN$OUTPUT bits) are canceled. This bit is ignored if CAN$ALL is also set.

CAN$NOINT (bit 4) = Do not interrupt

When this bit is set, the normal termination interrupt is suppressed for the canceled or aborted I/O requests.

CAN$OUTPUT (bit 1) = Cancel output requests

When this bit is set for a full duplex device, output requests are canceled. When it is set for a non-full duplex device, both input and output requests are canceled.

CAN$INPUT (bit 0) = Cancel input requests

When this bit is set for a full duplex device, input requests are canceled. It is ignored for a non-full duplex device.

Note that for this system call to do anything, at least one of the CAN$OUTPUT or CAN$INPUT bits must be set. For a non-full duplex device, the CAN$OUTPUT bit must always be set. If neither of the CAN$ALL or CAN$AFTER bits is set, only the request associated with the QAB specified by the *qab* argument is canceled.

EXAMPLES:

svcIoClose - Close Device

svcIoClose

CALLING SEQUENCE:

XOSSVC svcIoClose(long dev, long options);

VALUE RETURNED:

The value returned is zero unless an error occurred, in which case the value is the negative error code.

DESCRIPTION:

This system call closes a device descriptor previously opened. There are several options that can be specified when closing the file. These options are bit encoded in the *options* argument as follows:

Name	Bit	Description
C$RESET	31	Reset file/device to original state
C$DELETE	3	Delete file
C$TRUNC	1	Truncate file at current I/O position
C$NODEAL	0	Do not deallocate unwritten space in file

These bits are described in detail below.

C$RESET (bit 31) = Reset file/device to original state

C$DELETE (bit 3) = Delete file

C$TRUNC (bit 1) = Truncate file at current I/O position

C$NODEAL (bit 0) = Do not deallocate unwritten space in file

This system call is implemented as a user mode routine which sets up a queued argument block (qab) and then issues an svcIoQueue system call. More complete control of this operation is available by using the QFNC_CLOSE function of the svcIoQueue system call directly.

EXAMPLES:

svcIoControl - I/O Request Control

svcIoControl

CALLING SEQUENCE:

```
XOSSVC svcIoControl(struct qab far *qab, func, data);
```

VALUE RETURNED:

The value returned is zero if normal or a negative error code if an error occurred.

DESCRIPTION:

This system call controls the handling of I/O requests which are currently being executed. It will fail with an ER_NACT error if the request is still waiting to start or has already completed. Values specified in an I/O parameter list are not used until a request is started (not when the request is queued), so any value (such as a timeout value), changed in the I/O parameter list after a request is queued but before it is started, will be effective. It will not be effective if it is changed after the request is started. The correct procedure for changing a parameter value is to first change the value in the parameter list (in case the request is not yet started), and then to issue a svcIoControl call to change the value if the request has been started. An ER_NACT error should be ignored. Parameter list values that do not have a matching svcIoControl function should not be changed after the request is queued. The argument *func* specifies the function to perform as summarized in Table 16.2.

Table 16.2 - svcIoControl Request Bits

Name	Value	Description
QIOC_SITO	1	Set input time-out
QIOC_SOTO	2	Set output time-out

Following is a detailed description of these functions.

QIOC_SITO = 1 - Set input time-out

This function changes the time-out value for the current input operation for full duplex devices. If the device is not full duplex, this function is identical to the QIOC_SOTO function. The input time-out value is set to the value specified in the *data* argument.

QIOC_SOTO = 2 - Set output time-out

This function changes the time-out value for the current output operation for full duplex devices or the current input or output operation for non-full duplex devices. The time-out value is set to the value specified in the *data* argument.

EXAMPLES:

svcIoDefLog - Define Logical Name

svcIoDefLog

CALLING SEQUENCE:

```
XOSSVC svcIoDefLog(long proc, long type, char far *name, char far *defin);
```

VALUE RETURNED:

The value returned is 0 if normal or a negative error code if an error occurred.

DESCRIPTION:

This system call defines a logical device name.

The logical name is defined for the process specified by the argument *proc*. If bits 14 and 15 of this value are both 1, a system level logical name is defined. If the entire value is 0, the logical name is defined in the session level process for the process issuing the system call. Otherwise, the argument specifies a PID. The process specified must be a session level process.

The argument *type* specifies the type of logical name being defined

Name	Bit	Description
TLN$SUBST	30	Substituted logical name
TLN$ROOTED	29	Rooted logical name

These values are described in detail below. All bits not specified are reserved and must be 0.

TLN$SUBST (bit 30) = Define substituted logical name

When this bit is 1, the logical name is defined as a substituted logical name. See Chapter 2 for a discussion of substituted logical names. When it is 0, the name is defined as a non-substituted logical name.

LOG$ROOTED (bit 29) = Define rooted logical name

When this bit is 1, the logical name is defined as a rooted logical name. See Chapter 2 for a discussion of rooted logical names. When it is 0, the name is defined as a non-rooted logical name.

The argument *name* specifies the logical name being defined. This must be a 1 to 8 character name containing only letters, numbers, or the characters

underscore, minus, or dollar sign. It must be followed by a colon and the string must be terminated with a NULL.

The argument defin specifies the definition for the logical name. This must begin with a valid device name (which may be null), optionally followed by a path specification. A path specification, if given, must end with a back-slash character. A name which begins with an underscore character is a physical only name. The name (with the underscore removed) is taken as the name of a physical device, i.e., no further logical name searches are done. Normally, a logical name search is done on each name in a definition chain until no match can be found. Specifying names which are known to be real device names as physical names reduces overhead somewhat by eliminating unnecessary logical name searches. This feature can also be used to prevent a user from changing an important logical name by redefining its target name.

A non-substituted logical name may be defined as a search list logical name. This is done by providing a list of two or more definitions separated by commas in the string specified by the defin argument. When this logical name is used, the system will attempt to access the file specified using each element of the list in turn until the access succeeds.

A discussion of the use of logical names can be found in Chapter 2.

EXAMPLES:

svcIoDelete - Delete File

svcIoDelete

CALLING SEQUENCE:

```
XOSSVC svcIoDelete(long cmdbits, void far *name, struct parmlist far *parmlist);
```

VALUE RETURNED:

The value returned is the number of files deleted if normal or a negative error code if an error occurred.

DESCRIPTION:

The file specified in the argument *name* is deleted by this call. The filename may contain wildcards, although wildcards are not allowed in the device or directory specifications.

The argument *cmdbits* specifies open command bits used when accessing the file to be deleted. These bits are defined in the description of the svcIoOpen system call.

The argument *parmlsit* is a pointer to an optional I/O parameter list.

Note that if an error occurs on a repeated delete, the number of files deleted before the error occurred is not available with this call. This number can be obtained by using the QFNC_DELETE function with the svcIoQueue call. This system call is implemented as a user mode routine which sets up a queued argument block (qab) and then issues an svcIoQueue system call. More complete control of this operation is available by using the QFNC_DELETE function of the svcIoQueue system call directly.

EXAMPLES:

svcIoDevParm - Get or Set Device Parameters

svcIoDevParm

CALLING SEQUENCE:

```
XOSSVC svcIoDevParm(long cmd, char far *name, struct parmlist far *parmlist);
```

VALUE RETURNED:

The value returned is the number of items processed (a positive value) if normal or a negative error code if an error occurred.

DESCRIPTION:

This system call processes an I/O parameter list for a device or file specified by name without actually opening the device or file. In most cases, this call is equivalent to an svcIoOpen system call followed by an svcIoClose system call. An exception is that the svcIoDevParm call allows the O$REPEAT command bit while the svcIoOpen call does not.

This call is most often used to obtain information about a file, such as the creation date and time, or the written or allocated length. It is also used to obtain a list of files in a directory (with the O$REPEAT bit set). This is the standard method of obtaining the contents of a directory. XOS does not have a separate directory search function.

The *cmd* argument specifies the open command bits for the operation. See the description of the svcIoQueue function QFNC_OPEN in Chapter 15 for a description of these bits. The *name* argument specifies the device or file. wildcard characters are allowed. The *parmlist* argument specifies the address of the I/O parameter list. I/O parameter lists are described in detail in Chapter 11.

The value returned is the number of files processed. This will be 1 unless O$REPEAT was set in the *cmd* argument. If an error occurs on the n-th file processed with O$REPEAT set, the value of n is not available, except by counting the number of names returned for the IOPAR_FILSPEC parameter if it is used. This system call is implemented as a user mode routine which sets up a queued argument block (qab) and then issues an svcIoQueue system call. More complete control of this operation is available by using the QFNC_DEVPARM function of the svcIoQueue system call directly.

EXAMPLES:

svcIoDstName - Build Destination Name

svcIoDstName

CALLING SEQUENCE:

```
XOSSVC svcIoDstName(char far *gvnname, char far *rtnname, char far *dstname,
        long dstlength);
```

VALUE RETURNED:

The value returned is the length of the name built (positive) if normal or a negative error code if an error occurred.

DESCRIPTION:

This system call builds a destination file name for a copy or rename operation when given a destination file specification (which includes wildcard characters) and the file specification string returned by the IOPAR_FILSPEC parameter when the source file was opened. While this function is a utility routine which could be duplicated by user written code, its use to generate destination names is strongly recommended. XOS wildcard handling for source/destination file operations is very complex and the use of this system call insures uniform behavior for such operations.

The argument *gvnname* points to a string containing the file specification given for the destination file. It may contain wildcard characters or wild-card replacement fields (see the description of wildcard handling in Chapter 2). The argument *rtnname* points to a string which was returned by the IOPAR_FILSPEC I/O parameter for the open call which accessed the source file. This string must include at least the file name and the file search mask (FO$NAME and FO$MASK set in the value of the IOPAR_FILOPTN I/O parameter). Other returned items may also be present. The destination name is returned in the buffer pointed to by the argument *dstname*. The length of this buffer is specified by the argument *dstlength*. The length should be at least the length of the string pointed to by *gvnname* plus 14 (for DOS file systems) or 34 (for XOS file systems). To allow for possible new file systems which will be supported in the future, add at least 70 to the length of the given file specification. In any case, a length of 512 will always be sufficient, since this is the internal XOS file specification maximum length.

EXAMPLES:

svcIoDupHandle - Duplicate Device Handle

svcIoDupHandle

CALLING SEQUENCE:

XOSSVC svcIoDupHandle(long oldhandle, long newhandle, long newcmd);

VALUE RETURNED:

The value returned is the new handle (positive) if normal or a negative error code if an error occurred.

DESCRIPTION:

This system call duplicates an open device handle. The new handle can be either allocated or specified explicitly.

The argument *oldhandle* specifies a currently open device handle which is to be duplicated. The argument *newhandle* specifies the new handle. A value of -1 means to allocate a handle. The argument *newcmd* specifies new open command bits for the duplicated handle. A value of 0 means to use the open command bits from the old handle without change.

If a new handle is specified and that handle is currently open, the next available handle (in increasing numeric value) will be used.

EXAMPLES:

svcIoFindLog - Find Logical Name

svcIoFndLog

CALLING SEQUENCE:

XOSSVC svcIoFindLog(long proc, char far *string, char far *name, char far *defin, long length, long far *skip);

VALUE RETURNED:

The value returned is the length of the defined name returned ored with the type bits from the definition (positive value) if normal or a negative error code if an error occurred. The type bits are:

Name	Bit	Description
TLN$SUBST	30	Substituted logical name
TLN$ROOTED	29	Rooted logical name

These bits are described in detail below.

TLN$SUBST (bit 30) = Define substituted logical name

When this bit is 1, the logical name is a substituted logical name. When it is 0, the name is a non-substituted logical name. See Chapter 2 for a discussion of substituted logical names.

LOG$ROOTED (bit 29) = Define rooted logical name

When this bit is 1, the logical name is a rooted logical name. When it is 0, the name is a non-rooted logical name. See Chapter 2 for a discussion of rooted logical names.

DESCRIPTION:

This system call searches for a specified logical name. wildcard characters can be used.

The logical name is searched for in the process specified by the argument proc. If bits 14 and 15 of this value are both 1, the system level logical names are searched. If the entire value is 0, the logical name is searched for in the session level process for the process issuing the system call. Otherwise, the argument specifies a process ID. The process specified must be a session level process.

The argument string points to the name to search for, which may contain wildcard characters. The argument *name* points to a buffer which receives

the logical name actually matched. This buffer must be at least 13 bytes long. The argument defin points to a buffer which receives the definition of the logical name. The length of this buffer is specified by the argument length. If the definition of the logical name is too long to fit in the buffer, the final two characters stored in the buffer are s RUBOUT (0xFF) followed by a null (0x00). Otherwise, the definition is stored followed by a null (0x00). It is recommended that a length of 513 bytes be used, since this is the maximum length of an XOS logical name, plus one for a final null character.

The argument skip points to a 32-bit longword which specifies the number of definitions to skip before starting to check for a match. This value is updated so that a succeeding call will return the next logical name in the table, assuming no definitions are added or removed. A value of 0 returns the first matching definition.

EXAMPLES:

svcIoInBlock - Input Block

svcIoInBlock

CALLING SEQUENCE:

```
XOSSVC svcIoInBlock(long handle, char far *buffer, long count);
```

VALUE RETURNED:

The value returned is the positive number of bytes read if the operation was successful or a negative error code if an error occurred.

DESCRIPTION:

This call reads the number of bytes specified in the *count* argument into the buffer pointed to by the *buffer* argument from the device specified by the device handle given by the *handle* argument.

If an I/O parameter list is specified by the *parmlist* argument, it is processed before the data transfer is done.

This system call is implemented as a user mode routine which sets up a queued argument block (qab) and then issues an svcIoQueue system call. More complete control of this operation is available by using the QFNC_INBLOCK function of the svcIoQueue system call directly.

EXAMPLES:

svcIoInBlockP - Input Block/Parameter List

svcIoInBlockP

CALLING SEQUENCE:

```
XOSSVC svcIoInBlockP(long handle, char far *buffer, long size, struct parmlist far
        *parmlist);
```

VALUE RETURNED:

The value returned is the positive number of bytes read if the operation was successful or a negative error code if an error occurred.

DESCRIPTION:

This system call is the same as the svcIoInBlock system call except that it also specifies an I/O parameter list. The parameter list is processed before the data transfer is done.

This system call is implemented as a user mode routine which sets up a queued argument block (qab) and then issues an svcIoQueue system call. More complete control of this operation is available by using the QFNC_INBLOCK function of the svcIoQueue system call directly.

EXAMPLES:

svcIoInSingle - Input Byte

svcIoInSingle

CALLING SEQUENCE:

```
XOSSVC svcIoInSingle(long handle);
```

VALUE RETURNED:

The value returned is the data byte value (zero extended to 32-bits) unless an error occurred, in which case a negative error code is returned.

DESCRIPTION:

This system call reads a single byte from the device specified by the device handle given in the handle argument.

This system call is implemented as a user mode routine which sets up a queued argument block (qab) and then issues an svcIoQueue system call. More complete control of this operation is available by using the QFNC_INBLOCK function of the svcIoQueue system call directly.

EXAMPLES:

svcIoInSingleP - Input Byte/Parameter List

svcIoInSingleP

CALLING SEQUENCE:

XOSSVC svcIoInSingleP(long dev, struct parmlist far *parmlist);

VALUE RETURNED:

The value returned is the data byte value (zero extended to 32 bits) unless an error occurred, in which case a negative error code is returned.

DESCRIPTION:

This system call is the same as the svcIoInSingle system call except that it allows the specification of an I/O parameter list. The I/O parameter list is processed before the data transfer is done.

This system call is implemented as a user mode routine Which sets up a queued argument block (qab) and then issues an svcIoQueue system call. More complete control of this operation is available by using the QFNC_INBLOCK function of the svcIoQueue system call directly.

EXAMPLES:

svcIoOpen - Open Device or file

svcIoOpen

CALLING SEQUENCE:

```
XOSSVC svcIoOpen(long cmdbits, char far *name, parmlist far *parmlist);
```

VALUE RETURNED:

The value returned is the positive device handle allocated unless an error occurred, in which case a negative error code is returned.

DESCRIPTION:

The device or file specified by the argument *name* is set up for use for input or output or both and a device handle is associated with the device or file. The operations to be allowed and some options for the open function itself are specified by the value of the *cmdbits* argument. These values are described in the description of the QFNC_OPEN svcIoQueue function in Chapter 15.

This system call is implemented as a user mode routine which sets up a queued argument block (qab) and then issues an svcIoQueue system call. More complete control of this operation is available by using the QFNC_OPEN function of the svcIoQueue system call directly.

EXAMPLES:

svcIoOutBlock - Output Block

svcIoOutBlock

CALLING SEQUENCE:

XOSSVC svcIoOutBlock(long handle, void far *buffer, long count);

VALUE RETURNED:

The value returned is the positive number of bytes actually output unless an error occurred, in which case a negative error code is returned.

DESCRIPTION:

This system call outputs the number of bytes specified in the *count* argument from the buffer specified by the buffer argument to the device specified by the device handle given by the handle argument.

This system call is implemented as a user mode routine which sets up a queued argument block (QAB) and then issues an svcIoQueue system call. More complete control of this operation is available by using the QFNC_OUTBLOCK function of the svcIoQueue system call directly.

EXAMPLES:

svcIoOutBlockP - Output Block/Parameter List

svcIoOutBlockP

CALLING SEQUENCE:

XOSSVC svcIoOutBlockP(long dev, char far *data, long size, struct parmlist far * parmlist)

VALUE RETURNED:

The value returned is the positive number of bytes actually output unless an error occurred, in which case a negative error code is returned.

DESCRIPTION:

This system call is the same as the svcIoOutBlock system call except that it allows an I/O parameter list to be specified. The I/O parameter list is processed before the I/O transfer is done.

This system call is implemented as a user mode routine which sets up a queued argument block (qab) and then issues an svcIoQueue system call. More complete control of this operation is available by using the QFNC_OUTBLOCK function of the svcIoQueue system call directly.

EXAMPLES:

svcIoOutSingle - Output Byte

svcIoOutSingle

CALLING SEQUENCE:

XOSSVC svcIoOutSingle(long handle, long byte);

VALUE RETURNED:

The value returned is the amount output (0 or 1) unless an error occurred, in which case a negative error code is returned

DESCRIPTION:

The single byte contained in the *byte* argument (the high order 24-bits are ignored) is output to the device specified by the device handle given by the *handle* argument.

This system call is implemented as a user mode routine which sets up a queued argument block (QAB) and then issues an svcIoQueue system call. More complete control of this operation is available by using the QFNC_OUTBLOCK function of the svcIoQueue system call directly.

EXAMPLES:

svcIoOutSingleP - Output Byte/Parameter List

svcIoOutSingleP

CALLING SEQUENCE:

```
XOSSVC svcIoOutSingleP(long dev, long byte, struct parmlist far * parmlist);
```

VALUE RETURNED:

The value returned is the amount output (0 or 1) unless an error occurred, in which case a negative error code is returned

DESCRIPTION:

This system call is the same as the svcIoOutSingle system call except that it allows an I/O parameter list to be specified. The I/O parameter list is processed before the I/O transfer is done.

This system call is implemented as a user mode routine which sets up a queued argument block (QAB) and then issues an svcIoQueue system call. More complete control of this operation is available by using the QFNC_OUTBLOCK function of the svcIoQueue system call directly.

EXAMPLES:

svcIoOutString - Output String

svcIoOutString

CALLING SEQUENCE:

```
XOSSVC svcIoOutString(long handle, void far *buffer, long count);
```

VALUE RETURNED:

The value returned is the positive number of bytes actually output, unless an error occurred, in which case a negative error code is returned.

DESCRIPTION:

This system call outputs a null terminated string from the buffer pointed to by the *buffer* argument to the device specified by the device handle given by the *handle* argument. The terminating NULL is not output. The argument *count* specifies the maximum number of bytes to output. A value of 0 indicates that there is no maximum number specified.

This system call is implemented as a user mode routine which sets up a queued argument block (qab) and then issues an svcIoQueue system call. More complete control of this operation is available by using the QFNC_OUTSTRING function of the svcIoQueue system call directly.

EXAMPLES:

svcIoOutStringP - Output String/Parameter List

svcIoOutStringP

CALLING SEQUENCE:

```
XOSSVC svcIoOutStringP(long dev, void far *data, long size, struct parmlist far *
    parmlist);
```

VALUE RETURNED:

The value returned is the positive number of bytes actually output, unless an error occurred, in which case a negative error code is returned.

DESCRIPTION:

This system call is the same as the svcIoOutString system call except that it allows an I/O parameter list to be specified. The I/O parameter list is processed before the I/O transfer is done.

This system call is implemented as a user mode routine which sets up a queued argument block (QAB) and then issues an svcIoQueue system call. More complete control of this operation is available by using the QFNC_OUTSTRING function of the svcIoQueue system call directly.

EXAMPLES:

svcIoPath - Set Default Path

svcIoPath

CALLING SEQUENCE:

```
XOSSVC svcIoPath(long cmdbits, char far *path, struct parmlist far *parmlist);
```

VALUE RETURNED:

The value returned is 0 if normal or a negative error code if an error occurred.

DESCRIPTION:

This system call sets or returns the current directory path for a device. If the string pointed to by the argument *path* contains only a device name, the current directory path for that device is returned. The path is returned as the value of the IOPAR_FILSPEC I/O parameter which must be included in the parameter list pointed to by the argument *parmlist* if the value is to be obtained. The IOPAR_FILOPTN I/O parameter must also be present to specify which components of the path are to be returned. If a path is included in the *path* string, the current directory path for the device is set to the path specified. All directories in the path must exist and be searchable by the process issuing the svcIoPath call or an error will be returned.

This system call is implemented as a user mode routine which sets up a queued argument block (QAB) and then issues an svcIoQueue system call. More complete control of this operation is available by using the QFNC_PATH function of the svcIoQueue system call directly.

EXAMPLES:

svcIoPorts - Control Access to I/O Ports

svcIoPorts

CALLING SEQUENCE:

```
XOSSVC svcIoPorts(long func, long base, long num);
```

VALUE RETURNED:

The value returned is 0 if normal or a negative error code if an error occurred.

DESCRIPTION:

This system call controls direct access to I/O ports by a user program. The process issuing this call must have the PORTIO privilege. The first argument, *func*, specifies the function to perform. A value of 1 indicates to allow access to the specified I/O ports and a value of 2 indicates to disallow access. All other values are illegal. The *base* argument specifies the first port affected. The *num* argument specifies the number of ports affected. Only ports between 0 and 0x3FF (inclusive) may be allowed or disallowed.

This system call should be used with care, since no checking is done to see if the port(s) requested are already in use. Misuse of this call and the resulting ability to directly access I/O ports can easily cause a system failure. It is intended for use by programs which must have direct, efficient access to a device's I/O ports.

If ports which would normally be virtualized by the system are specified as allowed, the virualization is by-passed and direct access is allowed.

EXAMPLES:

svcIoRename - Rename File

svcIoRename

CALLING SEQUENCE:

```
XOSSVC svcIoRename(long cmdbits, char far *oldname, char far *newname,
        struct parmlist far *parmlist);
```

VALUE RETURNED:

The value returned is the number of files renamed unless an error occurred, in which case the negative error code is returned. Note that if an error occurs on a repeated rename, the number of files renamed before the error occurred is not available with this call. This number can be obtained by using the QFNC_RENAME function with the svcIoQueue call.

DESCRIPTION:

The name and extension of the file specified in the first parameter is changed to the name and extension specified by the second parameter. The third parameter is a pointer to a device parameter block which will receive the file information for the file(s) which were renamed.

EXAMPLES:

svcIoRun - Run or Load Program

svcIoRun

CALLING SEQUENCE:

```
XOSSVC svcIoRun(QAB *qab);
```

VALUE RETURNED:

The value returned is 0 if the function was started succesfully or a negative error code if an error occured during the initial part of execution.

The value returned in qab_error must also be checked (see below).

DESCRIPTION:

This system call executes a program or batch command file. The program or command file can be executed in the same process or in a child process, depending on the options specified. This is the normal XOS mechanism to load and execute a program or batch command file.

A program to be executed can be either an XOS or a DOS program. If the program issueing this system call is an XOS program, the new program or command file either completely replaces the calling program or is executed as a child process. If the program issueing this system call is a DOS program and new program is also a DOS program, it can be run in the same process as the calling program without terminating the calling program or it can be executed as a child process. If the new program is an XOS program or a command file, it either replaces the calling program or is executed as a child process. A special case allows any program to be loaded into the same process but not executed. This is normally only used by debuggers to load the program to be debugged.

The single argument, qab, provides the address of a “queued IO argument block” (QAB). This is the same argument block that is used with the svcIoQueue system call. The usage of the QAB is described below.

Field	Usage
qab_func	Function
qab_status	Returned status
qab_error	Returned error code
qab_amount	Returned PID and image type
qab_handle	Not used
qab_vector	Signal vector number
qab_option	Run option bits
qab_count	Not used
qab_buffer1	Address of device/file specification
qab_buffer2	Not used
qab_parmlist	Address of I/O parameter list

In the qab_func value, the high order byte is not currently used. A future version of XOS may use the QFNC_WAIT bit in this byte. The QFNC_WAIT bit in qab_func is currently ignored and the system call always waits for completion. Note that in this case completion means that the child process is ready to start executing. Most the data transfer needed to load the new program is done in the child process, and occures after the call is "complete". A future version of XOS may remove this restriction when a program is run in a child process and allow this call to be completely non-blocking in this case. When the calling program is to be replaced by the loaded program, it does not make much sense to not wait. To insure future compatability, the QFNC_WAIT bit should always be set.

The low order of qab_func contains the "run function" as follows:

Name	Value	Description
RFNC_RUN	1	Load and execute program
RFNC_LOAD	2	Load program only

The values returned in the qab_status and qab_error fields are the same as for the svcIoQueue system call.

The value returned in qab_amount is the PID merged with the image type of the child process if a child process was created as follows.

Bits	Description		
31-16	Process sequence number (high order half of the PID)		
15-12	Image type		
	Name	Value	Description
	IT_XOS32	1	XOS 32-bit protected mode process
	IT_XOS16	2	XOS 16-bit protected mode process
	IT_XOSV86	3	XOS 16-bit "real" mode process
	IT_DOSEXE	8	DOS process loaded from .EXE file
	IT_DOSCOM	9	DOS process loaded from .COM file
	IT_BATCH	15	Batch process
11-0	Process index (low order half of the PID)		

The qab_vector field can be set to specify a vector for a signal upon completion. Since the call is always blocking this is not very useful, but it does work if desired. The signal will always be requested before the call returns.

The qab_option field contains the run options bits which are defined below:

Name	Value	Description
SAMEPROC	0x80000000	Use same process
CHILDTERM	0x40000000	Function is not complete until child process terminates
SESSION	0x04000000	Create new session for child process
DEBUG	0x02000000	Do debug load of program into same process
CPYENV	0x00800000	Copy current enviroment to new process
ACSENV	0x00400000	Allow new process to access this process's enviroment
CHGENV	0x00200000	Allow new process to change this process's enviroment
ALLDEV	0x00080000	Pass all devices to new process (device list is
CPYPTH	0x00040000	Copy default paths to new process
CHGPTH	0x00020000	Allow new process to change this process's default
DOSEXEC	0x00000001	DOS EXEC function (XOS internal use only)

The qab_buffer1 field points to the string which specifies the file to load.

The qab_parmlist field points to an IO parameter list which may contain both normal IO parameters (which refer to the file being loaded and can do such things as return its full specification) and the special RUN IO pa-

rameters, which are only used with this system call and specify or return various attributes of the new process being created.

EXAMPLES:

svcIoSetPos - Set I/O Position

svcIoSetPos

CALLING SEQUENCE:

```
XOSSVC svcIoSetPos(long dev, long pos, long mode);
```

VALUE RETURNED:

The value returned is the new absolute position in the file, which is always positive, unless an error occurred, in which case the value will be the negative error code.

DESCRIPTION:

This function sets the current I/O position for a currently open file. The first parameter is the device descriptor for a currently open file. The second parameter is the desired offset within the file. The third parameter specifies how this offset will be applied, and must be one of the following values:

Value	Description
0	Absolute
1	Relative to current position
2	Relative to the end of file
3	No change (return current position only)

Any other value results in an ER_FUNC error. The offset in the file returned is always relative to the beginning of the file, regardless of the mode value specified.

EXAMPLES:

svcIoWait - Wait Until I/O is Complete

svcIoWait

CALLING SEQUENCE:

```
XOSSVC svcIoWait(struct qab far *qab);
```

VALUE RETURNED:

The value returned is zero if normal or a negative error code if an error occurred.

DESCRIPTION:

This system call waits until the I/O operation associated with the QAB specified by the argument *qab* is complete. If there is no active outstanding I/O operation for the QAB, this call returns immediately. This call has no provision for specifying a time-out value. If a time-out is desired, it should be specified when the I/O operation is queued. Issuing this call immediately after an svcIoQueue call with the QFNC$WAIT bit not set in the qab_func field is equivalent to issuing the svcIoQueue call with the QFNC$WAIT bit set. The process can be interrupted when waiting, assuming the state of the processes allows the interrupt otherwise.

EXAMPLES:

svcIoWait

Chapter 17

Terminal System Calls

The terminal functions system calls perform various terminal related functions. They are somewhat like the I/O functions in that all of them take a device descriptor as an argument, but the device specified must be a terminal class device.

svcTrmAttrib - Get or Set Display Attributes

svcTrmAttrib

CALLING SEQUENCE:

XOSSVC svcTrmAttrib(long *handle*, long *func*, void far *data);

VALUE RETURNED:

The value returned is 0 if normal or a negative error code if an error occurred.

DESCRIPTION:

This system call sets or reads the default screen attributes. The handle argument must contain a handle for a terminal device. The func argument specifies the function as follows:

Bit	Meaning
7	Set to change current values
6	Set to return previous values
0-5	Specify set of values
	1 = Text mode base values
	2 = Graphics mode base values
	3 = Current values

The data argument specifies the address of a data block which contains 4 long values. These values specify (in order) forground color, background color, forground fill color, and background fill color. The fill colors are used when ever an area is filled as a result of scrolling.

EXAMPLES:

svcTrmCurPos - Get or Set Cursor Position

svcTrmCurPos

CALLING SEQUENCE:

XOSSVC svcTrmCurPos(long *handle*, long *page*, long *column*, long *row*);

VALUE RETURNED:

The value returned is the previous cursor position encoded as the character position plus 256 times the line position (positive value) if normal or a negative error code if an error occurred.

DESCRIPTION:

The current cursor position on the page specified by the *page* argument on the screen device specified by the device handle given in the *handle* argument is set to the column and row specified in the *column* and *row* arguments. A value of -1 for either the column or row values means not to change that value.

EXAMPLES:

svcTrmCurType - Get or Set Cursor Type

svcTrmCurType

CALLING SEQUENCE:

XOSSVC svcTrmCurType(long *handle*, long *curtype*);

VALUE RETURNED:

The value returned is the previous cursor type unless an error occurred, in which case the negative error code is returned.

DESCRIPTION:

The screen device specified by the device descriptor given in the *handle* argument has the cursor type set as specified by the *curtype* argument. The cursor type is defined as the ending line value plus 256 times the starting line value. Specifying -1 does not change the cursor type.

EXAMPLES:

svcTrmDspPage - Get or Set Current Display Page

svcTrmDspPage

CALLING SEQUENCE:

XOSSVC svcTrmDspPage(long *handle*, long *page*);

VALUE RETURNED:

The value returned is the previous display page if normal or a negative error code if an error occurred.

DESCRIPTION:

The console device specified by the *handle* argument is set to display the display page specified by the *page* argument. If the *page* argument is -1, no change is made.

EXAMPLES:

svcTrmFunction - General Terminal Functions

svcTrmFunction

CALLING SEQUENCE:

XOSSVC svcTrmFunction(long handle, long *func*);

VALUE RETURNED:

The value returned is 0 if normal or a negative error code if an error occurred.

DESCRIPTION:

This system call implements a number of simple terminal functions. These functions are summarized in Table 17.1.

Table 17.1 - svcTrmFunction Functions

Name	Value	Function
TF_RESET	0	Reset terminal data to defaults
TF_ENECHO	1	Enable echoing
TF_DSECHO	2	Disable echoing
TF_CLRINP	3	Clear input buffers
TF_CLROUT	4	Clear output buffer
TF_ENBOUT	5	Enable output

These functions are described in detail below.

TF_RESET = 0 - Reset terminal data to defaults

This function resets the value of all terminal data to the values used when an idle terminal is opened.

TF_ENECHO = 1 - Enable echoing

This function enables echoing of input. This is equivalent to setting the TIM$ECHO bit with the IOPAR_SINMODE device characteristic.

TF_DSECHO = 2 - Disable echoing

This function disables echoing of input. This is equivalent to clearing the TIM$ECHO bit with the IOPAR_CINMODE device characteristic.

TF_CLRINP = 3 - Clear input buffers

This function clears the terminal input buffers. This includes the input ring buffer and the input line buffer.

TF_CLROUT = 4 - Clear output buffer

This function clears the output ring buffer.

TF_ENBOUT = 5 - Enable output

This function enables output that has been disabled by typing ctrl-O.

EXAMPLES:

svcTrmGetAtChr - Get Attribute and Character

svcTrmGetAtChr

CALLING SEQUENCE:

XOSSVC svcTrmGetAtChr(long *handle*, long *page*);

VALUE RETURNED:

The value returned is the character value plus 256 times the attribute value with the high 16-bits set to zero if normal or a negative error code if an error occurred.

DESCRIPTION:

This system call returns the character and attribute from the current cursor position. It is only valid for displays in text mode..

EXAMPLES:

svcTrmGCurCol - Set Graphic Cursor Colors

svcTrmGCurCol

CALLING SEQUENCE:

XOSSVC svcTrmGCurCol(long *handle*, long *number*, long far **colors*);

VALUE RETURNED:

The value returned is 0 if normal or a negative error code if an error occurred.

DESCRIPTION:

This system call sets the colors used to display the hardware graphics cursor in graphic mode. This function is only implemented for display interfaces which require access to IO registers to set the graphics cursor colors. Since memory mapped registers are directly accessable to the application, this function is not necessary when the cursor colors are set using memory mapped registers.

The *handle* argument must specify a handle for a console device which is in graphic mode. The *number* argument specifies the number of colors to set. Currently this value must be 1 or 2. The *colors* argument specifies the address of an array of longs which specify the cursor colors. The first value in the array gives the color used when the value of the pixel in the cursor bitmap is 0. The second value give the color used when the value of the pixel is 1. Each value is encoded in 32 bits as three 12 bit values. The first value (highest order bits) gives the red value, the next the green, and the last (lowest order bits) gives the blue value (RGB 3x12 format)..

EXAMPLES:

svcTrmGCurPat - Set Graphic Cursor Pattern

svcTrmGCurPat

CALLING SEQUENCE:

XOSSVC svcTrmGCurPat(long *handle*, long *type*, long *width*, long *height*,
long *xhot*, long *yhot*, uchar far **pattern*, uchar far **mask*);

VALUE RETURNED:

The value returned is 0 if normal or a negative error code if an error occurred.

DESCRIPTION:

This system call sets the pattern for the hardware graphics cursor in graphic mode. This function is only implemented for display interfaces which require access to IO registers to set the graphics cursor pattern. Since memory mapped registers are directly accessable to the application, this function is not necessary when the cursor pattern is set using memory mapped registers.

The *handle* argument must specify a handle for a console device which is in graphic mode. The *type* argument specifies the cursor type. Currently, this value must be 1. The *width* and *height* arguments specify the size of the cursor bitmap. The *xhot* and *yhot* arguments specify the position of the cursor "hot spot" relative to the upper left corner of the cursor bitmap. The *pattern* argument specifies the address of the pattern bitmap. This is a one bit per pixel bitmap which specifies one of two colors for each cursor pixel (see the svcTrmGCurCol system call description). Each byte contains 8 pixels and each row is byte alligned. The *mask* argument specifies the address of the mask bitmap. This is a one bit per pixel bitmap wehich specifies which cursor pixels are transparent. It is also packed 8 pixels per byte and each line is byte alligned..

EXAMPLES:

svcTrmGCurPos - Set Graphic Cursor Position

svcTrmGCurPos

CALLING SEQUENCE:

XOSSVC svcTrmGCurPos(long *handle*, long *func*, long *xpos*, long *ypos*);

VALUE RETURNED:

The value returned is 0 if normal or a negative error code if an error occurred.

DESCRIPTION:

This system call sets the position of the hardware graphics cursor in graphic mode. This function is only implemented for display interfaces which require access to IO registers to set the graphics cursor position. Since memory mapped registers are directly accessable to the application, this function is not necessary when the cursor position is set using memory mapped registers.

The *handle* argument must specify a handle for a console device which is in graphic mode. The *func* argument is bit encoded as follows:

Bit	Function
1	0 = Do not change position 1 = Set new position
0	0 = Disable cursor 1 = Enable cursor

The xpos and ypos arguments specify the new cursor position in pixel from the upper left hand corner of the screen.

EXAMPLES:

svcTrmLdStdFont - Load Standard Font

svcTrmLdStdFont

CALLING SEQUENCE:

XOSSVC svcTrmLdStdFont(long *handle*, long *table*, long *font*, long *begin*,
long *count*);

VALUE RETURNED:

The value returned is 0 if normal or a negative error code if an error occurred.

DESCRIPTION:

This system call loads one of the standard system fonts into one of the 8 VGA font blocks.

The handle argument must specify a handle for a console device. The table argument specifies the VGA font block number in bits 2-0. Bits 6-3 are not used and should be 0. If bit 7 is 0 the display set up is not changed. If it is 1, the display set up is changed to match the font loaded. The font argument specifies the font to load as follows:

Value	Font
1	Standard 8x8 font
2	Standard 8x14 font
3	Standard 8x16 font
5	Standard 9x8 font
6	Standard 9x14 font
7	Standard 9x16 font

The *begin* argument specifies the first character to load and the *count* argument specifies the number of character to load. The *begin* value should be between 0 and 255 and the sum of the *begin* and *count* values should be 256 or less.

EXAMPLES:

svcTrmLdCusFont - Load Custom Font

svcTrmLdCusFont

CALLING SEQUENCE:

XOSSVC svcTrmLdCusFont(long *handle*, long *table*, long *size*, uchar far **data*, long *begin*, long *count*);

VALUE RETURNED:

The value returned is 0 if normal or a negative error code if an error occurred.

DESCRIPTION:

This system call loads a custom font into one of the 8 VGA font blocks.

The handle argument must specify a handle for a console device. The table argument specifies the VGA font block number in bits 2-0. Bits 6-3 are not used and should be 0. If bit 7 is 0 the display set up is not changed. If it is 1, the display set up is changed to match the font loaded.

The *size* argument specifies the number of scan rows in each character in the font. The *data* argument specifies the address of the array containing the font definition. It contains *size* bytes for each character. The first entry in this array defines the first character which will be stored.

The *begin* argument specifies the first character to load and the *count* argument specifies the number of character to load. The *begin* value should be between 0 and 255 and the sum of the *begin* and *count* values should be 256 or less.

EXAMPLES:

svcTrmMapScrn - Map Screen Buffer

svcTrmMapScrn

CALLING SEQUENCE:

XOSSVC svcTrmMapScrn(long *handle*, long *mode*, void far **buffer*, long *size*, long *offset*);

VALUE RETURNED:

The value returned is the actual size of the memory section created (in bytes, a positive value) if normal or a negative error code if an error occurred.

DESCRIPTION:

The screen buffer for the console device specified by the device handle given in the *handle* argument is mapped at the address specified by the *buffer* argument. The *mode* argument indicates the screen mode desired, while the *size* argument specifies the maximum amount of memory to map. The *offset* argument specifies the offset of the start of the part of the screen buffer to be mapped.

EXAMPLES:

svcTrmSelFont - Select Font

svcTrmSelFont

CALLING SEQUENCE:

XOSSVC svcTrmSelFont(long *handle*, long *primary*, long *secondary*);

VALUE RETURNED:

The value returned is 0 if normal or a negative error code if an error occurred.

DESCRIPTION:

The handle *argument* must specify a handle for a console device. The *primary* and *secondary* arguments specify the primary and secondary font blocks for that consol display. Each must have a value between 0 and 7, inclusive.

EXAMPLES:

svcTrmSetAtChr - Set Attribute and Character

svcTrmSetAtChr

CALLING SEQUENCE:

XOSSVC svcTrmSetAtChr(long *handle*, long *page*, long *chr*, long *attrib*, long *count*);

VALUE RETURNED:

The value returned is 0 if normal or a negative error code if an error occurred.

DESCRIPTION:

This system call sets the character and attribute at the current cursor position on the display page specified by the *page* argument. The *chr* argument specifies the character and the *attrib* argument specifies the argument to store. The *count* argument specifies the number of time the character and attribute are to be stored, starting at the cursor position and incrementing the position for each pair stored. The actual cursor position value, however, is not changed. It is only valid for displays in text mode.

EXAMPLES:

svcTrmSetChr - Set Character

svcTrmSetChr

CALLING SEQUENCE:

XOSSVC svcTrmSetChr(long *handle*, long *page*, long *chr*, long *count*);

VALUE RETURNED:

The value returned is 0 if normal or a negative error code if an error occurred.

DESCRIPTION:

This system call sets the character at the current cursor position on the display page specified by the *page* argument. The *chr* argument specifies the character to store. The *count* argument specifies the number of time the character is to be stored, starting at the cursor position and incrementing the position for each character stored. The actual cursor position value, however, is not changed. It is only valid for displays in text mode. No attribute values are changed.

EXAMPLES:

svcTrmScroll - Scroll Window

svcTrmScroll

CALLING SEQUENCE:

XOSSVC svcTrmScroll(long *handle*, long func, long *page*, long *xul*, long *yul*,
long *xlr*, long y*lr*, long *cnt*, long *fgattrib*,long *bgattrib*);

VALUE RETURNED:

The value returned is 0 if normal or a negative error code if an error occurred.

DESCRIPTION:

This system call scrolls or clears a rectangular window on a text mode screen. The *handle* argument contains a device handle for a console device. The *func* argument specifies the operation to perform as follows:

Value	Operation
1	Scroll up
2	Scroll down
3	Scroll right
4	Scroll left

The *xul* and *yul* specify the x position (character number) and y position (row number) of the upper left hand corner of the rectangle to scroll or clear. The *xlr* and *ylr* arguments sepcify the x position and y position of the lower right hand corner of the rectangle. The *count* argument specifies the number of positions to scroll. A *count* value of 0 means to clear the rectangle (the *func* value is ignored in this case). The *fgattrib* and *bgattrib* argument values are used to fill positions scrolled into the rectangle or when clearing the rectangle.

EXAMPLES:

svcTrmWrtInB - Write to Input Buffer

svcTrmWrtInB

CALLING SEQUENCE:

XOSSVC svcTrmWrtInB(long *handle*, char far **str*, long *cnt*);

VALUE RETURNED:

The value returned is 0 if normal or a negative error code if an error occurred.

DESCRIPTION:

This system writes the characters specified by the arguments *str* and *cnt* into the input buffer of the terminal device specified by the device handle given in the *handle* argument. Note that the data written is specified by address and length; it is not a null terminated string. The data is processed exactly as if it had been typed on the terminal's keyboard. This system call is used by the command shell to implement the command recall feature.

EXAMPLES:

Chapter 18

Screen Symbiont System Calls

This chapter describes the screen symbiont system calls. These system calls provide special functions used by the screen symbiont. They should not be used by other programs. All of these system calls require the SYMBIONT privilege. These system calls provide minimal error checking. Invalid arguments to these calls can cause the entire system to fail.

svcScnMapBufr - Map Physical Screen Buffer

svcScnMapBufr

CALLING SEQUENCE:

XOSSVC svcScnMapBufr(long *xtdb*, char far **buffer,* long *size*);

VALUE RETURNED:

The value returned is the amount of memory mapped (positive) if normal or a negative error code if an error occured.

DESCRIPTION:

The *xtdb* argument specifies the exec offset of the terminal TDB for the virtual screen. The *buffer* argument specifies the address where the display buffer is to be mapped. The *size* argument specifies the amount of screen memory to map. This memory must be unallocated.

This function maps the physical text mode display buffer for use by the screen symbiont. This mapping is NOT affected by any screen symbiont functions and always directly maps the physical buffer.

EXAMPLES:

svcScnMaskWrt - Masked Write to Screen Buffer

svcScnMaskWrt

CALLING SEQUENCE:

XOSSVC svcScnMaskWrt(long *xtdb*, long *func*, long *firstpg*, long *numpgs*,
long *width*, long *height*, long far **pglist*, char far **pgbufr*);

VALUE RETURNED:

The value returned is 0 if nothing was written, 1 if something was written or a negative error code if an error occurred.

DESCRIPTION:

The *xtdb* argument specifies the exec offset of the terminal TDB for the virtual screen.

Other than the *width* and *height* arguments, the arguments are the same as for the svcScnTrans system call.

This function is used by the screen symbiont to update the physical display buffer when the session menu is on the screen. It must be called for text (not character generator) pages only. It actually does a number of separate operations required here:

1. The page modified status is updated from all mappings of the display buffer
2. If none of the indicated pages have been modified, all the indicated pages are blocked and a value of 0 is returned.
3. Otherwise, the indicated pages are written to the physical buffer with the area for the menu in the upper right hand corner (as specified by the *width* and *height* arguments of the display masked out and a value of 1 is returned.

This function is used even though the screen symbiont has mapped the physical buffer. This is because it combines several operations and eliminates several seperate system calls and because it ensures that the updating of the buffer is atomic.

This function is only valid when the display is in text mode.

EXAMPLES:

svcScnTrans - Transfter Data for Screen Symbiont

svcScnTrans

CALLING SEQUENCE:

XOSSVC svcScnTrans(long *xtdb*, long *func*, long *firstpg*, long *numpgs*, long far **pglist*, char far **pgbufr*);

VALUE RETURNED:

The value returned is the raw hardware cursor position (read transfers) or 0 (write transfers) if normal or a negative error code if an error occurred.

DESCRIPTION:

Bit 31 of the func argument specifies a read (0) or write (1) operation. The meanings of the remaining bits of func depends on this bit. When bit 31 is 0 (read) the remaining bits are as follows:

Name	Bit	Meaning
SSR$FCURSOR	4	0 = Do not freeze cursor position 1 = Freeze cursor position
SSR$BLKVIRT	1	0 = Do not block access to virtualized pages 1 = Block access to virtualized pages
SSR$VIRTALL	0	0 = Only virtualize modified pages 1 = Virtualize all pages read

When bit 31 is 1 (write) the remaining bits are as follows:

Name	Bit	Meaning
SSW$SWITCH	3	0 = Do not switch screens 1 = Make screen specified by xtdb the current screen
SSW$UNBLKONLY	2	0 = Normal 1 = Unlock pages only
SSW$CLRSSS	1	0 = Do not change keyboard system-shift state 1 = Clear keyboard system-shift state when done

The page list is an array of longs, with one long for each page to read. Each long has the following format:

Name	Bit(s)	Description
PL$MODIFIED	31	0 = Page has been cleared (bits 15-0 give contents) 1 = Page has been modified
PL$VIRTUAL	30	0 = Page maps to physical display buffer 1 = Page has been virtualized
PL$BLOCKED	29	0 = Access to page is allowed 1 = Access to page is blocked
PL$INCMOD	28	0 = Page has not been modified since last check 1 = Page has been modified since last check
	15-0	Gives contents for page - For pages 0 - 15 this is the 16-bit attribute/character value for each character position. For pages 16 - 31 this is 2 * F + h where F is the font index and h is 0 for the first half of the character set and 1 for the second half.

The *firstpg* argument specifices the first page to read or write.

The *numpgs* argument specifies the number of pages in the page list.

Each page is 4096 bytes in length. The *pgbufr* argument point to an array of 4096 byte buffers, one for each page in the page list. This buffer may be virtually allocated.

For the read function, all modified pages are read and virualized. If function bit 0 is set all pages not already virtualized are read and virualized. Access is blocked to all pages which are not virtualized and all pages which are virtualized are unblocked. If function bit 1 is set, access is blocked to all pages. If a page is not read, the corresponding buffer page is not touched. If the buffer address is null, no pages are read, virtualized, or unblocked. The page list entry is stored for all pages in all cases.

The read function is used by the screen symbiont in several ways.

1. When placing the session menu on the screen, it is used to virtualize any modified visible pages and block access to all visible pages.

2. When switching screens, it is used to virtualize all modified pages and block access to all unmodified pages.

3. When a blocked unmodified page is touched, it is used to virtualize and unblock the touched page.

The read function is only valid when the display is in text mode.

For the write function, if the screen specified is not the current screen for the display, it is first made the current screen. The page list format is described for the vgassread function. The page list is set up by the caller. Display pages marked as modified (bit 31 = 1) are written from the corresponding buffer page. Display pages marked as unmodified (bit 31 = 0) are cleared to the value given in bits 15-0. All pages are unvirtualized and unblocked. If function bit 2 = 1, pages are unblocked only. In this case the buffer address must be specified to provide the access for mapping virtualized pages, but the buffer is not accessed.

If the write function is issued when the display is not in text mode, the only thing done is to clear the system-shift state. Everything else is ignored.

EXAMPLES:

svcScnUtil - Screen Symbiont Utility Functions

svcScnUtil

CALLING SEQUENCE:

XOSSVC svcScnUtil(long *xtdb*, long *data*);

VALUE RETURNED:

The value returned is the exec offset of the TDB (positive value) if the *xtdb* argument has a value less than 0x1000 or 0 if normal or a negative error code if an error occurred.

DESCRIPTION:

This system call implements several utility functions for the screen symbiont. If the *xtdb* argument is less than 0x1000 this argument is assumed to contain a device handle for a console device and the function returns the exec offset of that console's TDB (terminal data block). The *data* argument is ignored.

If the *xtdb* argument is 0x1000 or greater it is assumed to be the exec offset of a console TDB. In this case the data argument specifies the function as follows:

Value	Function
-1	Remove cursor from physical screen
0	Wake up main program level using xtdb value + 1 as wait index
> 0	Wake up extended fork frame with this selector

EXAMPLES:

Chapter 19

Device Dependent I/O Functions

This chapter describes the device dependent I/O functions. This functions are implemented using the QFNC_SPECIAL function of the svcIoQueue system call. These are described here rather than in the chapter which describes the other svcIoQueue functions because of the amount of material and the fact that much of it is not relavent to most programs since these tend to be fairly specialized functions.

These descriptions are organized by device and then by function. Some devices do not implement any device dependent functions as thus are not included here.

DISK Devices

DISK special device functions are only implemented for physical disk devices, not for files, the the underlying disk device must be opened specifying either O$RAW or O$PHYSIO to use these functions. The DISK special device functions are summerized in table 19.1 below.

Table 19.1 - DISK special device functions

Name	Value	Description
DISMOUNT	2	Dismount disk
MOUNT	3	Mount disk
FORMAT	4	Format disk

These functions are described in detail below.

DISMOUNT = 2 - Dismount disk

The disk is dismounted if it was mounted. The value returned is 1 if the disk was mounted, 0 if it was not mounted, or a negative error code if an error occured.

MOUNT = 3 - Mount disk

This function is intended to allow mounting a disk with a specified file system. It is not implemented in the current version of XOS. In this version, all disk mounting is done automatically with the type of file system based on data read from the disk.

FORMAT = 4 - Format disk

This function is intended to allow low level formatting of floppy and hard disks. It is not implemented in the current version of XOS.

SPL Devices

There are no special device functions for SPL class devices.

TAPE Devices

There are a number of special device functions implemented for TAPE class devices. These are summerized in table 19.2 below.

Table 19.2 - TAPE special device functions

Name	Value	Description
TAPE_UNLOAD	1	Unload tape
TAPE_REWIND	2	Rewind tape
TAPE_FORMAT	3	Format tape
TAPE_RETEN	4	Retension tape
TAPE_WRITEFM	5	Write filemarks
TAPE_WRITESM	6	Write setmarks
TAPE_LOCK	7	Lock/unlock tape mounting
TAPE_ERASEGAP	8	Erase gap
TAPE_ERASEALL	9	Erase gap
TAPE_SKPREC	10	Skip records
TAPE_SKPFILE	11	Skip filemarks
TAPE_CONFILE	12	Skip to consective filemarks
TAPE_SKPSET	13	Skip setmarks
TAPE_CONSET	14	Skip to consective setmarks
TAPE_SKP2EOD	15	Skip to end-of-data

These functions are described in detail below.

TAPE_UNLOAD = 1 - Unload tape

This function rewinds and then unloads the tape. The value returned is 0 if normal or a negative error code if an error occured.

TAPE_REWIND = 2 - Rewind tape

This function rewinds the tape. The value returned is 0 if normal or a negative error code if an error occured.

TAPE_FORMAT = 3 - Format tape

This function is not implemented in the current version of XOS.

TAPE_RETEN = 4. - Retension tape

This function preforms a retension operation which generally scans to the end of the physical tape and then rewinds it. The exact behavior depends on the type of tape drive. Not all tape drives implement this function. The value returned is 0 if normal or a negative error code if an error occured.

TAPE_WRITEFM = 5 - Write filemarks

This function writes the number of filemarks specified by the value in qab_count. The value returned is 0 if normal or a negative error code if an error occured.

TAPE_WRITESM = 6 - Write setmarks

This function writes the number of setmarks specified by the value in qab_count. The value returned is 0 if normal or a negative error code if an error occured.

TAPE_LOCK = 7 - Lock/unlock tape mounting

This function locks or unlocks tape mounting or unmounting according to the value in qab_count. If the value is 0 the tape is unlocked. Otherwise it is locked. The value returned is 0 if normal or a negative error code if an error occured.

TAPE_ERASEGAP = 8 - Erase gap

This function erases an interrecord gap on the tape. The value returned is 0 if normal or a negative error code if an error occured.

TAPE_ERASEALL = 9 - Erase entire tape

This function erases the entire tape. The value returned is 0 if normal or a negative error code if an error occured.

TAPE_SKPREC = 10 - Skip records

This function skips the number of records specified in qab_count. The value returned is 0 if normal or a negative error code if an error occured.

TAPE_SKPFILE = 11 - Skip filemarks

This function skips the number of filemarks specified in qab_count. The value returned is 0 if normal or a negative error code if an error occured.

TAPE_CONFILE = 12 - Skip to consective filemarks

This function skips forward until the number of consective filemarks specified in qab_count are found. The value returned is 0 if normal or a negative error code if an error occured.

TAPE_SKPSET = 13 - Skip setmarks

This function skips the number of setmarks specified in qab_count. The value returned is 0 if normal or a negative error code if an error occured.

TAPE_CONSET = 14 - Skip to consective setmarks

This function skips forward until the number of consective setmarks specified in qab_count are found. The value returned is 0 if normal or a negative error code if an error occured.

TAPE_SKP2EOD = 15 - Skip to end-of-data

This function skips forward until an end-of-data mark is found. The value returned is 0 if normal or a negative error code if an error occured.

TRM Devices

There are no special device functions for TRM class devices.

PCN Devices

The PCN device is a very specialized device which implements the server side of the psuedo-console device. The pseudo-console device is a device which, with the co-operation of a server process, emulates a text-mode only VGA display and console keyboard.

The client side device is implemented as a low level terminal driver (type PCN) which very closely emulates the actual VGA and console keyboard device interfaces, including all display and keyboard BIOS functions, direct access to the display buffer, direct access to the keyboard controller IO ports, and use of the keyboard hardware interrupt. The PCN device supports display mode changes (text modes only) and screen format and font changes using the INT 10 BIOS functions. It does NOT support screen format or font changes made doing direct writes to the display controller registers or direct writes to the character generator pages.

The server side interface is special to this device. It mainly uses the special device function and asynchronous signals for interaction with the server program The server program MUST be specially written to use this device.

The IOPAR_SIGVECT1 and IOPAR_SIGVECT2 IO parameters are used to specify the signal vectors used. IOPAR_SIGVECT1 sets the hang up signal vector number. The IOPAR_SIGVECT2 parameter is used to specify the mapped mode done signal vector. This signal occures when mapped mode is disabled. It uses two sigma; data items; the IOPAR_SIGDATA value, and the mapped mode buffer page modified bits.

PCN names are of the form PCNn or PCNnPm where n is the primary unit number and m is the secondary unit number. When the PCNn format is used and raw or physical IO is specified, the partial DCB for the primary unit is returned. This DCB does not allow any IO and is useful only for getting and setting device characteristics. If raw or physical IO is not specified, a full DCB is created and a new name of the form PCNnPm is assigned. This is the normal method of creating a new instance of a PCN device for use by a server. If the PCNnPm form is specified, the full DCB of that name will be returned if it exists, but a new DCB will not be created.

The client side device operates in three distinct modes: stream, mapped, and blocked. When operating in stream mode, all data output to the TRMPn device is written to both the display and shadow buffers and is sent to the server device as input data. The display buffer is mapped read-only for all processes which have it mapped. When operating in mapped mode, data is written to the display buffer only. No data is sent to the server device. When operating in blocked mode, any attempt

to modify the display contents, either by a system call or by writting directly to the screen buffer or to change the display mode, causes the client program to be suspended until the device leaves blocked mode.

The device initially operates in stream mode. It stays in stream mode until a user process attempts to write directly to the display buffer, when it switches to mapped mode and starts a timer. It also forces any buffered server device input to be made immediately available to the server device. It stays in mapped mode until the timer expires or until a display output function for the client device will cause the screen (or part of the screen) to be scrolled, at which time it enters blocked mode (which implies changing all user mapped buffers to read-only) and sends a notification to the server device (signal). It stays in blocked mode until a special device function is issued for the server device switching it to stream mode (PCSF_UNBLOCK). This action also wakes up any processes which were suspended while the client device was in blocked mode.

The intent of all of this is to provide reasonable performance for both character stream display output and direct screen writes. Note that once a user program writes directly to the screen, all writes to the screen, either by a user program or by the system routines which implement terminal output, are treated exactly the same, except that any system call which causes scrolling receives special treatment.

When the server program receives a notification that the mapped mode timer has expired or that a scrolling operation has been requested when in mapped mode, it must compare the display and shadow buffers to determine what data has been changed while in mapped mode. The device does indicate which pages have been changed. The server program must update the shadow buffer to be the same as the display buffer before switching the client side device back to stream mode.

A number of special device functions are used to provide the special functionality need for this device. These are summerized in table 19.3 below.

Table 19.3 - PCN Special Device Functions

Name	Value	Description
PCN_SETBUF	1	Set up buffer memory
PCN_MODBUF	2	Modify buffer memory
PCN_MODESET	3	Mode set complete
PCN_DISPTYPE	4	Set display type
PCN_FONTSET	5	Set font parameters
PCN_STREAMMD	6	Set to stream mode

PCN_SETBUF = 1 - Set up buffer memory

This function sets up screen buffer for use. Qab_buffer1 contains the base address of the msect will contain the buffer, which must be a private msect. Qab_count contains the number of pages to use, which in this version of XOS must be 8. Also, the msect must be at least 64KB in size.

The display buffer is set up at the beginning of the msect using the indicated number of pages. The shadow is set up immediately following the display buffer, also using the indicated number of pages. The pages used can be either real or virtual but must be writable. Note that once the buffer is set up, the pages used cannot be deallocated until the device is closed. The buffer cannot be changed or deallocated except by closing the server side device.

PCN_MODBUF = 2 - Modify buffer memory

This function provides page-by-page control of the screen buffer. qab_buffer contains the address of buffer page list and qab_count contains the number of buffer page list entries. Each buffer page list entry is consists of 4 bytes. The low 3 bytes specify a page number relative to the start of the buffer. The high byte specifies the new state for the page as follows:

0 = Blocked
1 = Read-only
2 = Read/write

PCN_MODESET = 3 - Mode set complete

This function indicates that a mode set operation is complete. Qab_buffer1 contains the address of a data block containing mode set values. This is the data block filled in by the svcTrmDspMode system call preceeded by the long value returned by that call. The format of this data block is as follows:

Offset	Size	Description
0x0	4	Mode bits
0x4	4	Active fonts
0x8	4	Number of text columns
0xC	4	Number of text rows
0x10	4	Number of horizontal pixels
0x14	4	Number of vertical pixels

0x18	4	Display type value

PCN_DISPTYPE = 4 - Set display type

This function sets the type of the emulated display. The display type is specified by the value in qab_count.

PCN_FONTSET = 5 - Set font parameters

This function indicates that a font set operation is complete. Qab_buffer1 contains the address of a data block containing the font set values which has the following format:

Offset	Size	Description
0x0	4	Active fonts
0x4	4	Number of text columns
0x8	4	Number of text rows
0xC	4	Number of horizontal pixels
0x10	4	Number of vertical pixels

PCN_STREAMMD = 6. - Set to stream mode

This function set the device to stream mode. This is normally called after a leaving mapped mode singal to switch back to normal stream mode operation. It does not use any additional QAB fields.

PPR Devices

There are no special device functions for PPR class devices.

NET Devices

There are no special device functions for NET class devices.

SNAP Devices

There are no special device functions for SNAP class devices.

ARP Devices

There are no special device functions for ARP class devices.

IPS Devices

The IPS device class implements a number of special device functions, all of which are used to manage and use the built-in DNS resolver and cache. These functions are summerized in table 19.4 below.

Table 19.4 - IPS Special Device Functions

Name	Value	Description
IPS_FINDIPA	1	Map domain name to IP addresses
IPS_FINDCNAME	2	Map domain name to canonical name
IPS_FINDMAIL	3	Map domain name to mail names
IPS_FINDPTR	4	Map domain name to pointer name
IPS_DUMP	7	Dump DNS cache
IPS_INITCACHE	8	Initialize DNS cache
IPS_CLRCACHE	9	Clear DNS cache
IPS_ADDIPA	10	Add IP address entry to cache
IPS_ADDCNAME	11	Add canonical name entry to cache
IPS_ADDMAIL	12	Add mail name entry to cache
IPS_ADDPTR	13	Add pointer entry to cache
IPS_WAKEIPA	14	Wake up waiters for IP address
IPS_ERROR	15	Set error for domain name
IPS_REMOVE	16	Remove entry from cache

These functions are mainly intended for the use of the user mode part of the DNS resolver. In general they should not be used by other programs. They are described in detail below.

IPS_FINDIPA = 1 - Map domain name to IP addresses

Data is specified and returned in the buffer pointed to by qab_buffer1 with length specified in qab_count. When the function is called, the first byte of the buffer contains modifier bits as follows:

Bits 7-4	Recursion level (should be 0 except for recursive DNS server calls)
Bits 3-1	Reserved, must be 0
Bit 0	Use counted label format for domain name (otherwise use period format)

The remainder of the buffer contains the domain name in the specified format. On return, the buffer contains a list of IP addresses. Each address

starts with a count byte which specifies the number of bytes in the address. (This is currently always 4.) This is followed by the specified number of address bytes.

Value returned is the number of IP addresses returned if normal or a negative error code if error. If some data was not returned because the buffer was too small, bit 30 is set.

IPS_FINDCNAME = 2 - Map domain name to canonical name

Data is specified and returned in the buffer pointed to by qab_buffer1 with length specified in qab_count. When the function is called, the first byte of the buffer contains modifier bits as follows:

Bits 7-4	Recursion level (should be 0 except for recursive DNS server calls)
Bits 3-1	Reserved, must be 0
Bit 0	Use counted label format for domain name (otherwise use period format)

The remainder of the buffer contains the domain name in the specified format. On return, the first byte of the buffer contains the length of the name returned. The returned name immediately follows this byte.

Value returned is 1 if normal or a negative error code if error. A value of 0x40000000 is returned if the buffer is too small for the name.

IPS_FINDMAIL = 3 - Map domain name to mail names

Data is specified and returned in the buffer pointed to by qab_buffer1 with length specified in qab_count. When the function is called, the first byte of the buffer contains modifier bits as follows:

Bits 7-4	Recursion level (should be 0 except for recursive DNS server calls)
Bits 3-1	Reserved, must be 0
Bit 0	Use counted label format for domain name (otherwise use period format)

The remainder of the buffer contains the domain name in the specified format. On return, the buffer contains a list of mail names in the specified format. Each name is preceeded by a count byte which specifies the length of the name.

Value returned is the number of names returned if normal or a negative error code if error. Bit 30 is set if one or more names was not returned because the buffer was too small.

IPS_FINDPTR = 4 - Map domain name to pointer name

Data is specified and returned in the buffer pointed to by qab_buffer1 with length specified in qab_count. When the function is called, the first byte of the buffer contains modifier bits as follows:

Bits 7-4	Recursion level (should be 0 except for recursive DNS server calls)
Bits 3-1	Reserved, must be 0
Bit 0	Use counted label format for domain name (otherwise use period format)

The remainder of the buffer contains the domain name in the specified format. On return, the buffer contains a list of pointer names in the specified format. Each name is preceeded by a count byte which specifies the length of the name.

Value returned is the number of names returned if normal or a negative error code if error. Bit 30 is set if one or more names was not returned because the buffer was too small.

IPS_DUMP = 7 - Dump DNS cache

The cache contents is returned in the buffer specified by qab_buffer1 with length specified by qab_count. The data is stored as a sequence of blocks for each domain name entry as follows:

Size	Description
1	Number of CNAME definitions
1	Number of A definitions
1	Number of MX definitions
1	Number of PTR definitions
4	Time-to-live
1	Error bits
1	Length of domain name
n	Domain name (counted label format)

Each domain name block is followed by a block for each CNAME definition, followed by a block for each A definitions, followed by a block for each MX definition, followed by a block for each PTR deinition as followed:

Size	Description
4	Time-to-live
2	Data value
1	Length of definition
n	Definition

Value returned is the number of bytes stored if normal or a negative error code if an error occured.

IPS_INITCACHE = 8 - Initialize DNS cache

This function initializes the DNS kernel cache. If a DNS cache exists, it is removed. The buffer pointed to by qab_buffer1 contains the maximum number of DNS entires allowed in the first 4 bytes followed by the IPM destination name (up to 31 characters)

Value returned is 0 if normal or a negative error code if an error occured..

IPS_CLRCACHE = 9 - Clear DNS cache

This function removes the DNS kernel database. If the DNS database does not exist, it does nothing. It has no parameters.

Value returned is 0 if normal or a negative error code if an error occured.

IPS_ADDIPA = 10 - Add IP address entry to cache

Data is specified in the buffer pointed to by qab_buffer1 with length specified in qab_count.

The format of the data is as follows:

Size	Description
4	Offset of first IP address block
n	Domain name (counted label format)

Each IP address is stored in a block as follows:

Size	Description
4	Offset of next IP address block
4	Time to live (seconds)
4	Reserved, must be 0
4	Length of IP address, must be 4
4	IP address

The value returned is 0 if normal or a negative error code if an error occured.

IPS_ADDCNAME = 11 - Add canonical name entry to cache

Data is specified in the buffer pointed to by qab_buffer1 with length specified in qab_count. The format of the data is as follows:

Size	Description
4	Offset of first canonical name block
n	Domain name (counted label format)

The length of the buffer must exactly match the length of the domain name, that is, it must be n + 24. The canonical name is stored in a block as follows:

Size	Description
4	Reserved- must be 0
4	Time to live
4	Reserved- must be 0
4	Length of cononical name
n	Cononical name (counted label format)

The value returned is 0 if normal or a negative error code if an error occured.

IPS_ADDMAIL = 12 - Add mail name entry to cache

Data is specified in the buffer pointed to by qab_buffer1 with length specified in qab_count. Format of the data is as follows:

Size	Description
4	Offset of first mail name block
n	Domain name (counted label format)

Each main name is stored in a block as follows:

Size	Description
4	Offset of next mail name block
4	Time to live (seconds)
4	Preference value
4	Length of mail name

n	Mail name (counted label format)

The value returned is 0 if normal or a negative error code if an error occured.

IPS_ADDPTR = 13 - Add pointer entry to cache

Data is specified in the buffer pointed to by qab_buffer1 with length specified in qab_count. Format of the data is as follows:

Size	Description
4	Offset of first pointer name block
n	Domain name (counted label format)

Each pointer name is stored in a block as follows:

Size	Description
4	Offset of next mail name block
4	Time to live (seconds)
4	Reserved- must be 0
4	Length of pointer name
n	Pointer name (counted label format)

The value returned is 0 if normal or a negative error code if an error occured.

IPS_WAKEIPA = 14 - Wake up waiters for IP address

This function is intended for use when the resolver/server has received a request for an A record (IP address) but has found a CNAME record instead. Data is in a buffer pointed to by qab_buffer1 with length specified by qab_count. It consists of a domain name in counted label format.

Value returned is 0 if normal or a negative error code if error. Note that if the domain name is not in the cache, no error is reported but nothing is done.

IPS_ERROR = 15 - Set error for domain name

This functions stores an entry in the DNS cache indicating an error for the name specified. Everyone waiting on the name (for any record type) are woke up. Data is in a buffer pointed to by qab_buffer1 with length specified by qab_count. Format of the data is as follows:

Size	Description	
2	Time to live value	
2	Error type:	0 = ER_NNNDF (name not defined)
		1 = ER_NNSNA (network name server not available)
n	Domain name (counted label format)	

The value returned is 0 if normal or a negative error code if error. Note that if the domain name is not in the cache, no error is reported but nothing is done.

IPS_REMOVE = 16 - Remove entry from cache

Data is specified in the buffer pointed to by qab_buffer1 with length specified in qab_count. Format of the data is as follows:

Size	Description
1	Length of domain name (n)
n	Domain name (xxx.xxx.xxx)

The specified entry is removed from the DNS cache.

The value returned is 0 if normal or a negative error code if an error occured.

UDP Devices

There are no special device functions for UDP class devices.

TCP Devices

There are no special device functions for TCP class devices.

TLN Devices

There are no special device functions for TLN class devices.

RCP Devices

There are no special device functions for RCP class devices.

XFP Devcies

There are no special device functions for XFP class devices.

Chapter 20

svcLoadLKE System Call

Loadable kernel extensions or LKEs provide a means of adding executable code to the XOS kernel after the system is running. Most LKEs provide support for various hardware devices and generally fall into the catagory of items ususually refered to as device drivers. The LKE mechanism is much more general than this, however, and can be used to extend the kernel functionality in many ways.

A complete description of the XOS kernel environment where an LKE executes is beyond the scope of this manual. It is described in the XOS Kernel Environment manual. The mechanism used to load an LKE, however, is a system call and thus is described here.

CALLING SEQUENCE:

XOSSVC svcSysLoadLke(struct lkeargs **args*);

VALUE RETURNED:

The value returned 0 if normal or a negative error code if an error occurred.

DESCRIPTION:

This system calls provides the means for loading LKEs (loadable kernel extensions). This call is intended mainly for use by the LKELOAD utility program, which does significant formating of the raw data to be loaded, which normally comes from an executable image. It also resolves exported symbols by directly reading the kernel's internal symbol tables. Normally, user programs should not execute this system call. It is considered an internal function any may change significantly in future versions of XOS.

The argument block pointed to by the *args* argument has the following format:

Name	Size	Set by	Description	
lla_state	4	Both	State	
			Name	Value
			LLS_BEGIN	0
			LLS_LOAD	1
			LLS_WAIT	2
			LLS_DONE	3
lla_value	4	System	Returned value	
lla_doffset	4	System	Offset for data msect	
lla_dsize	4	System	Final size for data msect	
lla_dcount	4	User	Size of data msect data	
lla_ddata	8	User	Address of data for data msect	
lla_coffset	4	System	Offset for code msect	
lla_csize	4	System	Final size for code msect	
lla_ccount	4	User	Size of code msect data	
lla_cdata	8	User	Address of data for code msect	
lla_init	4	User	Offset of initialiation routine	
lla_char	8	User	Address of characteristics list	
lla_xcount	4	User	Size of exported symbol table data	
lla_xdata	8	User	Address of exported symbol table data	
lla_soffset	4	User	Offset for debugger symbol table msect	
lla_ssize	4	System	Final size for debugger symbol table msect	
lla_scount	4	User	Size of debugger symbol table data	
lla_sdata	8	User	Address of debugger symbol table data	

The sequence to load an LKE is as follows:

1 - Read the .LKE file and determine the names of all referenced common data areas.

2 - Initialize the argument block and issue the svcSysLoadLke call - address of code msect data must be supplied - this call will always return without waiting.

3 - Use the relocation information returned to relocate the code and data msects, fill in count and address fields in the argument block, and issue the svcSysLoadLke call again (do not change the lla_state field). This call may wait if IO is done by the LKE's initialization routine. The calling process is uninterruptable during this wait.

Warning: None of the arguments block fields marked s must be modified after the initial SvcSysLoadLke call and before the second svcSysLoadLke call returns.

The load may be aborted before the second svcSysLoadLke call by setting the lla_init field to 0 and calling svcSysLoadLke.

Warning: The first call to svcSysLkeLoad obtains the exec memory allocate resource, which is given up by the second call. Keeping this resource for an extended time may cause many system operations to hang, so the second call should be made as soon as possible! Also, if the process terminates with the exec memory allocate resource, the system will crash! The program must ensure that the user cannot terminate it (by typing ^C, for example) between the two calls to svcLoadLke,

Appendix A

List of System Calls

This appendix provides a complete alphabetical list of the XOS system calls.

Name	Description
svcIoCancel	Cancel I/O Request
svcIoClose	Close Device
svcIoControl	I/O Request Control
svcIoDefLog	Define Logical Name
svcIoDelete	Delete File
svcIoDevParm	Get or Set Device Parameters
svcIoDstName	Build Destination Name
svcIoDupHandle	Duplicate Device Handle
svcIoFindLog	Find Logical Name
svcIoInBlock	Input Block
svcIoInBlockP	Input Block/Parameter List
svcIoInSingle	Input Byte
svcIoInSingleP	Input Byte/Parameter List
svcIoOpen	Open Device or File
svcIoOutBlock	Output Block
svcIoOutBlockP	Output Block/Parameter List
svcIoOutSingle	Output Byte
svcIoOutSingleP	Output Byte/Parameter List
svcIoOutString	Output String

Name	Description
svcIoOutStringP	Output String/Parameter List
svcIoPath	Set Default Path
svcIoQueue	Queue I/O Request
svcIoRename	Rename File
svcIoRun	Run or load program
svcIoSetPos	Set I/O Position
svcIoWait	Wait Until I/O is Complete
svcMemBlkAlloc	Allocate linear address space block
svcMemBlkChange	Change size of linear address space block
svcMemBlkFree	Give up linear address space block
svcMemChange	Change Memory Allocation
svcMemConvShr	Convert to Shared Section
svcMemCreate	Create New Segment
svcMemDebug	Memory Debug Functions
svcMemDescAlloc	Allocate segment descriptor
svcMemDescFind	Find segment descriptor
svcMemDescFree	Give up segment descriptor
svcMemDescRead	Read segment descriptor
svcMemDescSet	Set single segment descriptor value
svcMemDescWrite	Write segment descriptor
svcMemDosSetup	Set up DOS memory
svcMemLink	Link Segment Selectors
svcMemLinkShr	Link to Shared Section
svcMemMap	Map Physical Section
svcMemMove	Move Memory Section
svcMemPageType	Change Memory Page Type
svcMemRemove	Remove Segment
svcMemRmvMult	Remove multiple segments
svcMemSegType	Change Segment Type

Name	Description
svcMemWPFunc	Watch-point functions
svcMemWPSet	Set watch-point
svcSchAlarm	Alarm Functions
svcSchClrEvent	Clear Event(s)
svcSchCtlCDone	Report ctrl-C Processing Done
svcSchDismiss	Dismiss Software Interrupt
svcSchExit	Terminate Process
svcSchGetVector	Get Software Interrupt Vector
svcSchIntrProc	Interrupt Child Process
svcSchKill	Terminate Any Process
svcSchMakEvent	Create or remove event cluster
svcSchRelEvent	Release event
svcSchResEvent	Reserve event
svcSchSetEvent	Set event
svcSchSetLevel	Set Signal Level
svcSchSetVector	Set Signal Vector
svcSchSpawn	Create child process
svcSchSuspend	Suspend Process
svcSchWaitProc	Wait for Process to Terminate
svcSchWaitMEvent	Wait for multiple events
svcSchWaitSEvent	Wait for single event
svcScnMapBufr	Map physical screen buffer
svcScnMaskWrt	Masked write to display memory
svcScnTrans	Transfer data for screen symbiont
svcScnUtil	Screen symbiont utility functions
svcSysCmos	CMOS memory function
svcSysDateTime	Date and time functions
svcSysDefEnv	Define environment string
svcSysErrMsg	Get error message text
svcSysFindEnv	Find enviroment string

Name	Description
svcSysLoadLke	Load LKE
svcSysLog	Place entry in system log file
svcTrmAttrib	Get or set display attributes
svcTrmCurPos	Get or set cursor position
svcTrmCurType	Get or set cursor type
svcTrmDspPage	Get or set current display page
svcTrmFunction	General terminal functions
svcTrmGetAtChr	Get attribute and character
svcTrmGSetCurPat	Set graphics cursor pattern
svcTrmGSetCurCol	Set graphics cursor colors
svcTrmGSetCurPos	Set graphics cursor position
svcTrmLdStdFont	Load standard font
svcTrmLdCusFont	Load custom font
svcTrmMapScrn	Map screen buffer
svcTrmSelFont	Select font
svcTrmSetAtChr	Set Attribute and Character
svcTrmSetChr	Set character
svcTrmScroll	Scroll window
svcTrmWrtInB	Write to input buffer

Appendix B

System Error Codes

This appendix provides a complete listing and discussion of all error codes returned by XOS system service calls. All system service calls and many C library functions return one of the following error codes when an error is detected. All codes are unique; their meaning is independent of the routine which indicated the error. In the following alphabetic listing, each error code is specified as follows:

ERCOD = n - Message text

where ERCOD is the three to five character mnemonic for the error code, n is the numeric value of the error code (decimal), and "Message text" is the standard message text associated with the error (as returned by the svcSysErrMsg system call).

This line is followed by a discussion of the error condition indicated. This listing of error codes is in alphabetical order by the five character error code mnemonic. A numerically ordered list follows. Note that the symbol used to reference the value (for both C and assembler) is obtained by prefixing the characters "ER_" before the the error mnemonic. Thus the symbol for the ADRER error code is ER_ADRER.

The numeric listing includes just the value, mnemonic, and message text in tabular format for quick reference given the numeric value of the error code.

Alphabetical List of System Error Codes

ABORT = -150 - I/O operation aborted

An I/O operation was canceled after it was started. At least the amount of data indicated by the qab_amount value was transferred, but more may have been transferred.

ACT = -28 - Device is active

A function was specified for an active device which required that the device be inactive.

ADRER = -60 - Address out of bounds

An invalid address was specified as a parameter for a system service call.

ALDEF = -141 - Already defined

An attempt was made to define an entity (such as a shared memory segment or interprocess message name) with a name which was already in use.

BDALM = -205 - Bad alarm handle

The alarm handle specified as an argument for the svcSchAlarm system call did not correspond to an active alarm.

BDDBK = -56 - Bad disk block number

An illegal disk block number was specified.

BDDVH = -57 - Bad device handle

A device handle, which did not specify a currently open device or file, was specified.

BDLNM = -99 - Bad logical name

The logical name specified contained illegal characters or was too long.

BDNAM = -18 - Bad process name

The process name specified contained illegal characters or was too long.

BDPID = -19 - Bad process ID

A process ID was specified which did not correspond to a possible process slot or to an existing process (if a reference to an existing process was required).

BDSPC = -29 - Bad device or file specification

An improperly formed device or file specification was specified.

BPIPE = -79 - Pipe error

An attempt was made to write to a pipe or to read from an empty pipe, the other end of which had been closed.

BUSY = -40 - File or device is busy

An attempt was made to perform some operation on a file or device which was busy for purposes of the attempted operation. For example, an attempt was made to delete or supersede a file which is currently being superseded.

CAASP = -184 - Close action already specified

An attempt was made to specify a close action for a file using the IOPAR_CLSACT I/O parameter when a close action had already been specified for the file. Only one close action can be specified for an open file.

CAERR = -185 - Close action error

An error occurred when attempting to perform the close action specified for a file.

CANCL = -151 - I/O operation cancelled

An I/O operation was cancelled after it queued but before it was started. No data was transferred.

CCMSS = -190 - Cannot change memory section size

An attempt was made to change the size of a memory section which has a constant size. Shared sections and sections which map device buffers usually have constant size.

CDAAD = -161 - LKE common data area already defined

A common data area being defined by an LKE is already defined.

CDAND = -162 - LKE common data area not defined

A common data area referenced by an LKE was not defined when the LKE was loaded. This error can only occur when loading an LKE.

CHARF = -16 - Illegal characteristic function

An illegal function was specified for a device or class characteristic.

CHARM = -17 - Required characteristic missing

A required device or class characteristic was not specified.

CHARN = -12 - Illegal characteristic name

The name specified for a device or class characteristic was illegal.

CHARS = -14 - Illegal characteristic value size

The value size specified for a device or class characteristic was illegal.

CHART = -15 - Illegal characteristic type

An illegal type was specified for a device or class characteristic.

CHARV = -13 - Illegal characteristic value

The value specified for a device or class characteristic was illegal.

CHNNA = -146 - DMA channel not available

The requested DMA channel is not available.

CLSAD = -109 - Device class already defined

The device class being added to the system is already defined.

CPDNR = -199 - Child process did not respond

The child process being created by a svcIoRun system call did not indicate that it had finished loading its program within the time out period specified.

DATER = -46 - Data error

An unrecoverable data read error was encountered when attempting to input data from a device.

DATTR = -54 - Data truncated

Less data than expected was transferred to or from an I/O device. This error is only reported when this condition is due to an error condition. Reading less data than requested from a file because the end of file was reached does not produce this error, since this is a normal occurrence.

DEVER = -53 - Device error

An error was encountered when trying to access a device. This error generally indicates a failure of the device rather than the media, although this is not always true.

DEVFL = -35 - Device full

A mass storage device has no space available.

DEVIU = -31 - Device in use

A single user device was specified which was already in use by a different session process.

DFDEV = -37 - Different device for rename

An svcIoRename system service call specified a different device for the old and new file specifications.

DIRFL = -43 - Directory full

An attempt was made to create a new file when no additional space was available in the directory indicated (but not because the device was full). This error can only occur when attempting to create a new file in the root directory of an MS-DOS file system.

DIRNE = -44 - Directory not empty

An attempt was made to delete a non-empty directory.

DIRNF = -42 - Directory not found

A directory in the path specified for a file was not found.

DIRTD = -45 - Directory level too deep

A path to a file was specified with directories nested to more than the maximum directory nesting level. This limit is usually set to 7 directory levels.

DIVER = -201 - Divide error

The child process being created by an svcIoRun system call terminated during initialization because of a divide error (divide by 0 or divide overflow).

DKCHG = -70 - Disk changed

The media for a removable media disk has been changed since a file was opened.

DKRMV = -149 - Disk removed

A removable media disk was removed since a file was opened.

DLOCK = -80 - Deadlock condition

An operation was attempted which would likely result in a deadlock condition.

DOSMC = -154 - DOS memory allocation data corrupted

The DOS memory block headers were corrupted, making it impossible to allocate or deallocate memory in a DOS program.

DOSPB = -180 - Permanent DOS process is busy

An attempt was made to load a DOS program into a session's permanent DOS process when that process was not idle.

DPMIC = -207 - DPMI environment corrupted

The DPMI emulator was unable to perform the requested function because the per-process data which describes the DPMI environment for the process was not valid. This data is stored in user memory and can be corrupted by a misbehaved user program.

DQUOT = -96 - Disk quota exceeded

A process attempted to allocate more disk space than allowed by its disk quota.

DRFER = -90 - Directory block format error

When searching a directory for a file or for an empty slot, an illegal format was encountered.

DRRER = -91 - Directory block read error

When searching a directory for a file or for an empty slot, a read error occurred.

DRWER = -92 - Directory block write error

A write error occurred when attempting to update a directory block.

DTINT = -157 - Data transfer interrupted

A multi-block disk data transfer was terminated early because of a disk error. This error code is used internally by the disk optimization routines and should never be returned to a user program.

DUADF = -107 - Device unit already defined

The system device specified for an add unit function of the svcIoClass system service call was already defined.

EOF = -1 - End of file

An attempt was made to read data past the end of a file. Note that this error is generally returned only when there is no data at all available before the end of file. An attempt to read past the end of a file when some data is available results in less data being read than was requested.

ERROR = -168 - Untranslatable/general error

This error code is reported for any error which is not covered by another error code. It is also used when mapping an error from a remote system and there is no direct mapping to a more specific error code.

EVNRS = -197 - Event is not reserved

The svcSchRelEvent system call was issued to attempt to release an event which had not been reserved.

EVRES = -196 - Event is reserved

The svcSchResEvent system call was issued to attempt to reserve an event which was already reserved.

EVSET = -198 - Event is set

An attempt was made to set an event using the svcSchSetEvent system call which specified that the event should not be overwritten and the event was already set.

FBFER = -81 - FIB format error

An illegal format was encountered in a file information block (XOS file system).

FBPER = -82 - FIB pointer error

An invalid pointer was encountered in a file information block (XOS file system).

FBRER = -83 - FIB read error

An error occurred while reading a file information block (XOS file system).

FBWER = -84 - FIB write error

An error occurred while writing a file information block (XOS file system).

FILAD = -41 - File access denied

An attempt was made to access a file which the user is not privileged to access, which belongs to the user.

FILAF = -169 - File access failure

This error is returned for certain file access problems reported by foreign remote systems.

FILCF = -170 - File creation failure

This error is returned for certain file creation problems reported by foreign remote systems.

FILEX = -39 - File exists

An attempt was made to create a new file which specified that it should fail if the file existed and the file did exist, or an attempt was made to rename a file to the name of an existing file.

FILNF = -38 - File not found

An attempt was made to open a file which does not exist, or one which does exist that the user cannot access because it does not belong to the user or to a member of his user group.

FILRF = -172 - File rename failure

This error is reported for certain file rename problems reported by foreign remote systems.

FILXF = -171 - File extend failure

This error is reported for certain file allocation problems reported by foreign remote systems.

FSINC = -97 - File system is inconsistent

The file system was found to be internally inconsistent. This is generally a serious error which probably indicates that some data on the file system will not be accessible and that continued use of the file system will probably result in additional data loss. If this error occurs, the file system should be backed up immediately to recover as much data as possible and then it should be reformatted.

FTPER = -186 - FAT block pointer error

An invalid value was found in a FAT block when accessing a file on a DOS file structure.

FTRER = -187 - Error reading FAT block

An error occurred when reading a FAT block for a DOS file structure.

FTWER = -188 - Error writing FAT block

An error occurred when writing a FAT block for a DOS file structure

FUNC = -3 - Illegal function

A system service, which requires the specification of a function, was called with an illegal function specified.

FUNCM = -4 - Illegal function for current mode

The function specified in a system service call is illegal for the current mode of the device referenced.

HBFER = -85 - Home block format error

When attempting to mount a disk, an illegal format was found in the disk's home block (XOS file system) or boot block (DOS file system).

HBRER = -86 - Home block read error

When attempting to mount a disk, an error occurred when reading the disk's home block (XOS file system) or boot block (DOS file system).

IADEV = -69 - Illegal buffer address for device

The buffer specified for a data transfer had an illegal address. This generally occurs when a buffer which spans a page boundary is specified for a physical device transfer.

IATTR = -94 - Illegal file attribute change

An illegal file attribute change was specified, such as attempting to set or clear the directory attribute.

ICDEV = -68 - Illegal count for device

A transfer count was specified which was too large for the device.

IDEVC = -156 - Incorrect device class

The device class specified for the IOPAR_CLASS I/O parameter was incorrect.

IDFER = -47 - ID field error

An unrecoverable error was encountered when attempting to read the ID field of a disk.

IDREN = -159 - Invalid directory rename operation

An attempt was made to rename a directory into a directory which would result in an invalid directory tree.

IDSPC = -139 - Illegal destination file specification

The destination file specification for the svcIoDstName system call is not properly formed or is inconsistent with the source file specification and the search mask specified.

IFDEV = -67 - Illegal function for device

A function was specified for a device which is illegal for the device.

IIFF = -62 - Illegal image file format

An image file which was specified to be loaded for execution was not a properly formatted image file.

IIFRD = -63 - Illegal relocation data in image file

An image file which was specified to be loaded for execution contained improperly formatted relocation information.

IIFT = -61 - Illegal image file type

An image file which was specified to be loaded for execution was not of the proper type.

IINUM = -138 - Illegal interrupt number

An illegal interrupt number was specified by a device driver when attempting to initialize an interrupt vector.

ILLIN = -202 - Illegal instruction

The child process being created by an svcIoRun system call terminated during initialization because an illegal instruction was executed.

ILSEK = -78 - Illegal seek function

A seek operation was requested for a device which does not support seeks.

IMEMA = -144 - Illegal memory address

The memory address specified is illegal.

INCMO = -181 - Incomplete output operation

An output operation was not completed.

ININU -163 - Interrupt number in use

A device driver attempted to initialize an interrupt vector which was already in use.

INVST = -194 - Invalid segment type

An attempt was made to illegally move or otherwise modify a memory segment. This will occur if the segment was linked to an exec segment.

IOSAT = -158 - I/O saturation

An I/O operation was terminated because it was occurring at too high a rate. This generally indicates a defective device interface which is not clearing an interrupt request. The data transfer is terminated and the device is reset to keep it from hanging the rest of the system.

IPDIR = -192 - Illegal pointer in directory

An illegal value was found in the pointer in a directory to the first cluster of a file.

ISDIR = -76 - File is a directory

An operation was attempted on a file which is a directory, which is illegal for directories.

LASNA = -34 - Linear address space not available

The linear address space requested is not available.

LKEAL = -160 - LKE already loaded

The LKE being loaded is already loaded.

LOCK = -183 - File record lock violation

An attempt was made to lock a file record which was already locked by another user.

LSTER = -50 - Lost data error

Data was lost due to a device overrun or underrun.

MACFT = -24 - Memory address conflict

Memory is already allocated at the address which was specified for the allocation of memory. This error is most often associated with the svcMemChange and svcMemMap system service calls.

MAERR = -25 - Memory allocation error

User modifiable data which is required for memory allocation is inconsistent. This error can only be returned by the routines which emulate DOS memory allocation in virtual mode.

MATH = -257 - Math function error

An error was detected in one of the math library routines.

MEMLX = -208 - Memory limit exceeded

An attempt was made to allocate more memory than allowed for the process.

MPILK = -175 - Memory page is locked

An attempt was made to deallocate a memory page which was locked by an active I/O request.

MSNPR = -193 - Msect is not private

A attempt was made to convert a memory section that was not a simple private section to a shared section.

NACT = -72 - Device not active

The specified I/O request was not active on the device.

NCCLR = -126 = - Network connection cleared

The connection to a remote system was cleared by the remote system.

NCLST = -124 - Network connection lost

The connection to a remote system was terminated unexpectedly.

NCOMP = -142 - Not compatible

The requested operation was not compatible with the current state of the device, network, or system.

NCONG = -120 - Network congestion

Network communication failed because of excessive congestion in the network.

NCRFS = -127 - Network connection refused

An attempt to establish a connection with a remote system was refused by the remote system.

NDOSD = -155 - No DOS I/O data block available

A DOS system call could not be performed because no DOS I/O data block was available. These are dynamically allocated 256 byte blocks used to store parameters for certain DOS system calls.

NEMA = -23 - Not enough memory available

There is not enough free memory available in the system to satisfy a request for the allocation of additional memory, or the request would cause the amount allocated to the process to exceed the amount the process is allowed. This error may be returned by any system service call which allo-

cates memory to provide temporary workspace as well as by the memory system service calls.

NHSTA = -125 - Network host not available

The default network host for an XOS system configured as a workstation was not available.

NILAD = -116 - Illegal network address

An illegal network address was specified. This generally means that the external network complained about the format of the address.

NILPC = -118 - Illegal network protocol type

An illegal network protocol type was specified.

NILPR = -115 - Illegal network port number

An illegal network port number was specified.

NILRF = -117 - Illegal request format

This is a general error which means that the external network rejected some request because it was not formatted correctly.

NIYT = -256 - Not implemented yet

The operation attempted is not implemented in the current version of XOS.

NLKNA = -195 - Network link not available

An attempt was made to do I/O using a network interface which had been disabled.

NMBTS = -137 - Name buffer is too small

The name buffer (as specified with the IOPAR_FILSPEC I/O parameter) is not large enough to contain at least one file name when a repeated operation was specified.

NNAVL = -128 - Network not available

The connection to the network is not available.

NNOPC = -191 - No network protocol specified

A attempt was made to do I/O on a network device which requires specification of a underlying protocol and no such protocol was specified for the

device. An example of this would be to attempt output on an Ethernet IP device for which no Ethertype value had been specified.

NNSER = -134 - Network name server error

A Domain Name Server indicated an unspecified error.

NNSNA = -131 - Network name server not available

A remote system cannot be accessed because no Domain Name Server could be found to resolve its Domain Name.

NNSNC = -129 - Network name server not capable

A Domain Name Server refused a request with an indication that it was not capable of performing the request.

NNSRF = -130 - Network name server refused request

A Domain Name Server refused a request. This often means that the system is not privileged enough to make the request.

NNSRQ = -132 - Network name server bad format request

A Domain Name Server refused a request with an indication that the request had a bad format.

NNSRS = -133 - Network name server bad format response

The response received from a Domain Name Server had a bad format.

NOBUF = -27 - No system buffer available

This error is reported if a buffer cannot be obtained for the system internal buffer pool. This error should not occur. If it does, it probably means that the memory on a small system is heavily overcommitted.

NODCB = -26 - No disk cache buffer available

This error is reported if no disk cache buffer was available for use when needed. Disk cache buffers are large buffers used internally by the kernel for many purposes, but primarily for buffering disk data. This error should not occur. If it does, the number of disk buffers specified in the system startup file should be increased.

NOERR - 0 - No error indicated

An error code value of 0 is not used by the system. Such a value usually indicates an error in the user program's error handling routines, usually

that a normal return from a system service or library routine was taken to be an error.

NOIN = -59 - Input not allowed

Input was attempted from a device that was opened without the O$IN bit set or from a device which does not support input.

NOMEM = -140 - Memory not allocated

The memory section specified does not exist.

NOOUT = -58 - Output not allowed

Output was attempted to a device that was opened without the O$OUT bit set or to a device which does not support output.

NOPAP = -143 - Printer is out of paper

A printer reported that it was out of paper.

NORSP = -55 - Device did not respond

A device did not complete the requested operation in a reasonable amount of time.

NOSAD = -65 - No starting address specified in image file

An image file was specified to be loaded for execution which did not contain a starting address specification.

NOSTK = -66 - No stack specified in image file

An image file was specified to be loaded for execution which did not contain a stack address specification.

NPCIU = -119 - Network protocol type in use

The network protocol type specified was already in use by the interface.

NPERR = -112 - Network protocol error

This error reports a general network protocol error which does not fall into a more specific error category.

NPRIU = -114 - Network port in use

The network port specified is in use.

NPRNO = -113 - Network port not open

The network port specified is not open.

NRTER = -121 - Network routing error

The external network was unable to route a message to its destination.

NRTNA = -136 - Network router not available

No router is available when one is required to send a message to a remote system which is on a different sub-net.

NSCLS = -108 - No such device class

The device class specified for the svcIoClass system service call did not exist.

NSDEV = -30 - No such device

A device which does not exist in the system was specified or a device which does exist but which the process is not privileged to use was specified.

NSEGA = -22 - No segment available

An attempt was made to create a segment when the process already has the maximum allowable number of segments.

NSLP = -182 - Not a session level process

A process which was not a session level process was specified for a function which required a session level process.

NSNOD = -122 - No such network node

The remote network node specified did not respond or is not known to the external network.

NSP = -20 - No such process

The requested process does not exist.

NSTYP = -145 - No such device type

The device type specified for the add unit function was not valid for the device class to which a unit was being added.

NTDEF = -98 - Not defined

The logical name or environment string specified was not defined.

NTDIR = -75 - File is not a directory

A directory specified in a path for a file is not a directory.

NTDSK = -95 - Device is not a disk

A non-disk device was specified for a function which requires a disk.

NTFIL = -93 - Device is not file structured

A device which is not file structured was specified for an operation which requires a file structured device.

NTIMP = -167 - Not implemented

The requested function is not implemented.

NTLCL = -179 - Not local

An operation which is valid only for a local device was requested for a remote device.

NTLNG = -101 - Name is too long

The file, directory, or device name specified was too long.

NTRDY = -74 - Device not ready

The requested I/O operation could not be performed because the device was not ready (printer was off line, floppy disk was not inserted, etc.).

NTTIM = -123 - Network time-out

A response to a network message was not received in a reasonable length of time.

NTTRM = -77 - Device is not a terminal

A terminal specific operation was attempted on a device which was not a terminal.

NWPA = -204 - No watchpoint available

An attempt was made to set a hardware watchpoint when all available watchpoints were already in use. The 80386/80486 processors support a maximum of four hardware watchpoints.

NXERR = -135 - Network transmit error

An error occurred when attempting to transmit a network message.

PARMF = -10 - Illegal parameter function

An illegal function was specified for an I/O parameter.

PARMI = -6 - Illegal parameter index

The index value specified for an I/O parameter was illegal.

PARMM = -11 - Required parameter missing

A required I/O parameter was not specified.

PARMS = -8 - Illegal parameter size

The value size specified for an I/O parameter was illegal.

PARMT = -9 - Illegal parameter type

An illegal type was specified for an I/O parameter.

PARMV = -7 - Illegal parameter value

The value specified for an I/O parameter was illegal.

PDADF = -106 - Physical device already defined

The physical device specified for an add unit function of the svcIoClass system service call was already defined as a system device.

PDNAV = -105 - Physical device not available

The physical device specified for an add unit function of the svcIoClass system service call could not be found.

PDTYP = -104 - Physical device type incorrect

The physical device type was invalid for the function requested.

PRIV = -21 - Insufficient privilege

An attempt was made to perform an operation which required a privilege which the process did not possess.

RANGE = -258 - Math function argument out of range

An argument to one of the math library routines was out of range.

RELTR = -64 - Relocation truncation in image file

An image file which was specified to be loaded for execution contained relocation information which resulted in a relocated value being truncated.

RNFER = -49 - Record not found error

The requested record was not found on the indicated track on a disk.

SBFER = -87 - Storage allocation block format error

When attempting to mount a disk, a format error was found in a storage allocation block (XOS file system) or file allocation table (DOS file system).

SBRER = -88 - Storage allocation block read error

When attempting to mount a disk, a read error occurred while reading a storage allocation block (XOS file system) or file allocation table (DOS file system).

SBWER = -89 - Storage allocation block write error

When attempting to allocate space or close a file, a write error occurred while writing a storage allocation block (XOS file system) or file allocation table (DOS file system) to the disk. If this error occurs, serious consideration should be given to backing up the disk involved immediately, since it probably indicates a serious problem which could compromise file integrity.

SEKER = -48 - Seek error

An unrecoverable seek error was encountered when attempting to position a disk to the desired track.

STKER = -200 - Stack error

The child process being created by an svcIoRun system call terminated during initialization because of a memory fault when accessing the user stack.

SVC = -2 - Illegal SVC function

An illegal system service function was specified.

TMALM = -206 - Too many alarms for process

An attempt was made to create more alarms than allowed for a process.

TMDDV = -178 - Too many device units for device

There are too many device units declared for this device.

TMDVC = -111 - Too many devices open for device class

There are no more allocatable devices available in the device class.

TMDVP = -36 - Too many devices open for process

An attempt was made by a single process to open more devices than the process device limit.

TMIOM = -173 - Too many I/O requests for memory page

A I/O operation would result in a single memory page being locked by more than 255 different requests. It is highly unlikely that this error will occur in normal system use.

TMIOP = -174 - Too many I/O request pointers

More than 8 contiguous groups of memory pages needed to be locked in memory for an I/O operation. A memory referenced by an I/O operation must be locked, including file specification strings, data buffers, I/O parameter lists, and string values of I/O parameters.

TMIOQ = -176 - Too many I/O requests queued

An svcIoQueue system call would exceed the request queue limit for the I/O device.

TMPSS = -103 - Too many processes or shared segments in system

A attempt to create a child process was made when the system already contained the maximum allowed number of processes.

TMRNC = -148 - Too many requests for network connection

A function was requested which exceeded the multiplexing capacity of a network connection for the protocol being used.

TMRQB = -189 - Too many requests for buffer

Too many requests were made for access to a disk cache buffer. This error is extremely unlikely, since the maximum number of requests is 65534 for each buffer.

TMUDV = -177 - Too many users for device

A device has been opened too many times.

TMUSR = -102 - Too many users

An attempt was made to create a new user session which would exceed the maximum number of user sessions allowed.

TRMNA = -210 - Terminal is not attached

An I/O operation was attempted on a terminal device which was not attached to a physical serial port or console display.

UNXSI = -203 - Unexpected signal.

The child process being created by an svcIoRun system call terminated during initialization because of an unexpected signal.

VALUE = -5 - Illegal value

An illegal value was given as an argument to a system call.

VECNS = -209 - Signal vector not set up

An attempt was made to request some action (such as setting up an alarm) which required that a vector be set up and the vector was not set up.

WLDNA = -100 - Wild-card name not allowed

A wild-card file name was given where a fully specified file name was required.

WPRER = -52 - Write protect error

An attempt was made to write to a write protected device.

WRTER = -51 - Write fault error

A detectable error occurred while writing to a device.

XFRBK = -110 - Transfer blocked

A data transfer could not be completed.

Numerical List of System Error Codes

Value	Name	Description
0	NOERR	Normal return
-1	EOF	End of file
-2	SVC	Illegal SVC function
-3	FUNC	Illegal function
-4	FUNCM	Illegal function for current mode
-5	VALUE	Illegal value
-6	PARMI	Illegal parameter index
-7	PARMV	Illegal parameter value
-8	PARMS	Illegal parameter value size
-9	PARMT	Illegal parameter type
-10	PARMF	Illegal parameter function
-11	PARMM	Required parameter missing
-12	CHARN	Illegal characteristic name
-13	CHARV	Illegal characteristic value
-14	CHARS	Illegal characteristic value size
-15	CHART	Illegal characteristic type
-16	CHARF	Illegal characteristic function
-17	CHARM	Required characteristic missing
-18	BDNAM	Bad process name
-19	BDPID	Bad process ID
-20	NSP	No such process
-21	PRIV	Not enough privilege
-22	NSEGA	No segment available
-23	NEMA	Not enough memory available
-24	MACFT	Memory allocation conflict
-25	MAERR	Memory allocation error
-26	NODCB	No disk cache buffer available

Value	Name	Description
-27	NOBUF	No system buffer available
-28	ACT	Device is active
-29	BDSPC	Bad device or file specification
-30	NSDEV	No such device
-31	DEVIU	Device in use
-32	DEVIO	Device is open
-33	DEVNO	Device not open
-34	LASNA	Linear address space not available
-35	DEVFL	Device is full
-36	TMDVP	Too many devices open for process
-37	DFDEV	Different device for rename
-38	FILNF	File not found
-39	FILEX	File exists
-40	BUSY	File or device is busy
-41	FILAD	File access denied
-42	DIRNF	Directory not found
-43	DIRFL	Directory full
-44	DIRNE	Directory not empty
-45	DIRTD	Directory level too deep
-46	DATER	Data error
-47	IDFER	ID field error
-48	SEKER	Seek error
-49	RNFER	Record not found error
-50	LSTER	Lost data error
-51	WRTER	Write fault error
-52	WPRER	Write protect error
-53	DEVER	Device error
-54	DATTR	Data truncated
-55	NORSP	Device did not respond
-56	BDDBK	Bad disk block number

Value	Name	Description
-57	BDDVH	Bad device handle
-58	NOOUT	Output not allowed
-59	NOIN	Input not allowed
-60	ADRER	Address error
-61	IIFT	Illegal image file type
-62	IIFF	Illegal image file format
-63	IIFRD	Illegal relocation data in image file
-64	RELTR	Relocation truncation in image file
-65	NOSAD	No starting address specified in image file
-66	NOSTK	No stack specified in image file
-67	IFDEV	Illegal function for device
-68	ICDEV	Illegal count for device
-69	IADEV	Illegal buffer address for device
-70	DKCHG	Disk changed
-71	RTOBG	Record too big
-72	NACT	Device not active
-73	FMTER	Format error
-74	NTRDY	Device not ready
-75	NTDIR	File is not a directory
-76	ISDIR	File is a directory
-77	NTTRM	Device is not a terminal
-78	ILSEK	Illegal seek function
-79	BPIPE	Pipe error
-80	DLOCK	Deadlock error
-81	FBFER	FIB format error
-82	FBPER	FIB pointer error
-83	FBRER	FIB read error
-84	FBWER	FIB write error
-85	HMFER	HOM format error
-86	HMRER	HOM read error

Value	Name	Description
-87	SBFER	SAT format error
-88	SBRER	SAT read error
-89	SBWER	SAT write error
-90	DRFER	Directory format error
-91	DRRER	Directory read error
-92	DRWER	Directory write error
-93	NTFIL	Not a file structured device
-94	IATTR	Illegal file attribute change
-95	NTDSK	Device is not a disk
-96	DQUOT	Disk quota exceeded
-97	FSINC	File system is inconsistent
-98	NTDEF	Not defined
-99	BDLNM	Bad logical name
-100	WLDNA	Wild card name not allowed
-101	NTLNG	Name is too long
-102	TMUSR	Too many users
-103	TMPSS	Too many processes or shared segments in system
-104	PDTYP	Physical device type incorrect
-105	PDNAV	Physical device not available
-106	PDADF	Physical device already defined
-107	DUADF	Device unit already defined
-108	NSCLS	No such device class
-109	CLSAD	Device class already defined
-110	XFRBK	Data transfer blocked
-111	TMDVC	Too many devices open for device class
-112	NPERR	Network protocol error
-113	NPRNO	Network port not open
-114	NPRIU	Network port in use
-115	NILPR	Illegal port number
-116	NILAD	Illegal network address

Value	Name	Description
-117	NILRF	Illegal request format
-118	NILPC	Illegal network protocol type
-119	NPCIU	Network protocol type in use
-120	NCONG	Network congestion
-121	NRTER	Network routing error
-122	NSNOD	No such network node
-123	NTTIM	Network time out
-124	NCLST	Network connection lost
-125	NHSNA	Network host not available
-126	NCCLR	Network connection cleared
-127	NCRFS	Network connection refused
-128	NNAVL	Network not available
-129	NNSNC	Network name server not capable
-130	NNSRF	Network name server refused request
-131	NNSNA	Network name server not available
-132	NNSRQ	Network name server bad format request
-133	NNSRS	Network name server bad format response
-134	NNSER	Network name server error
-135	NXERR	Network transmit error
-136	NRTNA	Network router not available
-137	NMBTS	Name buffer is too small
-138	IINUM	Illegal interrupt number
-139	IDSPC	Illegal destination file specification
-140	NOMEM	No memory allocated
-141	ALDEF	Already allocated
-142	NCOMP	Not compatible
-143	NOPAP	Printer is out of paper
-144	IMEMA	Illegal memory address
-145	NSTYP	No such device type
-146	CHNNA	DMA channel not available

Value	Name	Description
-148	TMRNC	Too many requests for network connection
-149	DKRMV	Disk removed
-150	ABORT	I/O operation aborted
-151	CANCL	I/O operation cancelled
-154	DOSMC	DOS memory allocation data corrupted
-155	NDOSD	No DOS I/O data block available
-156	IDEVC	Incorrect device class
-157	DTINT	Data transfer interrupted
-158	IOSAT	I/O saturation
-159	IDREN	Invalid directory rename
-160	LKEAL	LKE already loaded
-161	CDAAD	LKE common data area already defined
-162	CDAND	LKE common data area not defined
-163	ININU	Interrupt number in use
-167	NTIMP	Not implemented
-168	ERROR	Unspecified general error
-169	FILAF	Cannot access file
-170	FILCF	Cannot create file
-171	FILXF	Cannot extend file
-172	FILRF	Cannot rename file
-173	TMIOM	Too many I/O requests for memory page
-174	TMIOP	Too many I/O request pointers
-175	MPILK	Memory page is locked
-176	TMIOQ	Too many I/O requests queued
-177	TMUDV	Too many users for device
-178	TMDDV	Too many device units for device
-179	NTLCL	Not local
-180	DOSPB	Permanent DOS process is busy
-181	INCMO	Incomplete output operation
-182	NSLP	Not a session level process

Value	Name	Description
-183	LOCK	File record lock violation
-184	CAASP	Close action already specified
-185	CAERR	Close action error
-186	FTPER	FAT block pointer error
-187	FTRER	Error reading FAT block
-188	FTWER	Error writing FAT block
-189	TMRQB	Too many requests for buffer
-190	CCMSS	Cannot change memory section size
-191	NNOPC	No network protocol specified
-192	IPDIR	Illegal pointer in directory
-193	MSNPR	Msect is not private
-194	INVST	Invalid segment type
-195	NLKNA	Network link not available
-196	EVRES	Event is reserved
-197	EVNRS	Event is not reserved
-198	EVSET	Event is set
-199	CPDNR	Child process did not respond
-200	STKER	Stack error
-201	DIVER	Divide error
-202	ILLIN	Illegal instruction
-203	UNXSI	Unexpected software interrupt
-204	NWPA	No watchpoint available
-205	BDALM	Bad alarm handle
-206	TMALM	Too many alarms for process
-207	DPMIC	DPMI environment corrupted
-208	MEMLX	Memory limit exceeded
-209	VECNS	Signal vector not set up
-210	TRMNA	Terminal is not attached
-256	NIYT	Not implemented yet
-257	MATH	Math function error

Value	Name	Description
-258	RANGE	Math function argument out of range

INDEX

A

B

C

D

E

F

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

X