FlexOS™
Programmer's Guide

Version 1.3

1073-2023-001

First Edition: November 1986

The kernel is based on an event-driven dispatcher that does priority-driven scheduling. Time slicing is done by a timer event that occurs once per TICK, typically every 16 to 20 milliseconds (implementation dependent). Scheduling of equal priority processes is done in a round-robin fashion.

Pipe File System

FlexOS performs process communication and synchronization through named pipes. These in-memory files are available to pass data from one process to another or to synchronize activities when acting as semaphores.

Console File System

The FlexOS console system provides dedicated functions designed specifically for the fast manipulation of bit-mapped and character-oriented displays. A single call can copy or modify a screen region ranging in size from a single character cell to the entire screen. These functions give you a consistent, hardware-independent interface to the computer's interactive devices without sacrificing program portability.

The console system also provides window management facilities that allow applications to create and manage multiple virtual consoles.

Intended Audience and Manual Organization

This manual (hereinafter referred to as the Programmer's Guide) is written for the programmer whose goal is to write applications and utilities to run under the FlexOS operating system. The Programmer's Guide anticipates, but does not require, a working knowledge of the C programming language.

The Programmer's Guide is organized as follows:

Section 1	Terms and conventions used in this manual; file system characteristics; summary of Supervisor calls and tables.
Section 2	Disk Resource Manager
Section 3	Console Resource Manager
Section 4	Pipe Manager
Section 5	Process Management
Section 6	Miscellaneous Device Management
Section 7	Supervisor call reference
Section 8	system table reference
Appendix A	FlexOS character codes
Appendix B	System return and error codes
Appendix C	FlexOS country codes

The FlexOS Documentation Set

The Programmer's Guide is one of several manuals in the FlexOS documentation set. The other documents are:

- FlexOS User's Guide: The user's reference for FlexOS operation. The User's Guide describes the command shell, advanced FlexOS concepts, and command files. It also provides an overview of system manager functions.
- FlexOS System Guide: The guide to FlexOS system implementation for an original equipment manufacturer or driver writer. Information presented in this guide includes driver and supervisor interfaces, FlexOS's driver services, and how to construct a boot loader.
- FlexOS Supplements: Microprocessor-dependent supplements to the Programmer's Guide and the FlexOS System Guide.
- FlexOS Programmer's Utilities Guides: The reference to FlexOS assembly language programming tools. There is a separate utilities guide for each microprocessor supported by FlexOS.

Foreword

FlexOS™ is a real-time, multitasking operating system designed for single-user and multiuser microcomputer systems. The programming interface to FlexOS is CPU- and peripheral-independent so you can develop programs that are portable between machines with different components and processors.

FlexOS Features

FlexOS provides comprehensive facilities for process, file, console, and device management. The following list summarizes these facilities:

- Process Management
  - FlexOS process execution
  - Independent, modifiable process environments
  - Asynchronous events and software interrupt handling
  - Interprocess communication and synchronization
- Disk System
  - PC DOS compatible, with hierarchical file directories
  - Shared file system with file and record locking
  - File system protection based on file and directory ownership
  - User and group file ownership
  - Removable media support
- Real-time processing
  - Support for real-time data acquisition and communications
  - Primitives for real-time process control and other real-time applications

- Console
  - Escape sequence decoding
  - Standard character and bit-mapped screen interface
  - Standard 16- and 8-bit keyboard interfaces including function keys, numerical keypad and multikeyed characters
  - Virtual console management primitives that include window support
- International considerations
  - Support for 16-bit foreign languages
  - Customization of console messages including country codes
- Memory mapping and protection
- Dynamically loadable device drivers
- CPU-independent programming

Disk File System

The FlexOS disk file system is designed for multiuser and networked microcomputer systems. Hierarchical, shared disk files allow for the large, shared data bases commonly used with professional work stations. The record and file locking mechanisms, along with security through ownership, allow integrity and protection of data.

FlexOS's disk file system is designed to protect against file destruction from power interruptions or accidental system resets. A utility is provided that reconstructs file directory entries and allocation tables from the data area of the disk.

The FlexOS file system distinguishes removable from permanent media and recognizes removable media that have open door interrupts.

Real-time Kernel

The kernel provides multiuser and multitasking environments that allow both real-time control applications and integrated office environments on the same CPU.

The Programmer's Guide, User's Guide, and System Guide are generic in that they are appropriate for FlexOS systems based on any supported microprocessor. Before developing programs, you should become familiar with the sections of the FlexOS supplements that describe microprocessor-dependent distinctions and differences of operation. In most cases, the points of difference are noted in the appropriate sections of this manual. However, not all information is cross-referenced.

Contents

1 TERMS, CONCEPTS, AND CONVENTIONS

2 DISK FILE MANAGEMENT

5 PROCESS MANAGEMENT

6 MISCELLANEOUS RESOURCE MANAGER

7 SUPERVISOR CALL DESCRIPTIONS

Figures

Tables

Listings

Terms, Concepts, and Conventions

This section defines the terms, concepts, and conventions used in this manual and describes the file system characteristics and FlexOS architecture.

1.1 C Language Conventions

Table 1-1 lists the data-type definitions used to promote C portability and reduce compiler differences.

Table 1-1. Standard Data-type Definitions

Data Type	Definition
BYTE	Signed 8-bit value
BOOLEAN	Byte with one of two values: true/false
WORD	Signed, 16-bit value
UWORD	Unsigned 16-bit value
LONG	Signed 32-bit value
STRUCT	Named sequence (structure) of variables

1.2 Supervisor Calls

The functions performed by FlexOS are referred to as Supervisor calls (SVCs). SVCs provide file, console, event, process control, and device I/O and management services. Table 1-2 lists the SVCs according to their purpose (asterisks indicate those SVCs that can be called asynchronously).

Table 1-2. Supervisor Call Summary

Purpose	Call	Action
File Management		
	DEFINE	Define logical name for a path
	CREATE	Create a file
	DELETE	Delete a file
	OPEN	Open a disk file
	CLOSE	Close a disk file
	READ*	Read from a file
	WRITE*	Write to a file
	SEEK	Modify or obtain current file pointer
	LOCK*	Lock/Unlock an area of a disk file
	RENAME	Rename or move a file
Console Management		
	KCTRL	Obtain keyboard and mouse ownership
	ORDER	Order windows on parent screen
	XLAT	Specify keystroke translation
	GIVE	Give keyboard and mouse to child proces
	COPY	Copy one screen rectangle to another
	ALTER	Alter a screen rectangle
	RWAIT*	Wait for mouse to enter/leave a rectangle
	BWAIT	Wait for mouse button state change
Event Management		
	CANCEL	Cancel asynchronous events
	WAIT	Wait for multiple events
	STATUS	Get status of asynchronous events
	RETURN	Get return code of completed event

Table 1-2. (Continued)

Purpose	Call	Action
Real Time and Process Management		
	TIMER*	Set and wait for timer interrupt
	ABORT*	Abort specified process
	COMMAND*	Perform command
	EXCEPTION	Set software interrupts on exceptions
	MALLOC	Allocate memory to heap
	MFREE	Free memory from heap
	EXIT	Terminate with return code
	ENABLE	Enable software interrupts
	DISABLE	Disable software interrupts
	SWIRET	Return from software interrupt
	CONTROL*	Control a process for debugging
	OVERLAY	Load overlay from command file
Device Management		
	SPECIAL*	Perform special device function
	DEVLOCK	Lock or unlock device for user/group
	INSTALL	Install, replace and associate drivers
Table Management		
	GET	Get a table
	SET	Set table values
	LOOKUP	Scan and retrieve tables

* Your program can call these SVCs asynchronously.

Table 1-3 lists the SVCs by their number.

Table 1-3. Supervisor Calls by Number

Number	Call	Number	Call
0	F_GET	21	Reserved
1	F_SET	22	F_GIVE
2	F_LOOKUP	23	Reserved
3	F_CREATE	24	F_TIMER
4	F_DELETE	25	F_EXIT
5	F_OPEN	26	F_ABORT
6	F_CLOSE	27	F_CANCEL
7	F_READ	28	F_WAIT
8	F_WRITE	29	F_STATUS
9	F_SPECIAL	30	F_RETURN
10	F_RENAME	31	F_EXCEPTION
11	F_DEFINE	32	F_ENABLE
12	F_DEVLOCK	33	F_DISABLE
13	F_INSTALL	34	F_SWIRET
14	F_LOCK	35	F_MALLOC
15	F_COPY	36	F_MFREE
16	F_ALTER	37	F_OVERLAY
17	F_XLAT	38	F_COMMAND
18	Reserved	39	F_CONTROL
19	F_KCTRL	40	Reserved
20	F_ORDER	41	F_SEEK

1.2.1 Calling Conventions

FlexOS Supervisor calls are made by invoking the FlexOS entry point. The entry point takes two arguments and returns a value, as follows:

Arguments: a SVC 16-bit number
a parameter block pointer or value, 32-bit

Return: a 32-bit value

See the processor-specific supplement for the actual entry mechanism and registers used.

You can call FlexOS independent of a processor by calling the _osif function supplied with the operating system. The _osif function has two arguments: the SVC number (16 bits) and, depending on the SVC, a 32-bit parameter block address or parameter value. The C language definition of the _osif function is:

```
WORD    SVCno;
LONG    parm;
LONG    ret;

ret = _ _osif(SVCno,parm);
```

The _osif function returns the return code in registers, according to the convention of the language processor used to create the program.

You can also call FlexOS independent of the processor by using the standard FlexOS SVC library supplied with the language processor available for FlexOS. Each of these library functions builds a parameter block for the corresponding SVC and calls FlexOS. This high-level interface allows the description of FlexOS supervisor calls in processor-independent and register convention independent methods.

1.2.2 Data Structure Representation

Throughout this manual, data structures are represented as shown in Figure 1-1. Listing 1-1 contains the corresponding code representation. Byte and word order are critical when using these structures.

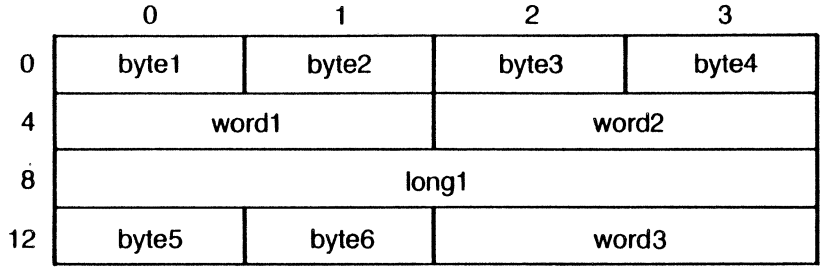


Figure 1-1. Data Structure Diagram

Listing 1-1. Data Structure Representation

```
STRUCT thisstruct
{
        BYTE        byte1;          /* byte offset = 0              */
        BYTE        byte2;          /* byte offset = 1              */
        BYTE        byte3;          /* byte offset = 2              */
        BYTE        byte4;          /* byte offset = 3              */
        WORD        word1;          /* byte offset = 4              */
        WORD        word2;          /* byte offset = 6              */
        LONG        long1;          /* byte offset = 8              */
        BYTE        byte5;          /* byte offset = 12             */
        BYTE        byte6;          /* byte offset = 13             */
        WORD        word3;          /* byte offset = 14             */
 );                             /*          length = 16             */
```

1.2.3 Synchronous and Asynchronous SVCs

All SVCs have a synchronous form. This means the call does not return until the operating system completes the event--for example, reads a record from the disk, writes a string to the console, or opens a file. Some SVCs also have an asynchronous form. These calls return a value immediately which uniquely identifies the event requested. Program operation can then proceed independently of the event.

Synchronous and asynchronous SVCs take the following forms:

```
ret = s_funcname(parm1,parm2,...,parmN);
emask = e_funcname(swi,parm1,parm2,...,parmN);
```

SVC names starting with "s_" are synchronous SVCs. The ret value is the completion code for the event.

SVC names starting with "e_" are asynchronous. The emask value is the event mask which uniquely identifies the event. The completion code for asynchronous calls is acquired with the RETURN SVC.

The contents of the parameter block block built from an SVC call are different, depending on the SVC.

The individual parameters are always provided in the form shown in Figure 1-2. The largest parameter block is 28 bytes long.

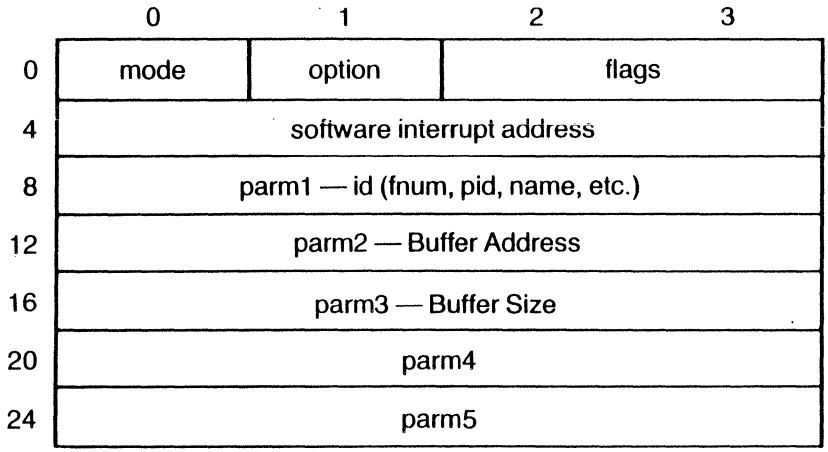


Figure 1-2. SVC Parameter Block

The Supervisor checks the mode to determine if the SVC is synchronous or asynchronous. A mode value of 0 indicates a synchronous SVC; a mode value of 1 indicates an asynchronous SVC. A parameter error is returned if the mode specified is not 0 or 1. The option and flags values select options unique to each SVC.

The software interrupt address is a pointer to an optional software interrupt routine available with asynchronous SVCs. FlexOS forces the calling process to jump to this routine when the asynchronous event completes.

The ID parameter uniquely identifies the object of the call. For example, for a WRITE call the object is a disk file, console, printer, or other peripheral device. The ID value in this case is a 32-bit file number.

The buffer is used to store data for transfer to or from the object. FlexOS checks the address and size values to ensure there are no memory boundary violations.

Many fields are marked with a 0 (zero) in the individual SVC calls. These fields must be set to zero to be compatible with future releases of FlexOS. An error is returned if they are not zero.

1.2.4 Return Codes

The return code for synchronous and asynchronous SVCs is always a LONG value (32 bits). A zero or positive value (high order bit is off) indicates a successful operation. SVCs not returning any particular value, such as a file number or a process ID, return a NULL (0) value to indicate success. For synchronous SVCs, the return value is the completion code. For asynchronous SVCs, the return value is the event mask, not the results of the operation.

A negative return code (high order bit is on) for synchronous and asynchronous SVCs indicates that an error occurred. The high order word contains the module or device code and the low order word contains the error type code. See Appendix B for the error code descriptions. These codes also apply to the asynchronous call's completion code.

1.2.5 Asynchronous Supervisor Calls

Asynchronous Supervisor calls allow a program to process multiple events simultaneously. Table 1-4 lists the Supervisor calls with asynchronous forms.

Table 1-4. Asynchronous SVCs

SVC	Purpose
READ*	Read from a file.
WRITE*	Write to a file.
LOCK	Lock/unlock an area of a disk file.
TIMER	Wait for a time period to expire.
COMMAND	Create a process.
SPECIAL	Perform a special device function.
CONTROL	Control a process with another process.
ABORT	Wait for a process to terminate.
BWAIT	Wait for a mouse button state to occur.
RWAIT	Wait for the mouse to enter or exit a region.

* You cannot read or write a disk file asynchronously. You can only use asynchronous READ and WRITE on console files, pipes, printers, and designated special devices.

Each process can have up to 31 on-going events; each identified by a single bit set in the event mask. The event mask is relevant to the following SVCs:

- WAIT to synchronize on one or more asynchronous events
- RETURN to acquire an event's completion code
- STATUS to indicate completed events
- CANCEL to cancel an event

FlexOS provides two mechanisms sensitive to event completion. You can suspend program execution until an event or one of several events completes or you can execute the software interrupt routine (swi) when the event completes.

Waiting on Events

Use the WAIT SVC to synchronize program operation on the completion of an event.

The event or events to wait on are specified in the WAIT argument and the call returns when any of the specified events completes. The event completed is indicated in the return code. While the process is waiting, it is removed from the dispatcher's ready list and minimizes the CPU load.

To get the completion code for an asynchronous event, use the RETURN SVC. RETURN use is limited to asynchronous events that do not have a software interrupt (swi). (The completion code is passed to the software interrupt and hence is not available to the process.) For asynchronous events without a swi, use the WAIT return code as RETURN's event mask. The event mask bit is not reset until RETURN has been called.

The STATUS SVC is also useful to determine completed events. STATUS places a heavy burden on the CPU and excessive use impacts program performance. You specify the events you want considered in STATUS's argument, and the call returns with the bit of all completed events set.

Interrupting upon Event Completion

Each asynchronous SVC allows a pointer to a swi so program code can be executed asynchronously when an event occurs. When the event completes, FlexOS preserves the stack pointers and proceeds with the swi.

Two values are passed to the swi, the completed event's mask and its completion code. Both are LONG values. A swi has the following C form:

```
swi_routine(emask,compcode);
    LONG emask;/*mask of completed event*/
    LONG compcode;/*event's completion code*/
{
/*interrupt routine*/

    s_swiret(0L);/*swi exit call--return to main program*/
}
```

FlexOS clears the event mask when the swi is called; do not call RETURN to reset the bit.

You must use the SWIRET SVC to exit the swi. It gives you two options: return to the program at the point of interruption or assume the process identity from the main program. For both options the stack pointer is restored to its condition when the program was interrupted.

When you have the swi assume the process identity, you can force a return to the main program or not return at all. If you force the return to the main program, the stack condition is unknown. Consider the use of a routine that returns the stack to a known place and jump to this routine from the swi.

When you have the swi assume the process identity, use EXIT to terminate the process. Do not call EXIT until after you have called SWIRET, however.

Note: The asynchronous form of ABORT is typically used as a mechanism to preserve a process when it is user-aborted. For example, consider a menu-driven program where the user enters a control-C to abort a menu-selection. To trap the control-C and return to a menu within the program rather than the operating system, you would use the asynchronous ABORT and a swi to force the return to the program. To abort the menu program entirely, the user would have to enter two control-Cs.

To establish critical regions where a swi cannot interrupt program execution, use the DISABLE SVC. No swi is executed while DISABLE is active, however, FlexOS does log the completion of asynchronous events during this time. Use the ENABLE SVC to end the DISABLE mode. All event swis impeded while DISABLE was active are executed after ENABLE is called.

1.3 File Specifications

A file is a logical construct applicable to the range of devices and functional units managed by FlexOS. FlexOS uses files to store or display information (disk files, pipes, console files, device files), get data input (keyboard and device files), and control access (zero length pipes).

Every file is specified by a path. A path consists of the following elements:

node::	network node name
device:	logical device name
\	root directory
directory\	subdirectory name
filename	file name and extension

These elements are always entered in the following sequence:

node::device:\directory\...directory\filename

If you do not specify a node, device, or directory, the current disk directory is assumed.

The node and device names can be one to eight alphanumeric characters. A directory name can have one to eight alphanumeric characters and always has the DIR extension. File names consist of a one to eight alphanumeric character name and an optional one to three alphanumeric character extension. You cannot have a NULL file name. The complete specification cannot exceed 127 characters.

Directories are distinguished from files in a path specification by either backslash (\) or slash (/). FlexOS recognizes the following abbreviations:

./ means the current directory
../ means the parent directory
// means the root directory of the specified device

Although ./, ../, and // are most useful at the user interface level, the FlexOS logical name substitution means these abbreviations can also be useful at the programmatic level as well. Note that // ignores whatever directory specification preceded the // and specifies the root directory on the specified device or, if no device was specified, the default device.

Paths are also used to identify pipe files, console files, and devices. The following are examples of path specifications.

remote disk file	svr::hd:\dir\FILE.EXT	full path
disk file	hd1:/mydir/file.typ	device, directory, and file
pipe	pi:mypipe	
virtual console	con1:vc002/console	screen and keyboard for virtual console #2
device	mydevice:	
abbreviations	m:a/b\./../x	means m:a/x
	m:a/b//c/d	means m:c/d

1.3.1 Uppercase Versus Lowercase Names

File names can consist of uppercase and/or lowercase characters. Name matching is conducted according to the following rules. The rules are summarized in Table 1-5.

- The Disk Resource Manager accepts two types of disk media, uppercase media (default) and case sensitive media. You make the selection in the disk label. All file names on uppercase media are converted to uppercase. On case sensitive media, the Disk Resource Manager either converts names to lowercase or leaves them as is, depending on the force case flag in the SVC.
- Device names are always lowercase and are searched in force lowercase mode. This way, an uppercase or lowercase name will match a device name.
- The DEFINE SVC forces logical names to lowercase but leaves the substitution string as is. All logical names are forced to lowercase when the define tables are searched, but left as is if no substitution occurs.
- Programs using FlexOS's SVCs can choose between force case or not.

Table 1-5. Rules for Forcing Name Case

Device	Default Case	Cases Supported	Forced Case
Disk	Upper-only	Upper-only Mixed	Upper Lower
Pipe	Mixed	Mixed	Lower
Console	Lower-only	Lower-only	Lower
Miscellaneous	Lower-only	Lower-only	Lower

1.3.2 Wildcards

Wildcard characters are available for use with the FlexOS LOOKUP SVC. This supervisor call is a scanning tool that searches tables by type and extracts items matching the name specification in the LOOKUP call. Where there is a match, LOOKUP puts all or part of the table into a buffer. The Table 1-6 lists the wildcard characters.

Table 1-6. Wildcards

Wildcard	Meaning
*	Matches any number of characters, 0 or more
?	Matches any single character
^	Finds names that do not match the wildcard name

The * and ? characters can be freely intermixed with characters that must be in the item names. The ^ must be the first character of the wildcard name. The following examples illustrate the use of wildcard characters.

Suppose the following set of names exists for a table type:

a ab abc bac bb bc c

The following wildcard names would specify the indicated set:

*	a,ab,abc,bac,bb,bc,c (all files)
*c	abc,bac,bc,c (all files ending with c)
^*c	a,ab,bb (all files not ending with c)
?b	ab,bb (all files with a 2 character name ending with b)
a*	a,ab,abc (all files starting with a)
b	ab,abc,bac,bb,bc (all files with b anywhere)
?	a,c (all files with a 1 character name)
?*b*	ab,abc,bb (all files with b anywhere after 1 character)
*b?	abc,bb,bc (all files with b as next-to-last character)

You can have logical name translation with LOOKUP and use paths in your LOOKUP name specification. In path specifications, wildcards can only be used in the last element of path. The following examples demonstrate valid and invalid uses of wildcards in LOOKUP name specifications.

Specification	Explanation
hd:/B1/GL*.*	Valid: Returns the table for all files in directory B1 on device hd: that begin with GL.
pi:^mx\input	Invalid: The wildcard cannot be used if the file is not at the end of the specification.
hd?:/	Invalid: The wildcard cannot be in the device name if there are subsequent directory or file references.
hd?:	Valid: Returns the table for all devices beginning with hd.

1.3.3 Reserved Names

Table 1-7 lists file names reserved by FlexOS. The BOOTINIT script initially defines default: in the process logical name table and defines system: and boot: in the system logical name table.

Table 1-7. Reserved File Names

Name	Definition
stdin	The standard input file.
stdout	The standard output file.
stderr	The standard error file.
stdcmd	Reserved for system use.
prn:	The system list (print spooler) device.
default:	The process's current directory: FlexOS expands a NULL path to the path associated with default:. A path consisting of filename alone is expanded to begin with default:.
system:	The process's system directory: The system directory is intended as the location to store shared program and data files. FlexOS searches it after any unsuccessful attempt to find a match in the default: directory when the path specification consists of a file name alone. Files in the system: directory must have the system attribute set to be loaded in this manner.
boot:	The system boot directory: Device drivers are typically located in the boot directory.

1.3.4 Logical Name Substitution

FlexOS contains a logical name preprocessor which allows paths to be represented by a single logical name. FlexOS checks the first item in a path specification against a logical name table and substitutes the

replacement string when a match is found. An item is defined as a character string delimited by a NULL, space, tab, or colon. For example, if you define home: to be the string

```
floppy1:dir1/dir2/
```

then the path specification home:datafile is expanded to:

```
floppy1:dir1/dir2/datafile
```

After the replacement string has been inserted into the original path specification, FlexOS checks the first item again for a replacement string. This loop continues until no replacement is found. The complete file specification after all substitution has been performed cannot exceed 127 characters.

If the file datafile in this example is a logical name, FlexOS does not search for the replacement string because it is not the first item in the path specification.

FlexOS maintains a single, system-wide logical name table--the SYSDEF table--and separate logical name tables for each process--the PROCDEF tables. FlexOS cross-references the logical names in the PROCDEF table first and then the SYSDEF table. You make changes to both tables with the DEFINE SVC; however, only privileged users can make changes to the SYSDEF table. You can assign logical names for complete or partial paths.

When a process creates another process, the new process, called the child, inherits a copy of its parent's local process logical name table. Any changes the child process makes affect its table only. The parent's table is not modified. This is how the logical names replacements for the standard files are passed from parent to child processes.

1.4 File Access

FlexOS monitors file access for four types of privileges--read, write, delete/set, and execute--and three types of users--owner, group, and world. For disk files, access is monitored only when disk security is enabled. (See the description of the disk label in Section 2.4.1 for the description of disk security.) Before you can read from or write to a

file, you must open it. In your open call, you select which privileges (read and/or write) you require and specify an access mode. The access privileges available to you depend upon your user and group ID numbers.

When the open is successful, FlexOS returns a 32-bit file number. You subsequently access the file by its number. FlexOS keeps all file numbers in a global table of open files and uses them to dispatch requests to the proper resource manager. The number is disassociated from the file when you close it.

1.4.1 Standard File Numbers

FlexOS reserves four file numbers for reference to the standard files. Table 1-8 lists these file numbers by their reserved names.

Table 1-8. Standard File Numbers and Names

File Number	Name	Description
0	stdin	standard input file
1	stdout	standard output file
2	stderr	standard error file
3	overlay	overlay file

Note: The overlay file is the command file from which the program was loaded. This file is left open when an indication of overlays exists.

These numbers are not the actual file numbers of your standard input, output, error, and overlay files. FlexOS translates these numbers into the actual file numbers. The definition of the standard to actual file numbers is made by the shell or window manager program. Should you need the actual file number, you can get them from the ENVIRON table.

The COMMAND SVC opens stdin, stdout, and stderr. These names are inherited from the parent process which called the COMMAND SVC. The standard input file is opened for read access in shared file pointer mode; the standard output and the standard error files are opened for write access in shared file pointer mode.

1.4.2 Access Privileges

There are four access privileges: read (R) allows the process to read from the file; write (W) allows the process to write to the file; execute (E) allows the process to run the program; and delete (D) allows the process to delete the file and set values in the file's table.

Access privileges are assigned on a owner, group, and world basis when the file is created. Which access privileges are available to a given process is determined by comparing its user and group identification numbers against the file creator's. At log in, FlexOS reads the user's ID numbers from the USER.TAB file. The comparison is made when the user attempts to open, execute, or delete the file. If both numbers match (indicating the user is the file owner), FlexOS allows the user the access privileges established for the owner. If there is a match on the group ID only, FlexOS allows only the group-level access privileges. If neither match or the user IDs match but the group IDs do not, only world-level privileges are available.

User, group, and world categories are independent and do not have to provide diminishing levels of access. For example, you can set the world level to have complete rights over a file, while the group level can only write to the file and the owner can only read the file. The file owner and superuser can always change the attributes of the file, regardless of the security word.

The privileges available for owner, group, or world access are kept in the file's File Security Word. The File Security Word is a 2-byte bit-map of the access privileges by level as shown in Figure 1-3. The values are set in the CREATE call. Only disk and pipe files and directories have a File Security Word. Console file access privileges are determined by the mode.

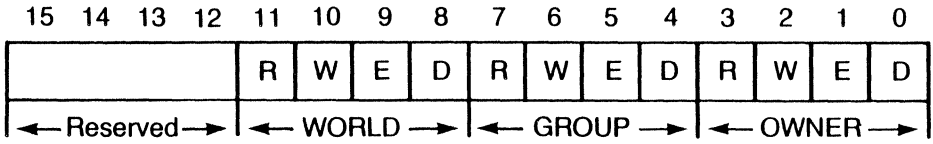


Figure 1-3. File Security Word

The execute and delete privileges are determined when the process attempts to run or erase the file. You do not need to open the file for either operation. You can, however, delete a file once it is open. The delete is not performed until the last close is performed on the file.

You select the process's read and/or privileges in bits 2 and 3 in the OPEN SVC's flags. If the privileges requested are available in the File Security Word, the resource manager checks them against the file's current access modes (see below). If the privilege is available given existing modes, the file is opened and the file number returned.

When a requested privilege is not available, the OPEN succeeds or fails depending on the value of the Reduced Access flag in the OPEN call. The access level granted is derived by ANDing the privileges requested with those in the File Security Word. For example, if the requested OPEN access rights are RW and the File Security Word access rights are RDE, then the reduced access right is R--the only common access privilege. The resource manager determines if that privilege is available given any current access modes before opening the file and returning the file number. If none of the requested rights match, then an access violation error code is returned and the file is not opened.

1.4.3 Access Modes

FlexOS provides a set of access modes which determine whether or not and, if so, how open files are shared. These modes are selected in the OPEN flags word and consist of the following:

- EX: exclusive access by calling process
- AR: allow reads by other processes
- ARW: allow reads and writes by other processes

The default mode is exclusive access, where the calling process prevents any other process from sharing the file. Exclusive access to a file is denied if another process has the file open.

If a process tries to open a file with write privilege and another file has the file open in (AR) mode, then the new open is denied and an error is returned.

ARW mode has two options: shared or unique file pointer. The shared file pointer mode is only available to processes with the same family ID, and all processes in the family must specify this mode.

For processes outside of the family, the file appears in exclusive mode. There are no such restrictions when the unique file pointer option is selected.

1.4.4 File Pointers

FlexOS supports both sequential and random access to pipes and disk files. Sequential file access is supported by a file pointer. File reads and writes increment the pointer so you need not constantly calculate your position within the file. Random file access is supported through the use of offsets in the READ and WRITE supervisor calls. The offset can be specified relative to the file pointer, the beginning of the file, or the end of the file.

The file pointer is initialized to 0 when you create or open the file. Subsequent reads and writes move the file pointer to the byte position of the next sequential location. For example, if a new file is created and then 12 bytes are written, the file pointer would be pointing at the 13th byte (essentially the EOF marker).

Separate processes sharing access to the same file can share the same file pointer or can have separate ones. File pointer sharing is limited to processes with the same family identification number (FID). When the pointer is shared, READ or WRITES by any process update the file pointer. Use the SEEK SVC to determine the file pointer's location. SEEK can also be used to set the pointer's location.

Random access on printer, console, and other serial files produces results that are device dependent. Consequently, file pointers are not maintained on these types of devices but rather assume an offset of 0 independent of the actual request.

1.5 Deleting Files

Files are deleted by name with the DELETE SVC. Unless the disk security has been enabled or the file has the read-only attribute, there is nothing to prevent the calling process from erasing the file. A file cannot be erased when it is set read-only. When file security is enabled and the file has not been opened, the calling process must have the delete privilege. If the process has the file opened, it must have either write or delete privilege.

FlexOS does not immediately erase an open file when you try to delete it. Instead, FlexOS returns success to the DELETE call but marks the file as temporary. FlexOS leaves files marked temporary available until the last close is performed. At this point, the file is deleted.

You can automatically delete files by setting one of two flags when you create the file. One CREATE flag designates the file as temporary or permanent. Temporary means the file is deleted after the last open is closed; permanent means the file remains after the last close. The other CREATE flag deletes a file if it has the same name as the file you are creating. (Alternatively, you can have CREATE return an error if it finds a file with the same name.)

1.6 Basic Terms

Table 1-9 defines the special terms used in this manual.

Table 1-9. FlexOS Operating System Terms

Term	Meaning
Buffer	Address of buffer: Many SVCs require buffers for either I/O or information. Buffers must be within the logical address range of the calling process. FlexOS checks the buffer address and size to ensure legal buffers.
Bufsiz	Size of buffer (in bytes): The size of the buffer sets the SVC's limit. For instance, the buffer size indicates the number of bytes to transfer in the WRITE SVC. The buffer size is also used with the buffer address to catch illegal buffer specifications.
Completion code	The return code of an asynchronous event.

Table 1-9. (Continued)

Term	Meaning
Event	Asynchronous operation: When a process issues an asynchronous SVC, the requested activity is called an event. For example, in an asynchronous write call to a printer, the event is the output of the character or string. Events can be successful or unsuccessful, the latter indicating that the resource manager's or driver's error recovery mechanism determined that the action could not be completed. A process can have up to 31 ongoing events.
Event Mask	Asynchronous SVC return value: When you call an asynchronous SVC, a 32-bit value is returned immediately. If it is positive (the most significant bit is 0), the value is the event mask for that event. If the value is negative, the SVC could not be performed. The event mask is a unique value in which one of bits 0 to 30 is set to designate the event started. You use this value in subsequent calls to check event status and to retrieve the event's completion code.
Flags	The flags word in many SVCs offers options that are enabled by setting a bit. Not all SVCs have flags. Bit 0 in the SVC descriptions corresponds to the lowest order bit and bit 15 the highest. All unused bits must be set to 0.
Fnum	File number: SVCs that do I/O require a file number. You get the file number from the OPEN and CREATE SVCs.
Name	File specification address: File specifications are not typically entered in an SVC. Instead, you enter the address of a NULL terminated string containing the complete specification. For all SVCs, the maximum length string is 128 bytes limiting you to a 127-byte file specification.

Table 1-9. (Continued)

Term	Meaning
OEM	Original Equipment Manufacturer: In the context of this manual, the OEM is the person or company who integrates FlexOS with the computer or develops the interface to a supplemental piece of hardware such as a plotter or communications card.
Option	SVC options: Several SVCs have, besides the flags, options numbered from 1 to 255. Where options are available they are shown in the SVC descriptions in Section 7. OEM-supplied SPECIAL calls may also have options not documented in this manual. Select the option by entering the corresponding value in your call or parameter block.
Privileged user	A process with group and user numbers of 0. Group and user numbers are established when FlexOS is loaded from information in the USER.TAB file
Process	Program entity: FlexOS provides a multitasking environment in which multiple processes can execute program instructions independently of each other. Processes are uniquely identified by a process identification number and are related to other processes through a family idenfication number. A process is always in one of three states: • **running** when it has the CPU • **ready** when it could use the CPU if it had it • **blocked** when it is waiting for an event to complete

Table 1-9. (Continued)

Term	Meaning
Return code	LONG value returned by a Supervisor call (SVC).
Superuser	Synonymous term for a privileged user.
swi	Software Interrupt Routine: Each asynchronous SVC allows the optional use of a software interrupt routine (swi) that functions similarly to a hardware interrupt routine. When the asynchronous SVC completes its operation, the calling process is interrupted and control passes to the swi. When the swi is finished, it either returns control back to the main process where the main process was interrupted or becomes the main process. It is not necessary to have a swi specified to execute an SVC asynchronously.
Table	FlexOS data structure: FlexOS provides information about itself in structures known as Tables. You can examine these tables and in many cases control process environments by setting values in the tables. FlexOS also provides an SVC for scanning and retrieving portions of tables. Table 1-10 lists the FlexOS tables.

1.7 Tables

You can monitor most aspects of FlexOS operation through its tables. Use the GET and LOOKUP SVCs to retrieve the information and the SET SVC to modify those table fields that are read/write. Tables are assembled by the supervisor when you make the call; they are not maintained in system memory. Section 8 contains detailed information about FlexOS tables. Table 1-10 lists the tables.

Table 1-10. FlexOS Tables

Table Name	Contents
Supervisor/Kernel	
PROCESS	Process information
ENVIRON	Process environment information
TIMEDATE	System time and date
MEMORY	System memory information
SYSTEM	Global system information
FILNUM	Table information for a given file number
SYSDEF	System level defined names
PROCDEF	Process level defined names
CMDENV	Command line entry
DEVICE	Information on devices
PATHNAME	Fully-expanded path for given logical name
Pipe	
PIPE	Pipe information
Disk	
DISK	Disk device information
DISKFILE	Disk file information
Console	
PCONSOLE	Physical console information
VCONSOLE	Virtual console information
CONSOLE	Screen and keyboard information
MOUSE	Mouse information
Miscellaneous Device	
PRINTER	Printer device information
PORT	Port device information
SPECIAL	Special device information

1.8 FlexOS Functional Components

The computer system software can be grouped into three categories. Figure 1-4 illustrates the three categories and their relationship to each other.

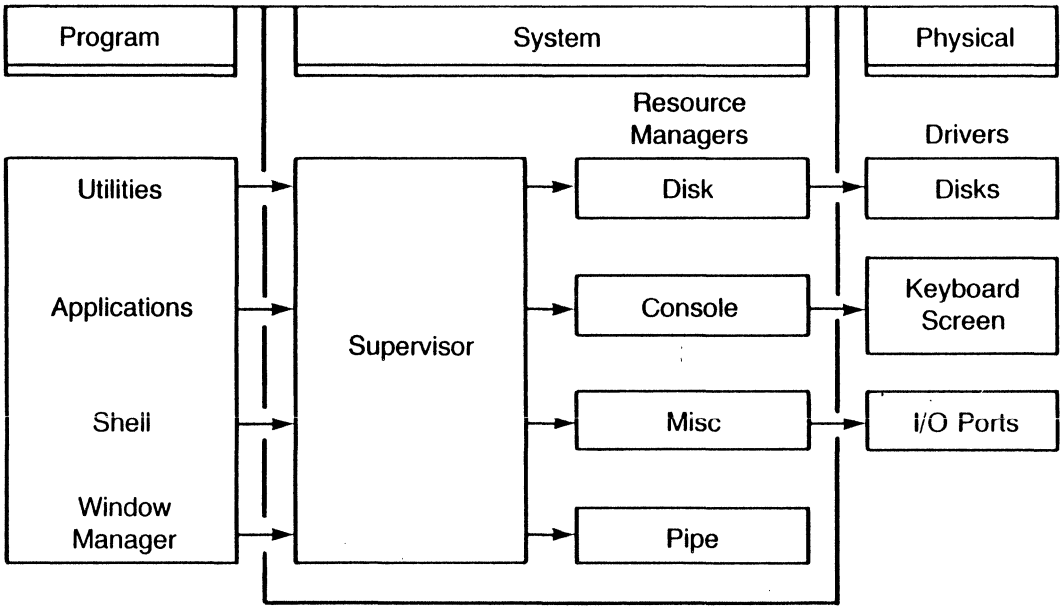


Figure 1-4. Computer System Software Categories

The categories are defined as follows:

- Program: This group includes the applications run by users to perform tasks and various system management utilities. Background programs which control the user interface such as the shell and window manager also fall into this category.

- System: This group provides the file system services, process scheduling, and data flow mediation. Programs use these services on a system call basis. The supervisor receives the functions and sends them to the appropriate resource manager for servicing.

- Physical: The physical functional unit contains the device-specific code, called device drivers. The physical functional unit varies for each computer system. This unit translates the generic SVCs from the resource managers into the device-specific routine for execution.

These divisions illustrate FlexOS's two interfaces: the program-to-system interface and the system-to-physical interface. This manual describes how to call the Supervisor and what you get in return. Refer to the FlexOS System Guide for detailed information on how the system functional unit relates to the physical functional unit.

1.8.1 The Supervisor and Resource Managers

The Supervisor receives SVCs from the program units and sends them to the appropriate resource manager. File numbering is another of the Supervisor's duties. Every time you open a file or device or create a file, the Supervisor returns the file number. You use this number to access the file, and the Supervisor uses it to send the call to the proper resource manager.

The resource managers control the access to the physical devices and pipes. Table 1-11 lists the resource managers and summarizes their tasks.

Table 1-11. Resource Managers

Resource Manager	Task
Disk	Manages the disk file system for disk drives.
Console	Manages physical and virtual consoles.
Pipe	Manages interprocess communications through FIFO (first-in-first-out) memory files called pipes.
Misc	Manages all devices not managed by the other resource managers.

1.8.2 Kernel

Not shown in Figure 1-4 is the FlexOS kernel. This proprietary module is responsible for all process management tasks. This includes process creation, state maintenance, and dispatching. The kernel also manages process context switching and scheduling and memory allocation.

Process scheduling is performed on a priority basis. Priority is established at program invocation by a number in the range of 0 to 255 (see the COMMAND description in Section 7). 200 is the recommended priority for user processes. Higher numbers have a lower priority; lower numbers have a higher priority. Processes with the same priority are scheduled on a round-robin basis.

End of Section 1

Disk File Management

This section describes FlexOS's disk file management tools and the fundamental concepts involved in dealing with files. Table 2-1 lists the SVCs available for disk device, directory, and file management.

Table 2-1. Disk Resource Manager

SVC	Disk Device	Disk Directory	Disk File
CLOSE	Y	Y*	Y
CREATE	Y	Y	Y
DELETE	Y	Y	Y
DEVLOCK	Y	N	N
GET	Y	Y	Y
LOCK	N	N	Y
LOOKUP	Y	Y	Y
OPEN	Y	Y*	Y
READ	Y	N	Y
RENAME	N	Y	Y
SET	Y	Y	Y
SEEK	N	N	Y
SPECIAL	Y	N	N
WRITE	Y	N	Y

* You open and close a directory file to get and set its DISKFILE table only; you do not open it to read from or write to it.

2.1 File Access

Access to files is initiated using the OPEN or CREATE SVC. Use OPEN to open an existing file; use CREATE to make and open a new file. Both calls require you to specify a file name; both return a 32-bit file number. You use the file number for all subsequent file operations. The CLOSE SVC disassociates the file number from the file. Use the DELETE SVC to remove files.

Files can be accessed in a byte-oriented manner. Any record in a disk file can be accessed at random. The file system maintains a byte level end-of-file on disk files.

2.2 Disk File Attributes

Each file in FlexOS has attributes that control access and define characteristics. The attributes are initially established by setting the ATTRIB word in the DISKFILE tables. Any user with the delete/set access privilege can change the ATTRIB word. The Disk Resource Manager records this value in the file's directory entry. Table 2-2 lists the attributes.

Table 2-2. FlexOS Disk File Attributes

Attribute	Meaning
Read-only	The Read-only attribute overrides the access rights that are User/Group based. A process cannot delete or write to a Read Only file even if it has write and delete privileges for the file.
Hidden	Files with the Hidden Attribute ON are not shown in a directory listing unless you use a special option.

Table 2-2. (Continued)

Attribute	Meaning
System	Files in the system: directory can be opened indirectly when the System attribute is ON. Indirectly means, "from another directory." On each open, if a filename is given without device or directory specification, the Disk Resource Manager first searches the default: directory. If the file is not found, the system: directory is searched. If the file is found and the System attribute is ON, the file is opened. Files with the System Attribute ON are not included in a directory listing unless you explicitly ask to see them (see LOOKUP in Section 7).
Archive	If the Archive Attribute is OFF, the file has been archived since it was created or last modified. It is automatically turned ON if the file is modified. Programs performing backup functions can turn this attribute OFF to perform incremental backups.

2.3 Disk Media

FlexOS disk media have the following characteristics:

Disk label
: A root directory entry containing the label name, user and group number of the label's creator, and mode flags. The mode flags determine if disk security is enabled and whether uppercase and lowercase or just uppercase file names are supported.

File security
: A four-byte field in the file's directory entry containing the creator's user and group number and the two-byte File Security Word. Both are set when the file is created.

File record size
: A two-byte field in the file's directory entry indicating its record size. A record size of zero is equivalent to a record size of one byte.

The File Security Word and record size are set when the file is created and the disk label is initialized to 0. The label and its options are set in the DISK table.

2.4 Disk File and Directory Security

File and directory access is controlled by four factors:

- The security enable flag value in the disk label.
- The user and group ID of the calling process.
- The owner, group, and world access privileges available.
- The access privileges requested in the OPEN call.

The read-only attribute supersedes the access privileges. An error will be returned if you attempt to write to, set DISKFILE values of, or delete a file with the read-only attribute return.

2.4.1 Disk Label

The disk label is created when you set the LAMODE and LABEL fields in the DISK table for the first time. The Disk Resource Manager completes the remainder of the DISK table's label fields by adding the label maker's user and group IDs and setting the LAFLAG. Subsequently, only that user or a superuser can change the label. You cannot remove a label after it is created. You can set all fields to NULL.

File security is not enabled on a disk without a label. Once the label is established, you enable and disable disk security by setting and resetting LAMODE bit 0. When file security is disabled, all processes have full (R,W,E,D) access to all files on the disk. When file security is enabled, all users except the superuser are monitored for read, write, execute, and delete privilege according to their user and group ID. The superuser always has full (R,W,E,D) access to files; regardless of the File Security Word contents.

The disk label also determines if the drive supports uppercase only or uppercase and/or lowercase file names.

2.4.2 User/group IDs and Available Access Privileges

Before a file is opened, FlexOS grants all processes the minimum access privileges. This lets the process lookup the file's DISKFILE table. To execute, read from, write to, or set the file's attributes, a process must have the corresponding privilege. FlexOS qualifies a process for read or write privilege when the process attempts to open the file. Execute and delete privilege are determined when the process attempts to run and delete the file, respectively.

To determine the access privileges available to a process, FlexOS compares the process's user and group ID against the file creator's user and group ID. This indicates whether the process falls into the owner, group, or world category. The privileges set in the file's File Security Word for that category are the only ones available to the calling process.

The privileges given to the calling process are dependent on three other factors: the comparison of the privileges requested to those available, whether or not the file has the read-only attribute, and any current access modes. FlexOS compares the privilege requested against those specified in the File Security Word. If there is a match, FlexOS then determines if the file is read-only. Finally, FlexOS checks the file to see if it is open and, if so, the access mode is set. Some access modes--for example, write exclusive mode--prevent all other processes from using the file. Other access modes--for example, read exclusive--let other processes open the file but only for the purpose of reading.

FlexOS opens the file and returns the file number, executes the program, or deletes the file when the requested privileges are available. The function is not performed if the privileges requested do not match those available or are not available given the current access mode. Processes can acquire reduced access by setting the corresponding flag bit in the OPEN call.

2.4.3 Directory Versus File Access Privileges

The user and group mechanisms used to qualify users for access to files are also used for directory security. However, access privileges to directories have a slightly different meaning than they do for files. Table 2-3 compares the two meanings. Directory security, like file security, is only enabled when the corresponding LAMODE bit is set.

Table 2-3. Privilege Definitions for Files and Directories

Security Privilege	File	Directory
Read (R)	Allows reading from a file	Allows LOOKUP operations on files in the directory
Write (W)	Allows writing to a file plus the privileges listed for delete/set	Allows file creation and deletion
Execute (E)	Allows a file to be executed	Allows opening of files in the directory
Delete/Set (D)	Allows renaming, changing file attributes, or deleting files	Allows changing attributes of files in directory

2.4.4 Access Rules and Restrictions

Read-only file attribute overrides file access privileges set in the File Security Word. The following list describes other access restrictions for files and directories. Recall that the rules only apply when disk security is enabled.

- To access any file you need execute access in each directory specified in the pathname of the file.
- To LOOKUP files you must have read access to the last named directory in the path. No access is needed of the files themselves.
- The GET SVC requires only a file number; you do not need any access privilege to a file's DISKFILE table.

- The SET SVC requires write access to the directory the file is in, as well as delete or write access to the file itself. The file must be successfully opened with delete access before SET can be called. A file owner cannot use SET to change file attributes without write access to the directory. After obtaining write access to a file's directory, the owner can always obtain delete access to a file, even if is set to R/O.
- The DELETE and RENAME SVCs require write access in the directory as well as delete or write access for the file. No exception is made for the owner of the file.
- The COMMAND and OVERLAY SVCs require execute access to the file being loaded. Read access in not required.
- The CONTROL SVC requires both execute and read access to load a file for debugging.
- The READ SVC requires read access to the file.
- The WRITE SVC requires write access to the file.

2.5 Disk File Access Modes

The FlexOS disk file system divides open modes and the privileges allowed with each mode into three categories.

1. All exclusive opens, with the exception of read/exclusive (R/EX) reserve the file for the exclusive use of the calling process.
2. (R/EX) opens are treated as read/allowed shared read (R/AR) opens in order to allow the shared open of read/only files by multiple processes.
3. Shared read/write opens do not restrict file access by other processes. You can restrict record access with the LOCK SVC.

The first category applies to other processes only. Previous open modes set by a process do not delimit its subsequent open modes options. Thus, a process with a file opened in read/write exclusive mode can open the file again and in any other mode. However, the exclusive mode is in force until that open is closed.

2.6 Direct Disk Access

There are two ways to access the disk directly:

- with the READ and WRITE SVCs
- with the SPECIAL SVC disk functions

Both methods require you to open the disk drive before access is provided. Use the OPEN SVC for this purpose using the device name to select the drive and the OPEN flags to select the access privileges and access mode. FlexOS returns the file number number you use in your READ, WRITE and SPECIAL calls. Disk security measures are provided to restrict access.

2.6.1 Disk Device READ and WRITE

Using the READ and WRITE SVCs requires the process to have read and write privilege and the drive to be installed to allow raw reads and writes. In your calls, the Disk Resource Manager translates the offset specified into a logical record number. (The disk is treated as a serial sequence of records starting with the first head, cylinder, and sector and ending at the last sector, cylinder, and head.) The buffer size you specify must be a multiple of the sector size and all operations must be performed on sector boundaries. The information transferred is the data portion of the sector only; the sector header is not included.

2.6.2 SPECIAL Disk Functions

The SPECIAL disk functions provide the disk format capabilility and direct access to any sector of the disk, including the system area, using the file system's head, cylinder, and sector identification scheme. Table 2-4 summarizes the SPECIAL functions and the access modes required to use them.

Note: We cannot guarantee the compatibility of the SPECIAL disk functions with future releases of Digital Research® operating systems.

Table 2-4. SPECIAL Disk Functions

Function	Description
0[1]	Read the system area of the disk
1[2]	Write to the system area of a disk
2[2]	Format the system area of a disk
3[2]	Format a track of the disk
4	Check the media for change or errors
5	Flush buffer contents to the disk
6[1]	Read physical record by head, sector, track
7[2]	Write physical record by head, sector, track
8[2]	Set drive's Media Descriptor Block (MDB)

1 Must open in at least shared read-only mode.
2 Must open in exclusive mode.

2.6.3 Disk Drive Open Modes

Disk device access is subject to the current access modes. You specify the mode you require along with the READ and/or WRITE privilege in your device OPEN call. FlexOS compares the request against the privileges available, any modes in affect, and, when write/exclusive mode is requested, the presence of open files by other processes. Three open modes are supported for direct disk access:

- GET-only
- Shared read-only (AR)
- Exclusive (EX)

The GET-only mode starts when you open the drive without requesting any access privileges or modes. Use this mode to make a connection to the device. This connection allows you to use GET to retrieve the drive's DISK table and DEVLOCK to lock the device. This type of open has no effect on disk usage by other processes until DEVLOCK is initiated.

The shared read-only mode allows the calling process to have read, write, and set access. Other processes are limited to read access with the SPECIAL read functions, with the READ SVC, and through the disk file system. Use DEVLOCK to restrict disk access further.

The exclusive mode precludes all access attempts by other processes. The calling process can declare read, write, and/or set access. FlexOS does not grant exclusive mode while there are open files or existing DEVLOCKs on the drive. While the disk is open, no file system operations can be performed. All the SPECIAL disk functions supported by a device driver can be accessed in this mode if the calling process has obtained the required access level. This is the only mode that lets you set the disk label.

NOTE: Superusers get full read, write, and delete access to a disk for any mode, DEVLOCK status, or INSTALL option.

2.6.4 Disk Security INSTALL Options

Disk security is established in two places: Options selected when the disk driver is installed and the options set in the disk label. See 2.4.1 above for the description of the disk label.

The installation options offered in the INSTALL SVC follow:

- Removable or Permanent Device
- Device raw reads allowed
- Device raw writes allowed
- Device set allowed
- DEVLOCKs allowed

The device read, write, and set options control the level of direct access to the disk supported by the device driver. Disk security cannot be guaranteed if the disk driver allows raw reads and/or writes. When a disk is opened as a device, the read, write, and set options determine the allowed access level.

The DEVLOCK option determines whether processes can use the DEVLOCK SVC to lock the drive. The DEVLOCK SVC allows a process to lock a disk for its exclusive use or the exclusive use of processes in the same family. Superusers can use the DEVLOCK SVC regardless of this option.

End of Section 2

Console Management

This section describes how to perform console I/O under FlexOS. The presentation has four parts:

- The first part describes general characteristics of the console system and introduces the supervisor calls and tables used to manage it. Also described are the FRAME and RECT data structures and the console file naming conventions.
- The second part describes how to use the WRITE, ALTER, COPY, and READ SVCs to control the screen and keyboard.
- The third part describes how to monitor console input from the keyboard and pointing device with the READ, XLAT, RWAIT, and BWAIT SVCs.
- The fourth part describes use of the CREATE, OPEN, KCTRL, GIVE, ORDER, SET, and GET SVCs to create and manage virtual consoles and windows.

The 8-bit and 16-bit character sets referenced in this section are described in Appendix A. For the list of the country codes mentioned below, see Appendix C.

3.1 Console File System

A console under FlexOS consists of a keyboard and screen and optionally a pointing device. For convenience, the term **mouse** is used to refer to all kinds of pointing devices.

The Console Resource Manager controls console I/O on a file-oriented basis. A single file can represent the keyboard and screen or you can have separate files. With a single file, read and write access are independent so that keyboard and screen access privileges and modes can be different. Separate files are used to monitor mouse input and to represent window borders.

Other Console Resource Manager features are: 8-bit and 16-bit character modes (individually selectable for keyboard and screen), escape sequence decoding, keystroke translation, and multiple international character sets. All features are turned on or off with the SET SVC.

FLexOS maintains each physical console independently of other consoles on the system, so different features and options can be selected for each console. The same independence applies to virtual consoles.

The COMMAND or CREATE SVCs open the standard files stdin (file number 0), stdout (file number 1), and stderr (file number 2). The files are opened in shared file pointer mode so all processes in the family have console access. (See Section 5 for the explanation of process families.) The definitions for these logical names are inherited from the parent process. The Supervisor always translates file numbers 0, 1, and 2 to the actual file numbers.

For applications invoked from the shell, the standard files should represent the keyboard and screen. However, these files might be defined to be other than console files through redirection. Get the FILNUM table for files 0, 1, and 2 to determine the type of device each references.

3.1.1 Console-Related SVCs

The console-related SVCs provide two types of services: console file I/O and virtual console management. The first type are the SVCs you use in applications and utilities to control the screen and gather user input. Use the second type in window management programs and applications to create and control virtual console displays. Table 3-1 lists the console-related SVCs by type.

Table 3-1. Console-Related Supervisor Calls

SVC	Purpose
Console File I/O	
ALTER	Modify a RECT (rectangle)
COPY	Copy a RECT from one FRAME to another
READ	Read from a console file
WRITE	Write to a console file
XLAT	Translate keyboard input
BWAIT	Wait for mouse button state change
RWAIT	Wait for mouse to enter or exit a RECT
Virtual Console Management	
CLOSE	Close a console file
CREATE	Create a virtual console file
DELETE	Delete a virtual console file
DEFINE	Set process's stdin, stdout, and stderr files
GET	Get a table
GIVE	Transfer physical keyboard and mouse ownership
KCTRL	Obtain physical keyboard and mouse ownership
LOOKUP	Scan virtual console tables
OPEN	Open a virtual console file
ORDER	Change order of virtual consoles
SET	Change table contents

3.1.2 Console-Related Tables

The Console Resource Manager also maintains tables for each physical, logical, and virtual console and mouse so applications can determine their console environment and, to the extent allowed, change it. Table 3-2 lists the tables associated with console management and indicates the console characteristics maintained in that table. Complete descriptions of the tables and their contents are provided in Section 8.

Table 3-2. Console-Related Tables

Table Name	Information
CONSOLE	Number of characters in keyboard's type-ahead buffer Screen and keyboard modes Cursor position Number of character rows and columns Virtual console number Console type Physical console name
ENVIRON	Current stdin, stdout, and stderr file numbers
PROCESS	Process's DEFINEd physical console number Process's virtual console number
VCONSOLE	Window mode Virtual console number Console type View origin reference point on virtual console Total character rows and columns in window Window position reference point on parent console Total rows and columns in virtual console Top, bottom, left, and right border sizes
PCONSOLE	Physical device name and identification number Current number of virtual consoles Number of pixel and/or character rows and columns Console type FRAME planes supported Attribute and extension plane bit maps Country code Number of function keys Number of mouse buttons Mouse serial number

Table 3-2. (Continued)

Table Name	Information
	Current mouse form position Keystate of Alt, Control and Shift keys Current state of mouse button Mickeys/pixel sensitivity of rows and columns Click interval time period Height and width of mouse form Position of mouse form hotspot Mask to mask effect of DATA rectangle DATA rectangle to "BLT" to screen

3.1.3 Console Screen Model and Data Structures

The screen is represented by a three-dimensional data structure called a FRAME. The FRAME's height and width are defined in terms of character columns and rows. Each intersection of a row and column defines a FRAME character cell. A cell is always one byte. Figure 3-1 illustrates the FRAME model.

The FRAME's depth is defined in terms of planes, each with the same dimensions as the FRAME. There are three planes: character, attribute, and extension. Each plane consists of either a two-dimensional byte array or a single byte used by the Console Resource Manager to set all plane bytes.

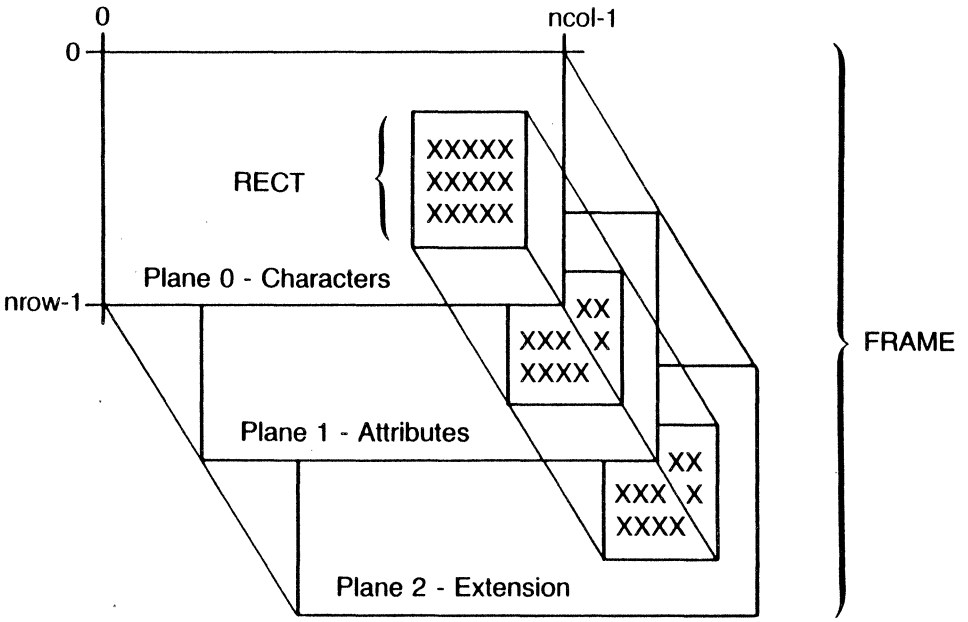


Figure 3-1. FRAME Planes with RECT

Plane Descriptions

The FRAME planes are defined as follows:

- Character Plane (plane 0): Each byte corresponds to a text character space on the screen. The 8-bit character set used in this plane is defined on a per country basis.
- Attribute Plane (plane 1): Each byte defines the foreground color, background color, and color intensity and contains a blink flag for the corresponding character cell. The attribute plane byte is used as shown in Figure 3-2.

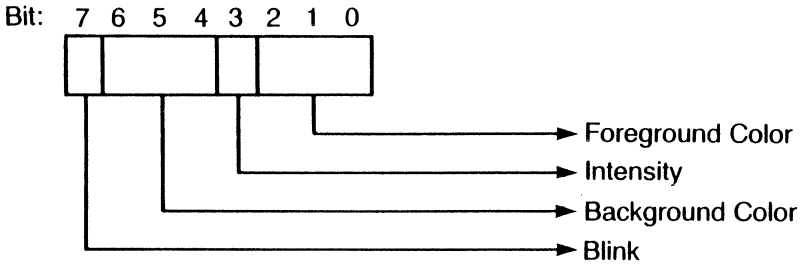


Figure 3-2. Attribute Plane Byte Format

- The three bits in the foreground and background color fields are assigned as follows:

  low bit: blue
  middle bit: green
  high bit: red

  Table 3-3 lists the colors corresponding to each 3-bit value in the lefthand column. The righthand column shows the foreground color resulting when the intensity bit is set.

Table 3-3. Foreground and Background Colors by Byte Value

Foreground and Background Colors	Foreground Color with Intensity Bit Set
0 - black	8 - dark gray
1 - blue	9 - light blue
2 - green	10 - light green
3 - cyan	11 - light cyan
4 - red	12 - light red
5 - magenta	13 - light magenta
6 - brown	14 - yellow
7 - light gray	15 - white

Set bit 7 to have the character blink. This feature is not available if the hardware does not support it.

- Extension Plane (plane 2): An OEM-implemented option that provides support for 2-byte characters. Each extension-plane byte is formatted as shown in Figure 3-3.

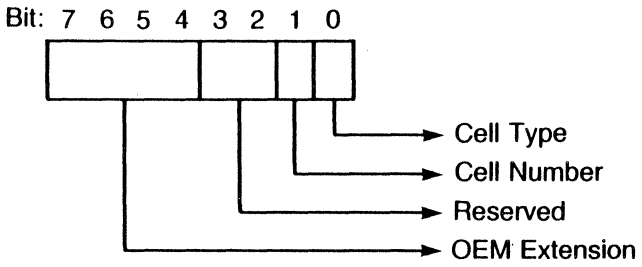


Figure 3-3. Extension Plane Byte Format

The Cell Type bit is 1 when characters are two cells long. Single cell characters are indicated by a 0 in this bit.

The Cell Number bit indicates if the corresponding character plane cell is the first or second cell of a two-cell character. If the value is 0, the cell is the first part of the character; if it's a 1, the cell is the second part. This bit is always 0 for single-cell characters.

The OEM Extension field is implementation-dependent and defines alternate character sets. The Console Resource Manager assumes the standard character set when this field is 0.

FRAME C Structure

The FRAME's C structure is as follows. Figure 3-4 illustrates this memory model.

```
struct FRAME
{
BYTE     *character,*attribute,*extension;
         /*Pointers to planes*/
WORD     nrow,ncol;
         /*Number of character rows and columns*/
WORD     use;
         /*Plane bit map*/
}
```

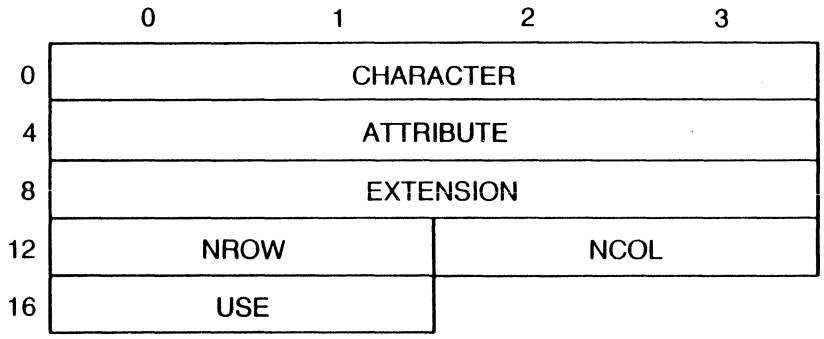


Figure 3-4. FRAME Data Structure Diagram

The FRAME fields are defined as follows:

- **character:** Address of FRAME's character plane
- **attribute:** Address of FRAME's attribute plane
- **extension:** Address of FRAME's extension plane
- **nrow:** Number of character rows in the FRAME
- **ncol:** Number of character columns in the FRAME
- **use:** A bit map indicating plane characteristics as follows:

  Bit 0: 1 - character pointer addresses a two-dimensional array
  0 - character pointer addresses a single byte

  Bit 1: 1 - attribute pointer addresses a two-dimensional array
  0 - attribute pointer addresses a single byte

  Bit 2: 1 - extension pointer addresses a two-dimensional array
  0 - extension pointer addresses a single byte

The FRAME's use field indicates if the plane consists of a complete two-dimensional array or a single byte. When the plane's bit value is 0, the Console Resource Manager applies the single byte's value to all bytes in the plane. Otherwise, the full array must be specified.

A FRAME is defined as either a **screen** FRAME or a **memory** FRAME. The screen FRAME is the console screen representation contained in the console file. You use the ALTER, COPY, or WRITE SVC to modify the screen FRAME, and modifications are immediately reflected on-screen. The memory FRAME is a data structure you create in the application's memory space and hence is not limited to modification by ALTER, COPY, and WRITE alone. Changes made to the memory FRAME are not reflected on-screen until they are COPYed to the screen FRAME.

Screen FRAME dimensions are indicated by the NROW and NCOL values in the CONSOLE table (see Figure 3-6). There are no restrictions except physical memory restraints limiting the size of a memory FRAME.

RECT C Structure

The RECT data structure defines a rectangular region of a FRAME. The point of reference is the FRAME coordinates of the region's upper lefthand corner. The region's width and height are specified within the data structure in terms of character rows and columns. The SVCs using the RECT structure specify which FRAME planes are included in the RECT. Figure 3-5 shows the RECT data structure diagram. The corresponding C structure is as follows:

```
struct RECT
{
        WORD    row,col,nrow,ncol;
/*              Top left corner FRAME coordinates
                and RECT width and height*/
}
```

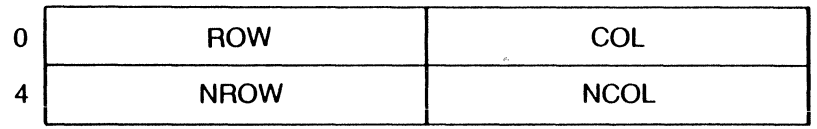


Figure 3-5. RECT Structure

The RECT fields are defined as follows:

- **row:** The row coordinate relative to the FRAME of the rectangle's upper lefthand corner
- **col:** The column coordinate relative to the FRAME of the rectangle's upper lefthand corner
- **nrow:** The number of rows (height) in the rectangle
- **ncol:** The number of columns (width) in the rectangle

3.2 Controlling the Console

Console attributes such as screen and keyboard modes, cursor location, and the number of character rows and columns are contained in the CONSOLE table. You manage the console screen on a FRAME basis with the ALTER and COPY SVCs and on a character basis with the WRITE SVC.

3.2.1 Console Attributes

The CONSOLE table is your reference source for information regarding console attributes and conditions. Figure 3-6 illustrates the CONSOLE table data structure. To get or set your process's CONSOLE table, use 0 or 1 or the stdin and stdout file numbers from the ENVIRON table as the GET or SET ID value

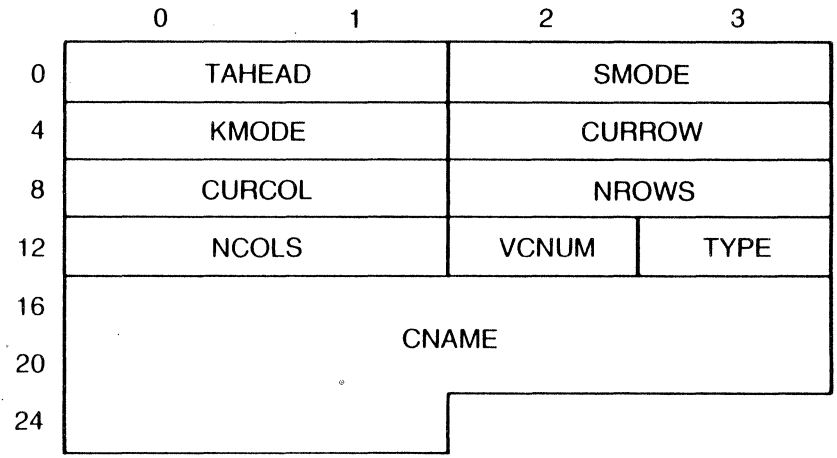


Figure 3-6. CONSOLE Table

SMODE and KMODE set the screen and keyboard modes, respectively. CURROW and CURCOL indicate the current cursor location. These values are initialized to 0 when the console is created. Set the mode options to select 8-bit or 16-bit characters, escape sequence decoding, and other features. Set CURROW and CURCOL to change the cursor position. The remainder of the parameters are read-only; their values determined by the physical console characteristics or established when the corresponding virtual console was created.

3.2.2 Manipulating the Screen

There are three ways to manipulate the console display: use ALTER to change a screen region, use COPY to copy one screen region to another, or WRITE to send a character, character string, or escape sequence. ALTER and COPY are also useful for character and string output, however, they cannot be used when console output is redirected to non-console devices.

Note: The window border files vcxxx/top, /bottom, /left, and /right are a special class of console file--only COPY and ALTER can be used to manipulate their contents. See 3.4 for the description of the border files.

Using ALTER and COPY

ALTER and COPY work on RECT structures to modify a FRAME. The FRAME can be a memory or a screen FRAME. The RECT can specify a FRAME region from single cell up to the entire FRAME itself. The ALTER form is as follows:

```
ret = s_alter(flags,fnum,dframe,drect,alterb);
```

Use ALTER's flags to select the character, attribute, and/or extension plane. To modify the screen FRAME, specify the console file number in the fnum field and set the dframe value to zero. To modify a memory frame, set the fnum value to zero and put the FRAME address in the dframe field. (Although 0 is the file number of the stdin file, the Console Resource Manager ignores the file reference.)

ALTER modifies the plane according to the two bytes in corresponding planes alterb argument. Alterb is an array of six bytes that determines the alteration of the destination frame.

Bytes 0, 2, and 4 in alterb are ANDed with each cell in, respectively, the character, attribute, and extension planes. Bytes 1, 3, and 5 are XORed with each cell in the same three planes.

COPY copies the contents of one rectangle to another. As with ALTER, each plane is individually selectable in the flag word. Source and destination RECT structures can be on the same or different FRAMEs and when on the same FRAME can overlap. The COPY form is as follows:

```
ret = s_copy(flags,fnum,dframe,drect,sframe,srect);
```

You distinguish memory from screen FRAMEs using specific combinations of fnum, sframe, and dframe. To specify the screen FRAME as the destination FRAME, put the console file number in the fnum field and set the dframe pointer to zero. To specify the screen FRAME as the source, set the sframe pointer to zero and specify the file number. To copy from one memory FRAME to another use a fnum value of 0 and enter the dframe and sframe addresses.

The source rectangle is described by the RECT at the srect address and the destination region by the RECT at the drect address. Rectangles do not have to be the same size. COPY clips the rectangles so that the region modified corresponds to the intersection of the srect and drect. Figure 3-7 illustrates how the excess is trimmed.

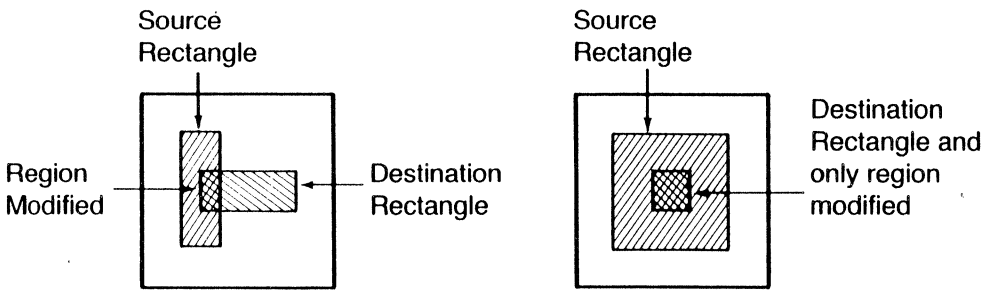


Figure 3-7. Examples of RECT Clipping

Using WRITE

The WRITE SVC sends the contents of the buffer to the console file specified by fnum. Use WRITE to send 8-bit escape sequences, 16-bit characters, and character strings to the console file. Each character output is placed at the current cursor position and the cursor position is updated. The screen scrolls automatically when the bottom line is reached.

The synchronous and asynchronous WRITE forms are as follows:

```
nbytes = s_write(flags,fnum,buffer,bufsiz,offset);
emask = e_write(swi,flags,fnum,buffer,bufsiz,offset);
```

The bufsiz value indicates how many bytes long the buffer is, not the number of characters in it. This is important when outputting 16-bit characters. Similarly, WRITE's return value (nbytes above) indicates the number of bytes, not characters, written. SMODE bit 1 determines if the Console Resource Manager outputs 8- or 16-bit characters.

Use an offset of zero when writing to a console file. Specify the offset relative to the end of file to accommodate redirection to non-console files. The flags and option values must be 0 for console writes.

The character string can contain displayable and non-displayable characters. The latter consist of 8-bit escape sequences, ASCII control sequences, and 16-bit character codes. Appendix A lists and describes the character sets and escape sequences supported by the Console Resource Manager.

3.3 Getting Console Input

Applications get console input from two sources: the keyboard and, if present, a mouse. The keyboard is accessed by reading stdin. The mouse is represented by a separate file. Mouse movement is automatically relayed to the screen without reading the mouse file. You use the mouse file to wait for a button state change--the BWAIT SVC--or for mouse form movement into or out of a RECT--the RWAIT SVC. The SET and GET SVCs are used to define the mouse form and determine its location.

3.3.1 Reading the Keyboard

There are two words in the CONSOLE table relevant to keyboard input: TAHEAD and KMODE. TAHEAD indicates how many characters are waiting in the type-ahead buffer. The KMODE word provides a variety of options including keystroke translation, character echo, 8-bit or 16-bit characters, and escape sequence decoding among others.

The READ SVC gets the characters from the console file's keyboard buffer and puts them in the buffer specified. READ might return fewer characters than requested; your program should be written accordingly. The READ forms are shown below. There are two synchronous forms: one for undelimited reads and one that allows delimiters to specify an end of string.

```
nbytes = s_read(flags,fnum,buffer,bufsiz,offset);
nbytes = s_rdelim(flags,fnum,buffer,bufsiz,offset,delimiters);
emask = e_read(swi,flags,fnum,buffer,bufsiz,offset);
```

Use an offset of 0 in your read calls and make it relative to the file pointer to accommodate redirection to a non-console file. The bufsiz specifies the end of the read event in terms of bytes read. Get the CONSOLE table's TAHEAD value to find out the number of characters waiting to be picked up from the keyboard buffer.

Delimiters let you set up conditions for terminating the read before the buffer is full and editing the character string. Set flag bit 1 to select a delimited read and bit 5 to use the editing characters. Use the READ return value to find the number of bytes read. Delimiters cannot be used with the asynchronous READ and you are limited to a READ buffer size of 256 bytes on delimited reads.

The delimiter specification is an address of a WORD array with two components. The first word is a number indicating the number of delimiters that follow. The remaining words are the delimiters themselves. Set the high order byte in each delimiter to 0 when the keyboard is in 8-bit mode.

The Console Resource Manager provides the line-editing characters listed in Table 3-4 when READ flag bit 5 is set. You can change these definitions with the XLAT SVC.

Table 3-4. Line-Editing Characters

Character	Action
LEFT ARROW	Moves cursor one character to the left
RIGHT ARROW	Moves cursor one character to the right
DELETE	Deletes next character
BACKSPACE	Deletes previous character
CTRL-B	Toggles cursor between beginning and end of line
CTRL-X	Erases from cursor to beginning of line

The Console Resource Manager compares each character read with each delimiting and editing character. READ returns with the number of bytes read when the buffer is filled or one of the delimiters is encountered. Use flag bit to select whether the delimiter is included in or excluded from the character string. When character echo is on (KMODE bit 5), the cursor is positioned at the beginning of the line just edited after a delimited read.

3.3.2 Monitoring the Mouse

Note: This discussion assumes that the mouse device was installed in the CONFIG.SYS configuration script. If it is not, your application can use the INSTALL SVC given the following conditions:

- The application must know the drive location and file name of the loadable mouse driver program.
- The application must have a user and group number of 0.

The INSTALL SVC is described in Section 7.

Mouse information and status is maintained in the MOUSE table. Figure 3-8 illustrates this data structure.

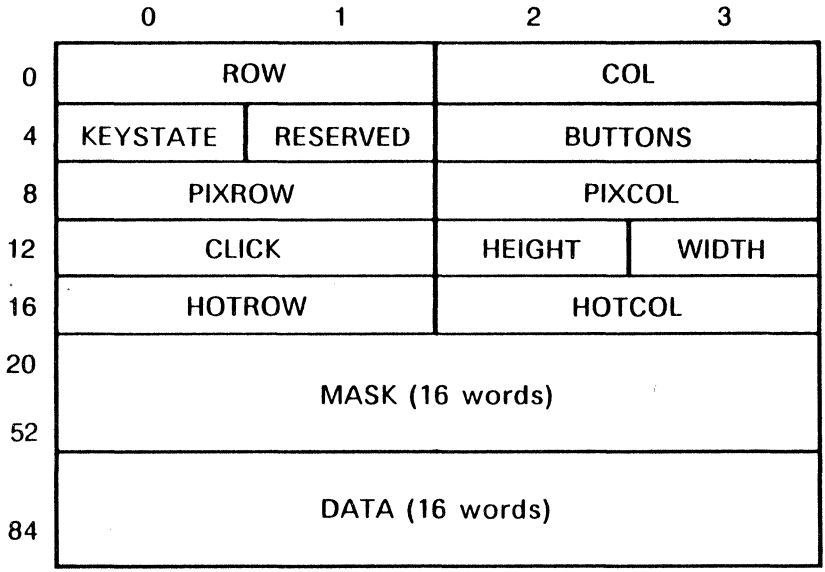


Figure 3-8. MOUSE Table

The PCONSOLE table also includes information on the mouse. See offset 1BH for the number of mouse buttons and offset 1CH for the mouse serial number. The mouse can have up to 16 buttons.

Mouse movement is automatically read from the device and shown on the screen by the mouse driver. Get the ROW and COL values from the MOUSE table to determine the mouse location; set these values to move the form independently of device input. The HOTROW and HOTCOL values set the hotspot--the point of reference within the mouse form. You can set these and all other MOUSE table values except the PIXROW and PIXCOL.

Opening the Mouse File

The mouse is opened by calling OPEN. In your OPEN call you specify the mouse name, the access privileges required, and the access mode. The mouse name is vcxxx/mouse where xxx is a decimal number indicating the current virtual console number. Get the virtual console number from the VCNUM field in your standard input file's CONSOLE table. (Call GET with an ID value of 0 to retrieve stdin's CONSOLE table.) For example, if the VCNUM value is 3, your mouse name would be vc003/mouse.

In your OPEN call, specify at least read access privilege. If you need to set the MOUSE table, request set access as well. For the access mode specify exclusive mode unless mouse access will be shared by multiple processes. In this case, specify shared, shared file pointer mode. Access is restricted to processes with the same family ID.

Your application should close the mouse file when you are done, otherwise you cannot close or delete the virtual console. CLOSE flag bit 0 has no meaning with respect to the mouse and is ignored.

Using BWAIT

Use the BWAIT SVC to monitor button state changes. BWAIT counts the number of times a specified mouse button condition occurs within a given time period. A button condition is defined by two criteria: buttons and their ON or OFF state.

The BWAIT form is as follows:

```
  ret = s_bwait(clicks,fnum,mask,state);
emask = e_bwait(swi,clicks,fnum,mask,state);
```

The fnum value is the file number returned when you OPEN the vcxxx/mouse file. The mask and state parameters are 32-bit values which define the mouse button condition.

You select buttons for the mask value by their position on the mouse. The rightmost button is represented by the least significant bit in the mask; the next button to the left is represented by the next bit, and so forth. To select the button, set its corresponding mask bit.

You define whether the button selected is to be ON or OFF in the state value. The Console Resource Manager looks only at the state bits corresponding to the buttons selected in the mask. Set the bit for ON.

As an example of the use of the mask and state fields, consider a two-button mouse. You can have the following button conditions:

1. The right button is pressed (ON) without concern for the state of any other buttons: mask = 1, state = 1.
2. The right button is pressed while the left button is not: mask = 3, state = 1.
3. The left button is pressed while the right button is not: mask = 3, state = 2.
4. The left button is pressed without concern for the state of any other buttons: mask = 2, state = 2.
5. Both buttons are pressed simultaneously: mask = 3, state = 3.

Use the clicks value to delimit the event by a specific number of incidences of the specified button condition. Any number of clicks can be specified, including 0. Use a click value of 0 to determine the mouse's current condition. BWAIT returns with a value of 0 when you specify 0 clicks and the mouse is in the condition defined in the mask and state.

BWAIT counts button conditions for a limited time period--the CLICK time limit specified in the MOUSE table. If the time period expires before the BWAIT click count is reached, the event terminates. The Console Resource Manager starts the timer upon the first incidence of the condition. Consequently, the count returned is always at least one except as described above.

Using RWAIT

RWAIT establishes an event boundary for the mouse. RWAIT returns with the row and column coordinates of the mouse's hotspot when it crosses the boundary. The RWAIT form is as follows:

```
position = s_rwait(flags,fnum,region);
   emask = e_rwait(swi,flags,fnum,region);
```

Set RWAIT flag bit 0 to clip the region to the current window borders. Otherwise, the region can include areas not visible on the parent screen. Flag bit 1 determines if the event occurs when the form exits or enters the region. The other flag bits are not used.

The region is a RECT structure confined to the calling process's virtual console's FRAME. The position value returned is 32-bits where the high order word indicates the row and the low order word the column.

3.4 Managing Virtual Consoles

For applications with multiple processes sharing access to the console and keyboard, it is often necessary or convenient to have a separate virtual console for each process. The key to these applications is a process--the window manager--which creates the virtual consoles, sets each window's size and position, and passes keyboard and mouse access from one process to another according to a planned transfer scheme. (These are basically the same functions as the FlexOS window manager supplied with the operating system.)

The window manager flow chart would include the following FlexOS functions; the SVCs used appear in parentheses.

1. Create a virtual console (CREATE).
2. Get the virtual console number (GET).
3. Set the virtual console's window size and location (SET).
4. Assign the console file to stdin, stdout, and stderr (DEFINE).
5. Define conditions under which keyboard and mouse ownership is returned (KCTRL and/or MCTRL).
6. Invoke shell or application that will use screen (asynchronous COMMAND).
7. Give keyboard and mouse ownership to the new virtual console (GIVE).
8. Read from your keyboard buffer (READ).
9. Reorder the virtual consoles to put the selected one on top (ORDER).

Steps 1 through 5 are repeated to create each virtual console. You have a numerical limit of 255 virtual consoles.

3.4.1 Creating the Virtual Consoles and Windows

To create a virtual console, you must specify the console screen on which it is to appear. This is called the parent screen. The virtual console created is referred to as a child console. Child consoles created on the same parent screen are referred to as sibling consoles. There are four rules of virtual console management based on these relationships:

- A child console always overlays its parent.
- Sibling consoles are "stacked" on the parent in the order of their creation until reordered by ORDER.
- The ORDER SVC only reorders a "stack" of sibling virtual consoles and cannot be used to put a parent on top of a child.
- An application always has access to its entire console regardless of its virtual console's position in the stack and the size of its window.

Figure 3-9 illustrates the parent, child, and sibling console relationships and the three rules. As shown in this figure, you can have multiple tiers of virtual consoles. As you change tiers, the parent/child relationships change. All virtual consoles on a given level are siblings.

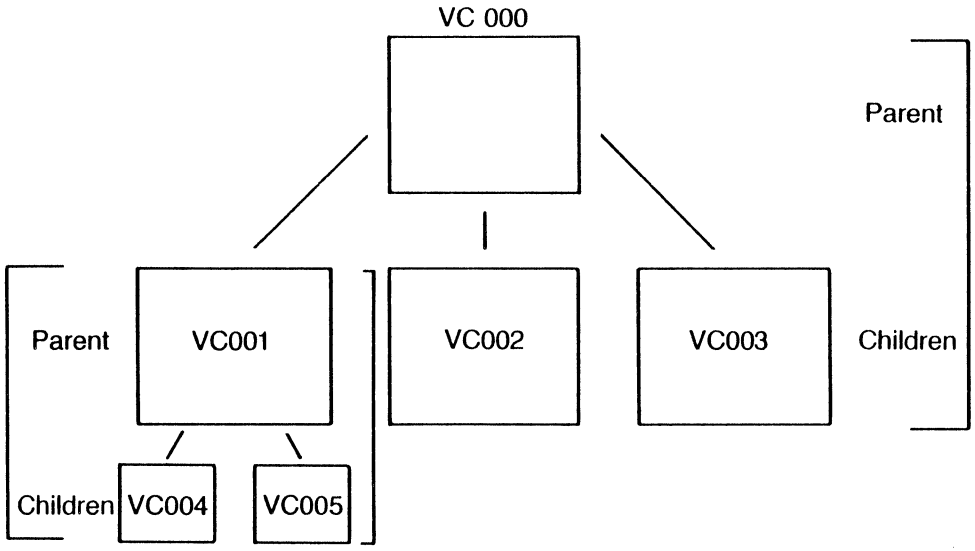


Figure 3-9. Virtual Console Relationships

Creating a virtual console requires write access to the parent console. The form of the CREATE SVC is:

```
fnum = s_vccreate(flags,fnum,rows,columns,top,bottom,left,right);
```

The flag bits select virtual console characteristics as follows:

- Whether the console and border are character- or bit-mapped
- Whether or not the parent's screen dimensions are used.
- Whether or not to keep the parent console contents in memory while the child console exists.
- Whether the virtual console is temporary (delete on last CLOSE) or permanent (delete only with DELETE).

For the fnum value, use the file number of the parent screen.

You specify the virtual console's dimensions in the rows and columns parameters. These become the ROWS and COLS values in the VCONSOLE table. The virtual console size is independent of the parent console's dimensions; you can, for example, create a virtual console larger than its parent. The top, bottom, left, and right parameters define window borders and are described in below.

CREATE returns the file number of your new virtual console file and automatically opens the file. Use this value as the ID number in your GET and SET calls to retrieve and modify the VCONSOLE table.

Virtual Console File Naming

The Console Resource Manager automatically names virtual consoles when you CREATE them. The name consists of the letters vc followed by a three digit decimal number corresponding to the VCNUM value from the virtual console's VCONSOLE table. For example, if the VCNUM is 10, the virtual console name is vc010.

A virtual console is composed of separate files representing the console (keyboard and screen), mouse, and window borders. Table 3-5 lists the names reserved for these files.

Table 3-5. Virtual Console File Names

File Name		Description
device:	vcxxx/console	Keyboard and/or screen file
	vcxxx/left	Left border of window file
	vcxxx/right	Right border of window file
	vcxxx/bottom	Bottom border of window file
	vcxxx/top	Top border of window file
	vcxxx/mouse	Mouse file

xxx = VCNUM, zero-padded left

Use the vcxxx/console file in your DEFINE call to assign the stdin, stdout, and stderr files to the virtual console's console file.

Be sure to define the files before you call COMMAND so that the files are automatically opened. The remainder of the files must be opened explicitly by the process before you can use them.

Windows

The term window refers to the view of the virtual console. When you create a virtual console, the window dimensions are initialized to 0 making the window a point with no height or width.

You set window dimensions in the VCONSOLE table's NROW and NCOL parameters. Set the VIEWROW and VIEWCOL parameters to position the window on the virtual console screen. Finally, set the POSROW and POSCOL parameters to position the window on the parent console's screen. Figure 3-10 illustrates these parameters.

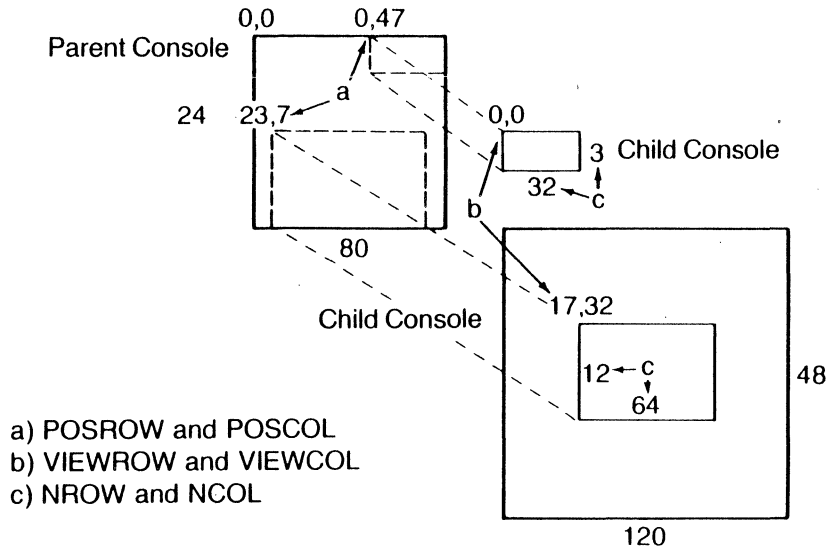


Figure 3-10. Virtual Console Characteristics

The VCONSOLE flag bit 1 gives you the option to have the window view adjust automatically to keep the cursor on-screen or remain fixed on a specific portion of the screen. Other flags determine how and when the view changes with respect to the cursor and freeze the window borders so you can make comprehensive changes to the border files without intermediate states appearing.

The window borders are contained in the vcxxx/top, /bottom, /left, and /right files. The Console Resource Manager creates these files when you specify top, bottom, left, and/or right parameters in your CREATE call. The top and bottom values set the height of the /top and /bottom border files only; the length is determined by the VCONSOLE table's NCOL value. The CREATE left and right values set the width of the /left and /right files; the height is set by the VCONSOLE NROW value.

3.4.2 Keyboard and Mouse Ownership

Keyboard and mouse ownership are always passed as a unit and only one virtual console can have ownership at a time. The window manager controls keyboard/mouse access by passing ownership to another virtual console with the GIVE SVC and specifying the conditions for its return with KCTRL or MCTRL. The conditions specified in KCTRL are keys or ranges of keys. With MCTRL, you specify a rectangle as the condition.

The KCTRL keys and MCTRL rectangles typically serve two purposes. First, they indicate that the user wants to change windows. Second, they indicate the user's choice of virtual consoles. When the user enters one of the specified window-control keys, that key and all subsequent keystrokes are sent to the parent virtual console's keyboard buffer. If there's a mouse keyboard entries a significant key is pressed or the mouse leaves the rectangle, the key and all subsequent keystrokes are sent to the window manager's keyboard buffer and mouse movement is updated in the window manager's MOUSE table. Use the information to determine which window to put on top with your ORDER call.

3.4.3 Deleting a Virtual Console

The virtual console's window remains on the parent screen until the file is closed or overlayed by a sibling virtual console. A partial CLOSE flushes the keyboard's type ahead buffer and, if the echo option (CONSOLE table KMODE bit 5) is selected, writes the buffer contents to the screen. A full CLOSE closes the file but leaves its contents intact unless it is the last close on a temporary console. In this case, the virtual console and all temporary files are deleted.

The Console Resource Manager only lets you delete a virtual console if it has no open /console, /mouse, /top, /bottom, /left, and /right files. Neither can you delete a virtual console with child consoles. If you try, an error message is returned. You can set CREATE flag bit 8 so that the virtual console is automatically deleted when the last of its virtual consoles is closed. Otherwise, use the DELETE SVC to remove the virtual console.

3.5 FlexOS Window Manager

WMEX, the window manager program provided with FlexOS, lets the user create, delete and switch virtual consoles. It also creates a message window you can use to interrupt the user and notify him or her that something has happened. You write to a reserved pipe to activate the window. When you write to this pipe, the message window overlays the current virtual console. You specify in your WRITE call to the pipe if a response is necessary.

To use the message window, you must open the file "wmessage." WMEX defines this file name to the message window's input pipe and waits for a message to appear there. In your OPEN call, specify the write privilege and the shared mode.

WMEX requires the display message to be preceded by a header. When you write to the pipe, format the contents of your WRITE buffer as follows:

UWORD	msgsiz	The total length of the message
LONG	pid	The writing process's process ID
BOOLEAN	rspflg	When true, indicates that a user response is expected; when false, no user response is allowed
UBYTE	rspname[10]	Name of the pipe in which WMEX should put the user's response (only necessary when rsplfg true)
UBYTE	message	The message to be displayed

The message itself can be 10 lines long. Each line must be terminated by a carriage return and line feed. The message is displayed as is.

If no response to the message is required, set the rspflg byte to false. The user enters a carriage return to remove the message window. If you want a response, set rspflg to true and give WMEX the pipe name to write the message to.

The response message is limited to one line in length and WMEX requires the user to enter a carriage return to terminate it. The carriage return is included in the string written to the pipe. If the message can be variable length, use the delimited READ call and specify the carriage return as the delimiter. WMEX removes the message window when the user enters the carriage return.

End of Section 3

Pipe Management

For two or more processes to communicate, a type of file known as a pipe is supported through a special device known as pi:. A file created on this device establishes a buffer used for the deposit and withdrawal of messages. Conceptually, pipe files have two ends, one to write into and the other to read from. Messages are deposited and withdrawn on a first in first out basis. Besides the pipe length, there is no limit to the number of messages you can store in a pipe at one time.

This section describes pipe management in the FlexOS operating system. Table 4-1 lists the pertinent SVCs.

Table 4-1. Pipe-related Supervisor Calls

SVC	Purpose
CLOSE	Close a pipe
CREATE	Create and open a pipe
DELETE	Remove a pipe
GET	Retrieve a pipe table
LOOKUP	Scan and retrieve pipe tables
OPEN	Open a pipe
READ	Read from a pipe
SEEK	Set or retrieve file pointer
WRITE	Write to a pipe

You cannot rename a pipe.

In all calls requiring a pipe name, you must precede the pipe name with the pi: device name or define a logical name that includes the pi: reference. Otherwise, the default: device is assumed.

4.1 Creating and Deleting Pipes

Use the CREATE SVC to make a pipe. The CREATE parameters are used as follows:

- Set the flags to request read, write, or delete privileges and the access mode. The privileges have the same meaning for pipes as they do for disk files. See 4.2 for the use of access modes with pipes. Flag bits 7 and 9 are meaningless with reference to pipes and are ignored.
- Put the address of your pipe name in the name field. The name itself is limited to eight alphanumeric characters.
- Set the record-size parameter to regulate the message blocks. For example, if a record size of four is specified, all pipe I/O is conducted in 4-byte blocks.
- Use the File Security Word to set the owner, group, and world access privileges.
- Set the size to the pipe buffer length. The size is independent of the message length but must be a multiple of the record-size.

The Pipe Resource Manager maintains a directory of all existing pipes. Each directory entry includes the pipe creator's user and group IDs and the File Security Word. The resource manager also makes a PIPE table for each pipe. PIPE table contents indicate the values set by the CREATE pipe call. Use LOOKUP and GET SVCs to retrieve PIPE tables. No special access privilege is required to lookup PIPE tables. However, you must have opened the PIPE to get its table. None of the values in the PIPE table can be set.

Use the DELETE SVC to remove a pipe. A CREATE option can be selected that automatically deletes a pipe on last close. If the pipe is being used solely to communicate between two or more processes for the life of the processes, the pipe is deleted automatically from the system when the processes terminate. This is because files are automatically closed on EXIT or ABORT.

4.2 Pipe Access

Processes must open the pipe before they can read from or write to it. When the OPEN call is made, the Pipe Resource Manager compares the user and group IDs of the calling process with those in the pipe's directory entry. This determines whether owner, group, or world access privileges are available. If both user and group IDs match, owner privileges are available; if only group match, group privileges are available. If there's no match, only world privileges are available.

The OPEN call succeeds when the access privileges requested either match or do not exceed the privileges available for the calling process's access level. If more privileges are requested, the OPEN succeeds or fails depending on the value of the OPEN call's reduced access flag. When this flag is set, the privileges granted are derived by ANDing the requested privileges with those available. Should none of the privileges match, the OPEN fails.

Pipe access privileges are also affected by existing access modes. The following rules govern the privileges available:

- A process's open access is never restricted by an open connection previously made by the same process.
- The read and write ends of a pipe are considered separate with respect to open restrictions. For example, an exclusive read open does not restrict a process from opening a pipe as shared write.
- Any exclusive open prevents other access requests to the same end.
- A shared open prevents other exclusive access requests but allows other shared requests to the same end.
- A shared file pointer request restricts pipe access to processes with the same family ID. All processes sharing the pipe must select the shared file pointer mode; a process that requests a different mode is denied access. For processes outside the family, the request functions as an exclusive request.

A pipe acts differently depending on whether an end is opened exclusive or shared mode. If one end of a pipe is opened in exclusive mode and then closed, a read or write attempt on the other end

results in an end-of-file (EOF) error. This is independent of how the other end was opened. If one end of a pipe has either never been opened, is currently opened, or the last open was in shared mode, the process accessing data through the opposite end waits until the operation is complete.

If one end of a pipe file is opened in shared mode and subsequently closed, FlexOS treats the file as if it were still open on the other end. Therefore, any process accessing it waits until the operation is complete. Note the distinction between shared mode and shared file pointer mode.

A pipe opened in shared file pointer mode is shared only by those processes with the same family ID (FID). After a pipe end opened in shared file pointer mode is closed by all of the processes, processes accessing the other end will receive an end of file error. This tells a process that the process on the other end of the pipe has either closed the file or terminated.

The use of modes to restrict access is consistent with spooler-type applications. For example, consider a spooler process which creates a pipe reserving for itself exclusive access to the read end. The write end is available for shared access by any process. Figure 4-1 illustrates this configuration.

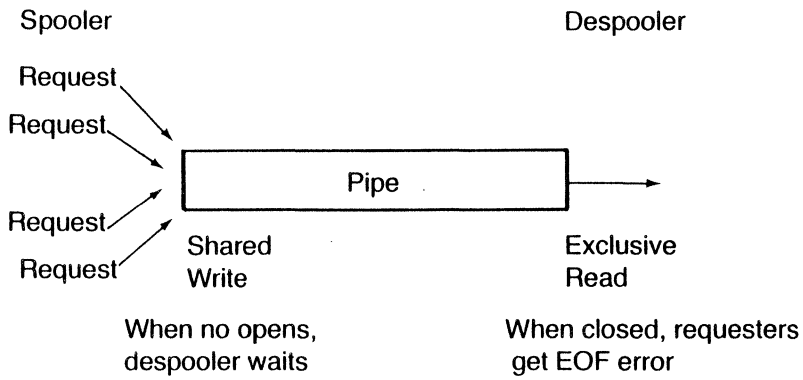


Figure 4-1. Spooler Pipe

Processes open and close the write end when they are sending files to the spooler. After the write end is closed, the despooler waits until the write end is opened by another process.

If the spooler closes the read end, processes attempting to write to the other end get an end of file error. This indicates to the writing process that there is no process at the other end that will read its file.

4.3 Interprocess Communication

Use the READ SVC to get data from the buffer and the WRITE SVC to put data in the buffer. The READ and WRITE flags and parameters are used in the same manner for pipes as they are for disk files and the file pointer is maintained. As many processes can participate in the exchange as you want.

The amount of data written to and read from the pipe can be independent of the pipe size. The following procedures are observed when the amount exceeds the size:

- On writes, the process waits when the pipe is full for another process to read data from the other end. The event completes when the reading process removes enough to compensate for the difference.
- On reads, the process waits when the pipe has been drained for another process to write data to the other end. The event completes when enough data has been written to compensate for the difference.

Pipes are often used to join two or more programs so one program's output becomes the input of another. To do this, a parent process would perform the following steps. The SVCs called are in parenthesis.

1. Create a pipe (CREATE).
2. Redefine stdin to be the name of the pipe (DEFINE).
3. Create the receiving process (COMMAND). The child inherits the parent's PROCDEF table, including the stdin prefix.
4. Reset stdin back to the original name (DEFINE)
5. Redefines stdout to the pipe name (DEFINE).
6. Create the source process (COMMAND). This child inherits the redefined stdout but, unlike the receiving child, has the original stdin.

When the two processes terminate, the parent process closes the pipe. If the parent terminates before the children, the pipe is automatically removed when the children terminate.

4.4 Synchronization and Exclusion

The Pipe Resource Manager lets you create pipes with a zero-length buffer size (bufsiz) for use as a simple semaphore. For semaphore pipes, a READ operation obtains the pipe and a WRITE releases it. If another process has obtained the pipe previously, the calling process waits until a WRITE to that pipe has been performed. WRITE operations, on the other hand, never wait; if the pipe was released previously, the call returns without error.

The process creating the semaphore pipe automatically owns it from the start. Although the Pipe Resource Manager keeps a record of who read the pipe, a WRITE by any process releases it. The process ID is maintained for two other reasons: First, so that a process can call multiple READs on a pipe it already owns and second, so the Pipe Resource manager can release pipes owned by a process that has terminated.

Use a semaphore pipe to regulate access to a resource not managed by the operating system. Any time a process wants to use the resource, it reads the pipe. If the pipe is already owned by another process, the calling process waits until another process releases it with WRITE. Upon return from the READ, the process is free to use the resource. Upon completion, the process writes to the pipe which releases the resource for other processes to use.

4.5 Nondestructive READ

The information stored in a pipe can be previewed using the READ SVC by setting flag bit 2. This allows a pipe to be used as a common data area among multiple applications. It also allows an application to preread a length field or message type field within a message and then read the complete message at a later time.

Note: Nondestructive reads can be dangerous if there are multiple readers of a pipe. It is the responsibility of the application to handle synchronization of pipe usage when there are multiple processes involved.

End of Section 4

Process Management

This section describes process creation, memory management, and process deletion. Table 5-1 lists the SVCs associated with these tasks.

Table 5-1. Process-related SVCs

SVC	Purpose
ABORT	Terminate a process or wait for a process to terminate
COMMAND	Create a process
CONTROL	Run a process under the control of another process
ENABLE	Enable software interrupts
EXCEPTION	Trap error conditions and jump to service routine
EXIT	Terminate a process with return code
DISABLE	Disable software interrupts
MALLOC	Add memory to heap
MFREE	Release memory from heap
SWIRET	Return from a software interrupt
TIMER	Delay process for specified time period
OVERLAY	Load overlay from a command file

The presentation belows describes how to use the ABORT, COMMAND, MALLOC, and MFREE SVCs. See Section 7 for the descriptions of the other SVCs listed in Table 5-1.

Three supervisor tables are pertinent to process management: the CMDENV, ENVIRON and PROCESS tables.

- CMDENV contains the command file specification and command tail from the process's spawning COMMAND call.
- ENVIRON contains the process's stdin, stdout, stderr, and overlay file numbers; user and group numbers; and identification numbers. A process's ENVIRON table contents are inherited from its parent's.
- PROCESS contains the process's identification number, user and group ids, name, current state, parent ID number, virtual console number, and memory allocation. Some PROCESS table values are set with the COMMAND SVC and while others are set by the Supervisor.

5.1 Process Relationships

A process executes program instructions independently of other processes. A process is constrained by a process environment maintained by the operating system. Environment characteristics include the process's logical address space (memory), CPU state and stack condition, user and group ID, and parent process. FlexOS uses these characteristics to manage the proces and protect it from other processes.

Processes are identified by a unique 32-bit process identification number--PID--and a name. The PID is assigned by the kernel when the process is created and remains assigned to the process until it terminates. The Supervisor splits running time for FlexOS processes on a priority basis. The recommended priority for user processes is 200. Processes with the same priority are allocated running time on a round-robin basis.

Besides the process ID, processes are also distinguished by a process family identification number--FID. When one process creates another, they keep the same FID unless you request a new number. Within a family, a process that creates another process is called the parent; the process created is called the child. Typically, the FID is used to distinguish processes running under different shells. That is, the shell is the head of the family.

5.2 Running a Program

Running a program has two steps:

- Loading an executable program file from disk into memory
- Assigning a process to execute the instructions

You use the COMMAND SVC to perform both steps. The process calling COMMAND must have the execute access privilege to the file.

Note: At the driver level, you can also use the PCREATE function to create a process. See the FlexOS System Guide for an explanation of PCREATE.

The COMMAND SVC searches the disk for the command file specified, opens it, and loads it into memory. Running a program does not require the creation of a new process. COMMAND gives you the following options.

- **Run the program as an independent (child) process:** This option creates a new process ID for the program. Child processes get their own memory allocation and execute FlexOSly with the other process.

- **Run the program as a procedure:** This option runs the program as a subroutine of the calling process; no new process ID is assigned. The calling process's memory allocation is supplemented by the new program's specification. When a procedure program exits, control is returned to the next executable statement in the parent process. You must use the synchronous form of COMMAND to use this option.

- **Chain the program to the current program:** This option runs the program as a subroutine within the context of the calling process; no new process ID is assigned nor memory allocated. However, the process never returns to the previous program. The chained program's memory requirements overlay the process's existing allocation. When the chained program exits, the process terminates. You must use the synchronous form of COMMAND to use this option.

When you call the synchronous form of COMMAND, the call does not return until the program exits or the process is aborted. When you use the asynchronous form of COMMAND, the event mask is returned and the child's process ID is recorded at the &pid address you specify in your COMMAND call. The event completes when the child process terminates.

Unless externally aborted, a process executes the program instructions up to and including the EXIT call. When the Supervisor receives the EXIT call, it closes all open files belonging to the process and cancels any outstanding events. This completes the COMMAND event. For the first and third COMMAND options described above, the process ID is then released along with the process's memory; for the second option, the process returns to the next instruction in the previous program.

5.3 Process Termination

The synchronous form of ABORT SVC terminates the process specified. This terminates the COMMAND event; all files belonging to the process are closed, outstanding events cancelled, and memory released.

The asynchronous form of ABORT is useful as a self-preservation measure for processes aborted externally. For example, consider the user who enters a control-C to terminate his or her program. For many applications, it is preferable to return to a previous location in the program rather than abort the program entirely.

To trap the control-C and force the return to a specific location in the program, call the asynchronous form of ABORT; use a process ID of 0 (this indicates current process) and include a software interrupt (swi) pointer. In your software interrupt, use the SWIRET option which keeps the process ID in the swi and then call a jump instruction to return to the program location.

Remember that the stack is in an unknown condition when you make the jump. At the return point in your program you should include instructions to mark the stack frame so it is restored to a known condition.

5.4 Memory Management

You use the MALLOC and MFREE SVCs to manipulate a process's memory allocation. Only the heap portion can be modified: MALLOC extends the heap space or adds a new heap and MFREE releases allocated heap memory back to the kernel for subsequent allocation.

To add contiguous memory to an existing heap, select MALLOC's expand option. In your MALLOC call you also give a pointer to a buffer indicating the minimum and maximum amount of memory and the base address of the heap to expand. Get this address from the process's PROCESS table. The kernel adds as much as can be allocated within the parameters given and returns the new block's starting address and the number of bytes allocated in your buffer. The original heap base address and contents are not affected.

To get a new, independent heap select MALLOC's allocate option. The new memory block may or may not be contiguous with an existing heap segment, depending upon current system memory usage. As above, you pass a buffer pointer where a minimum and maximum amount of memory is specified. You do not need to specify a starting address, however. The heap's base address and its actual size are returned in the buffer. When you add a new heap and it is contiguous with an existing heap, the existing heap becomes a fixed data area.

End of Section 5

Miscellaneous Resource Manager

This section describes the SVCs used for device management through the Miscellaneous Resource Manager. All devices except for disk drives, consoles, mouses, and network controllers are controlled by the Miscellaneous Resource Manager. This includes such devices as printers, plotters, modems, and custom, OEM-implemented peripherals. The term device is used in this section as a generic expression to refer to all miscellaneous devices. Table 6-1 lists the SVCs.

Table 6-1. Miscellaneous Device Control Supervisor Calls

SVC	Purpose
CLOSE	Close a device
DEVLOCK	Lock device from access by other processes
GET	Retrieve a device table
SET	Set device table values
OPEN	Open a device
READ	Read from a device
WRITE	Write to a device
INSTALL	Install the device driver or subdriver
SPECIAL	Send data to or receive data from driver

6.1 Device Tables

The Miscellaneous Resource Manager maintains four device-related tables.

- DEVICE: Scan the DEVICE table to get the logical port and printer names. The Miscellaneous Resource Manager manages all devices with type numbers 7xH and 80-FFH.

- PRINTER: Use this table to get and set printer parameters. You cannot get or set a printer's table until you have opened the device.
- PORT: Use this table to get and set I/O port parameters. You cannot get or set a port's table until you have opened it. Ports linked to a driver cannot be accessed with GET and SET; use the SPECIAL functions instead.
- SPECIAL: Devices not adhering to the PRINTER or PORT table models have SPECIAL tables. SPECIAL table contents are OEM-defined.

FlexOS reserves the name prn: for the system list device. Unlike stdin, stdout, and stderr, prn: does not have a reserved file number. Your program must open the prn: device, write data to it, and then close the device.

Note: The following description of device I/O assumes the device driver is already installed. If it is not or if your software must establish a driver-to-subdriver link, section 6.3 below reviews device installation. See the FlexOS System Guide for additional information on drivers.

6.2 Device Access

Unlike files, devices are installed and removed rather than created and deleted. Like files, you must open devices to access them. To indicate the device, you its name in the OPEN call. The access privileges and modes characteristic of disk files and pipes also apply to devices. Like pipes and consoles, read and write access privileges are treated separately and can be implemented with different modes.

6.2.1 Opening and Closing

Use the OPEN SVC to open the device. Set the flags to select the access privileges and modes. The privileges and modes supported are determined by INSTALL options selected and the device driver itself. The Miscellaneous Resource Manager compares the privileges requested with those available. Set flag bit 7 if you can accept reduced access. You must set flag bit 0 if you want to set the table values.

Note: The devices with PORT tables cannot be used for data I/O; access to these devices is limited to getting and setting PORT table values.

The OPEN call returns the file number used for all subsequent device access. The file number is also used as the ID number in GET and SET calls and the file number for your CLOSE call. When you close the file, the write buffer contents are output to the device.

After the device is opened, you can get and set table options. Devices are process independent; table variables are not initialized or modified as processes open and close the device. Thus, the PRINTER table typeface mode or PORT table baud rate selected by one process remains set for other processes.

6.2.2 Security

Security options come in two forms: access modes and device locking. The access modes are selected in the OPEN call. If multiple, related processes need to share access to the device, set flag bit 6 to shared file pointer. Although the file pointer may or may not be meaningful, this is the mechanism used to limit device access to those processes in the same family.

The DEVLOCK SVC can also be used to restrict access. This feature is only valid if the INSTALL option allowing DEVLOCK was selected. DEVLOCK options let you limit access to the process or the process family. The lock is removed explicitly using DEVLOCK or implicitly when the process terminates.

6.2.3 Data I/O

Use the READ and WRITE SVCs to get data from and to the driver. The file pointer offset may or may not be meaningful. Although the Miscellaneous Resource Manager does not maintain a file pointer, it does pass the offset to the device. Consequently, you can use an offset if the device supports random I/O. The SEEK SVC is not supported by the Miscellaneous Resource Manager. However, the function not implemented error is not returned if you call it. Instead, a zero is returned. (This is done so redirection to a miscellaneous device does not cause an error on seeks.)

All WRITE flag options are supported.

All READ flag options except the edited read (flag bit 5) are supported by the Miscellaneous Resource Manager. This includes the use of delimiters to complete the read event. As with consoles, your program should be able to accept fewer characters on a read than requested.

SPECIAL functions are another way to transfer data to and from a device. However, all SPECIAL functions are driver-dependent; there are no generic functions. The Miscellaneous Resource Manager acts as a conduit for SPECIAL calls and provides no services beyond transferring the SPECIAL databuf and parmbuf contents.

6.3 Device Installation

Device drivers are installed with the INSTALL SVC. The driver can be read from a disk file or replicated from an existing driver. Only privileged users can call INSTALL.

FlexOS supports the concept of subdrivers. This allows a driver unit to be independent of specific hardware by accessing the hardware in a generic way. For example, multiple units of an RS-232 subdriver can be installed and then linked to printer, network, or communications drivers. This relieves the driver writer of hardware dependent code and provides flexibility when installing add-on or custom peripherals.

6.3.1 Driver and Subdriver Installation

Although drivers and subdrivers are usually installed when the system is loaded, they can be installed after the installation script has been performed as well. (See the CONFIG.SYS description in the FlexOS System Guide for description of the installation script.) You can also uninstall drivers, disconnect a subdriver from a driver, and link a subdriver to a driver dynamically.

Once a driver-to-subdriver link is established, the subdriver is no longer individually accessible; the driver owns it. When the link is dissolved, the subdriver is available for linking to another driver. The Miscellaneous Resource Manager manages subdrivers until they are linked to a driver. Then the driver assumes subdriver management responsibilities. Subdrivers can themselves have their own subdriver.

Note: The subdrivers with PORT tables do not have a standard interface to the resource manager. Do not attempt to access these drivers directly.

6.3.2 INSTALL Options

Drivers and subdrivers are installed with the INSTALL SVC. There are four INSTALL options.

- **Load driver from the disk:** Read the driver from the specified disk file, load it into the system space, call the initialization routine for the first unit, and give it a logical name.
- **Add a unit to an existing driver:** Replicate an existing driver in system space, initialize the device, and give it a logical name.
- **Link one driver to another:** Make one driver the subdriver of another. Both drivers must already be installed.
- **Uninstall the driver:** Remove the device driver and release subdriver.

Each unit installed manages a separate device; for example an RS-232 port. Multiple units derived from the same driver are distinguished by unique names, however, they all share the same code.

The driver determines the maximum access privileges supported by the device and the access modes. The maximum access modes are determined when the device is installed.

You do not use the DELETE SVC to remove a driver. Instead, use INSTALL's uninstall option. Only the device specified is removed; if the driver had a subdriver, the subdriver is released and becomes available for direct access or linkage with another driver.

6.4 PORT Table Modification

PORT table devices cannot be accessed directly; they can only be used as subdrivers. You can, however, set PORT table values. There are two ways to do this; depending on whether the device has driver or subdriver status. To determine the device status, look at the INSTAT word in its device table.

A device that is not a subdriver (it as not been linked to another driver with INSTALL), can be opened directly. In your OPEN call, request only the set access privilege (flag bit 0). Use the file number returned as your GET and SET ID.

Devices that are subdrivers cannot be accessed directly. However, you can use SPECIAL functions 13H and 93H to get and set the PORT table values. These SPECIAL functions are described after the SPECIAL disk functions in Section 7.

To use the SPECIAL functions, you must know the driver that owns the subdriver. The OWNERID value from the subdriver's DEVICE table is the significant 16 bits of the subdriver's owner's DEVICE table key value. Use this value in a LOOKUP call to find the device name. If the owner is also a subdriver, repeat this procedure to get the owner. When you determine the owner, open the device and use the file number returned in your SPECIAL calls.

End of Section 6

Supervisor Call Descriptions

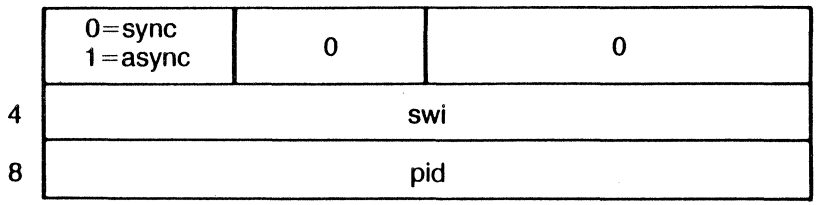
This section describes the FlexOS SVCs. The descriptions are presented alphabetically by SVC name and contain explanations of each call's arguments and return codes. See Section 1 for the description of the C and assembler interface conventions used in the descriptions. Appendix B lists the error return codes.

The descriptions also include a general explanation of the function's purpose and effect. For specific information regarding when and how to use the SVCs, refer to the chapters describing disk, console, pipe, process and special device management.

7.1 ABORT

C Interface:

```
LONG        pid;

ret = s_abort(pid);
emask = e_termevent(swi,pid);

ret = __osif(F_ABORT,&parmblk);

parmblk:
```



Parameters:

swi	Address of a software interrupt routine
pid	Process ID of target process to abort or to wait to terminate. Use 0 to specify calling process.

Return Code:

ret	Error Code

Description: The synchronous ABORT SVC removes a process from the system. Any outstanding events for that process are canceled, opened files are closed, and memory is released. Performing an ABORT on the calling process is equivalent to an EXIT with a return code of E_ABORTED.

A process can only be aborted by another process with the same user and group or by a superuser.

The asynchronous version of ABORT does not terminate the target process. Instead, it makes the target's termination an event. Specify a process ID (pid) of zero to have the program trap a control-C entered by the user or any other external abort. Use the software interrupt (swi) to control program flow from there.

The return code from ABORT reflects only the operating system's attempt to notify a process to terminate. If the process has a termination swi that does not perform an exit, ABORT's return code indicates success, but the process will still be running.

7.2 ALTER

C Interface:

```
UWORD       flags;
LONG        fnum;
FRAME       *dframe;
RECT        *drect;
BYTE        alterb[6];

ret = s_alter(flags,fnum,dframe,drect,alterb);

ret = __osif(F_ALTER,&parmblk);
```

parmblk:

0	0	0	flags	
4	0			
8	fnum			
12	dframe			
16	drect			
20	and0	xor0	and1	xor1
24	and2	xor2	RESERVED	

Parameters:

flags	Bit map of planes to alter. bit 0: 1 = Alter character plane 0 = Do not alter character plane bit 1: 1 = Alter attribute plane 0 = Do not alter attribute plane bit 2: 1 = Alter extension plane 0 = Do not alter extension plane bits 3-15 are reserved.
fnum	Console screen or border file number; use 0 to specify a memory FRAME
dframe	Address of FRAME to affect; use NULLPTR to indicate screen or border specified by fnum.
drect	Address of RECT specification indicating portion of FRAME to alter
and0	alterb[0] = character plane AND
xor0	alterb[1] = character plane XOR
and1	alterb[2] = attribute plane AND
xor1	alterb[3] = attribute plane XOR
and2	alterb[4] = extension plane AND
xor2	alterb[5] = extension plane XOR

Return Code:

ret
Error Code

Description: ALTER changes a rectangular area of the specified FRAME. The Console Resource Manager changes each cell in the planes selected according to the AND and XOR values you specify for that plane. The rectangular area is defined by a RECT structure. You select the planes in the flag word.

The FRAME can be a screen or memory FRAME. To specify a screen FRAME put its file number in the fnum field and a null pointer in the dframe field. To specify a memory FRAME, put a 0 in the fnum field and the FRAME's address in the dframe field.

The following table lists the AND and XOR bit values used to modify the destination byte or combine it with another value.

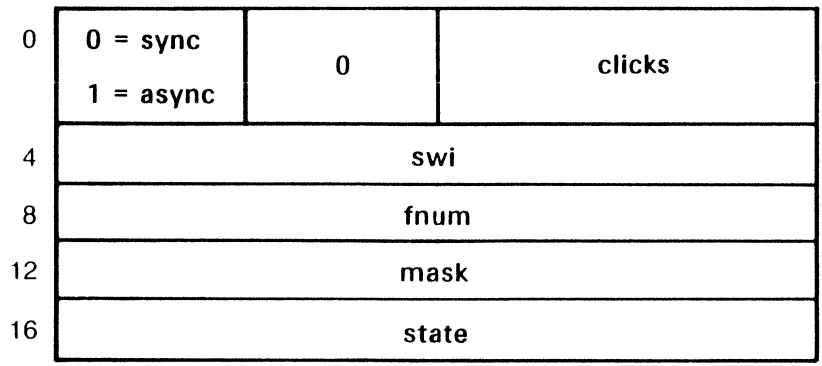
Action Performed on Bit in Destination Byte	Bit in AND Byte	Bit in XOR Byte
Clear	0	0
Set	0	1
As is	1	0
Complement	1	1
Logical Operation with Data and Bit in Destination Byte		
Load Data	0	data
AND Data	data	0
XOR Data	1	data
OR Data	NOT data	data

7.3 BWAIT

C Interface:

```
UWORD       clicks;
LONG        mask;
LONG        state;

ret = s_bwait(clicks,fnum,mask,state);
emask = e_bwait(swi,clicks,fnum,mask,state);

ret = __osif(F_BWAIT,&parmblk);
```

parmblk:



Parameters:

clicks	Number of times the mouse enters this state within the "click interval" set up in the MOUSE Table after this call is made. If **clicks** is 0 and the mouse is already in this state, the event is already complete.
fnum	Mouse file number
mask	Bit mask of buttons to consider. The lowest order bit is set if the first mouse button to the left is to be considered. The second lowest bit corresponds to the second button from the left. A total of 16 mouse buttons can be supported in the low word of **mask**.
state	Bit mask of buttons that define the button state given the mask that determines the buttons to ignore all together in the low word of **state**.

Return Code:

ret	Number of Clicks Error Code

Description: The BWAIT SVC allows the calling process to wait until a mouse button state is reached. The **mask** determines the number of mouse buttons the calling process wants considered. For example, by setting the mask appropriately, a one button mouse can be expected when there is more than one button.

The **clicks** field allows the calling process to receive multi-click mouse input. When a user presses a mouse button, releases it and presses it again within the "click interval", the mouse has been double clicked.

If **clicks** is set to two, and a second click is not performed within the "click interval", the event is considered complete. The return value indicates the number of clicks actually performed. If **clicks** is set to zero, BWAIT returns a zero if the button state is already in the specified state. Otherwise, it returns one upon the first entry to the state.

The "click interval" is changed in the MOUSE table through the SET SVC.

7.4 CANCEL

C Interface:

```
LONG        dmask;
LONG        events;

dmask = s_cancel(events);

dmask = __osif(F_CANCEL,events);
```

Parameters:

events	Logical OR of event masks to be canceled

Return Code:

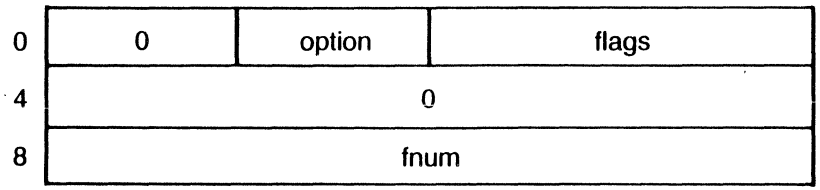
dmask	Bit map of events that could not be canceled because they have already completed

Description: The CANCEL SVC terminates one or more specified asynchronous SVCs. The events argument is the logical OR of the event masks you want to cancel. The dmask return code indicates events that, although requested for termination, had already completed. Use the RETURN SVC to get the return codes for these events so the event bits can be reused.

7.5 CLOSE

Interface:

```
UWORD       flags;
BYTE        option;
LONG        fnum;

ret = s_close(flags,fnum);

ret = __osif(F_CLOSE,&parmblk);

parmblk:
```



Parameters:

option	May be used by SPECIAL devices.
flags	bit 0: 1 = partial close (flush only) 0 = full close bits 1-15 are reserved.
fnum	File number of file to be closed

Return Code:

ret
Error Code

Description: CLOSE disassociates an I/O stream from a file number. Before the close is performed, all outstanding asynchronous I/O is completed and locked file areas are unlocked. If a device error occurs on a full close, the file is closed making the file number invalid.

For all types of files, a partial close flushes the associated I/O buffer but leaves the file open. For disk files, a partial close updates the directory.

For disk and pipe files and directories created with the temporary flag set, the last close deletes the files. This also applies to a file marked temporary because an attempt has been made to delete it while any process had it open. In this second case only, the closing process gets an error return code rather than success to indicate that the close also resulted in a file delete.

WARNING: CLOSE with the "full close" option always disconnects an open file from the calling process regardless of error. This can cause a failure to flush intermediate buffers to media if the error is a physical error.

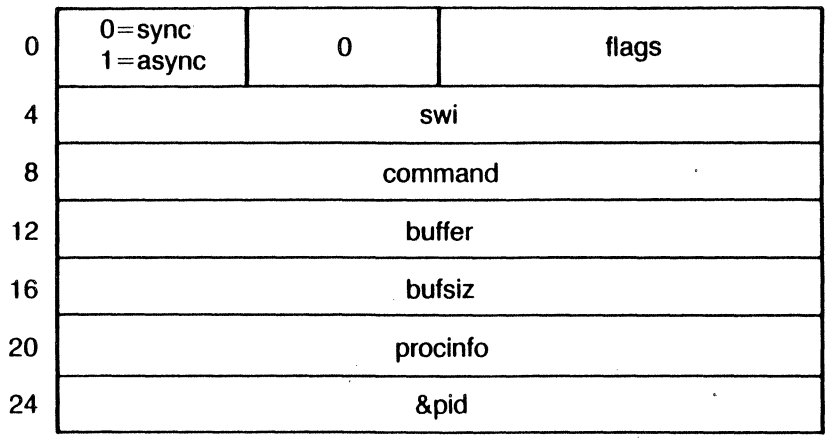
Specifically, if a floppy drive door is open at the time of a close, the final flush does not occur. An application can avoid this problem by performing a "partial close" to flush all intermediate buffers. If an error is returned indicating an "open door", the application can warn the user to replace the media and close the door before attempting the close operation again.

However, if any other activity occurs on the device from the time the door was originally opened, the "partial close" approach fails since all intermediate buffers have been discarded. In such a case, the application must assume the file has not been updated since the last successful partial close. After performing a successful partial close, the application can perform a full close to disassociate the file from the process.

7.6 COMMAND

C Interface:

```
UWORD   flags;
LONG    pid,bufsiz;
BYTE    *command,*buffer;
PINFO   *procinfo;

ret = s_command(flags,command,buffer,bufsiz,procinfo);
emask = e_command(swi,&pid,flags,command,buffer,bufsiz,procinfo);

ret = __osif(F_COMMAND,&parmblk);

parmblk:
```



Parameters:

flags bits 0–3 are reserved

bit 4: 1 = No new process (set bit 5 to 1)
0 = New process (ignore bit 5)

bit 5: 1 = Chain
0 = Not implemented (returns E_IMPLEMENT error)

bit 6: 1 = Do not release memory on termination of procedure if procedure uses the EXIT SVC with the stay resident flag set.
0 = Not implemented (returns E_IMPLEMENT error)

bit 7: 1 = Assign a new process family ID (FID)
0 = Keep the current process family ID (FID)

bits 8–12 are reserved (must be 0).

bit 13: 1 = Force case to media default
0 = Do not affect name case

bit 14: 1 = Literal command
0 = Prefix substitution allowed

bit 15: reserved (must be 0)

swi Address of a software interrupt routine

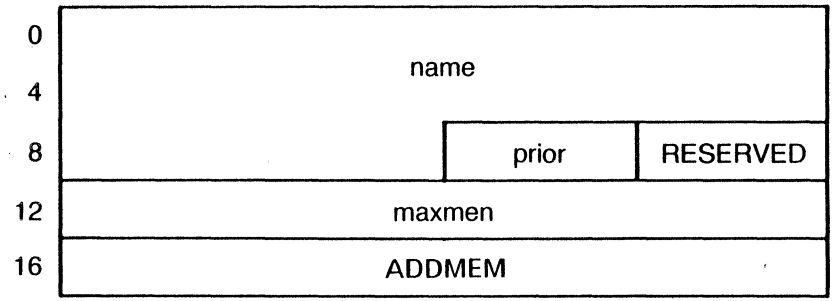
command Address of 128-byte, null-terminated string indicating the name of the loadable file.

buffer Address of a variable length buffer containing a 128-byte, null-terminated command tail and special information to be passed to the new process. (At most, the command tail can be 127 characters and one NULL byte long.) COMMAND puts the tail in the CMDENV table. Data after the first 128 bytes is put in the process's heap.

The PROCESS table contains the heap address and size. Use this buffer area to pass an environment string, common data, or special information to the program.

bufsize — Size of buffer in bytes

procinfo — Address of the PINFO table. PINFO must be constructed as follows:



20 = Length in bytes

name: Process name

prior: Process priority (user processes are usually set to 200)

maxmem: Maximum memory this process can own (larger minimum requirements specified by the command file supercede this amount)

addmem: The amount of memory to be added to the minimum amount specified by the command file (FlexOS allocates the greater of the two values: maxmem or the sum of the command file's specified minimum plus addmem)

pid — Address of new process ID. COMMAND puts the new process's 32-bit PID at this location when flag bit 4 equals 0 and COMMAND is called asynchronously.

Return Code:

ret — Process completion status:
High order word = 0
Low order Word = return status (negative if error)

Error Code: Indicates program load failure

Description:

The COMMAND SVC creates a new process or chains a new program to the calling process. The value of flag bit 4 determines which action is taken. When flag bit 4 is set, use flag bit 7 to specify whether you want the current process family ID kept or a new one assigned.

The return code is a long value with two components: if the high order word is zero, the low order word contains an utility return code. See Appendix B for a list of utility return codes. If the high order word is a negative number, the low order word contains an operating system error code. A return code can be used in batch files as an argument in the IF ERRORLEVEL statement.

When COMMAND is called synchronously, the return code is provided when the EXIT SVC is called by the program. When COMMAND is called asynchronously and a new process is requested, an event mask (emask) is returned and the new process ID is stored at the location indicated by pid.

The chain option causes the calling process's current program to be overlaid with the new program. The process ID does not change.

The COMMAND SVC opens the specified command file in EXECUTE mode without accepting reduced access. Any error in the attempt to open the file returns the file not found error.

Priority of 200 is the recommended number for user processes. Higher numbers have lower priority; lower numbers have higher priority.

7.7 CONTROL

C Interface:

```
UWORD       option;
LONG        pid,target,bufsiz,tpid;
BYTE        *buffer;

ret = s_control(option,pid,buffer,bufsiz,target,&tpid);
emask = e_control(swi,option,pid,buffer,bufsiz,target,&tpid);

ret = __osif(F_CONTROL,&parmblk);

parmblk:
```

0	0=sync 1=async	0	option
4	swi		
8	pid		
12	buffer		
16	bufsiz		
20	target		
24	&tpid		

Parameters:

option	0 - Invalid 1 - Load Program for control 2 - Delete Program 3 - Read Target Code Memory 4 - Read Target Data Memory 5 - Write Target Code Memory 6 - Write Target Data Memory 7 - Read Target Registers 8 - Write Target Registers 9 - Go 10 - Single Step (Trace) 11 - Reserved (Force Halt) 12 - 13 Reserved (All Exception Traps ON,OFF) 14 - Select Exception Trap ON 15 - Select Exception Trap OFF 16 - 255 are reserved
swi	Address of a software interrupt routine
pid	For option 1, a pointer to command file specification; for options 2-15, the process ID of the target process
buffer	Address of buffer in calling process's address space; the purpose of this buffer depends on the option selected.
bufsiz	Size of buffer in bytes
target	Address in controlled process's address space: the purpose of this buffer depends upon the option selected.
&tpid	Target process address: tpid is used only with option 1; see the first section of the chip supplement to this book for the description of its use.

Return Code:

ret	Error Code

Description: The CONTROL SVC controls the execution of one or more child processes. Use the CONTROL options to select debugging functions such as setting breakpoints, modifying memory or registers, and starting and stopping process execution. The use of the pid, buffer, target, and &tpid arguments depends upon the option selected.

Option 1--Load: Use this option to create the target process. The pid, buffer, bufsiz, target, and &tpid arguments have the same purpose as the command, buffer, bufsiz, procinfo, and &tpid arguments in the COMMAND SVC. CONTROL opens the specified program (command file) in Execute, Read mode with no reduced access. When called synchronously, the load option returns when the program is loaded; the return code indicates the success or failure of the operation. Similarly, the asynchronous CONTROL load event is complete when the program is loaded. Use the RETURN SVC to get the return code.

Option 2--Delete: Use this option to terminate the program. pid specifies the target process to terminate.

Options 3 and 4--Read target code or data memory: Use these options to transfer a portion of the target process's memory to the calling process's memory. pid specifies the target process, the buffer address points to the calling process's destination buffer area, and target contains the logical address in the target process from which to begin the transfer. The bufsiz value indicates the number of bytes to be transferred.

Options 5 and 6--Write target code or data memory: Use these options to transfer a portion of the calling process's memory to the target process's memory. pid specifies the target process, buffer contains the pointer to the calling process's source buffer, and target contains the first logical address of the target's destination buffer. The bufsiz indicates the number of bytes to be transferred.

Options 7 and 8--Read and write target registers: Use these options to transfer the target process's register data to or from the calling process's buffer. pid specifies the target process and buffer contains the pointer to the destination or source buffer.

Option 9--Go: Use this option to commence execution of the target program. Execution proceeds until one of the following conditions is encountered. The number shown in parenthesis indicates the condition's return code.

- Error code on CONTROL request (<0)
- Target process EXIT (0)
- Target process exception (>0): This condition exists when break point set by CONTROL option 14 or by the EXCEPTION SVC is encountered.
- Target about to be aborted through an external ABORT or Control-C (512): 512 is the return code for the COMMAND SVC when using the go option.

Option 10--Trace: Use this option to step through the target program one instruction at a time. pid specifies the target process and bufsiz must be 1. The return code is the same as the Go option. Resume execution with Go or another Trace.

Options 14 and 15--Trap ON and OFF: Use option 14 to set target program break points at exception handling routines and SVCs; use option 15 to have break points ignored. pid specifies the target process and the target value contains a buffer pointer indicating the exception numbers to break on (see EXCEPTION). When an exception set by the target program is reached, target program execution proceeds with the interrupt service routine (isr). When an exception set by CONTROL is reached, target program execution stops and control returns to the calling process.

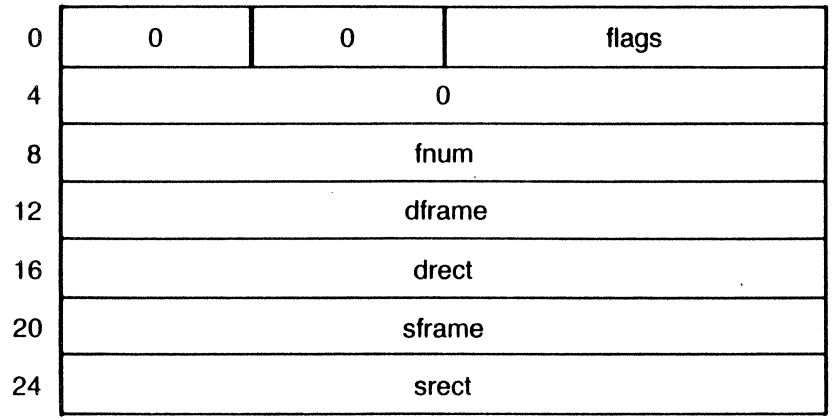
You set break points by writing a "break point" instruction into the target buffer, turning the break point exception trap on, and using the Go option. A return code greater than 0, where the number indicates the exception number, indicates that a break point has been encountered. To proceed, restore the target process code and set the instruction pointer to the break point location.

7.8 COPY

C Interface:

```
UWORD     flags;
LONG      fnum;
FRAME     *sframe,*dframe;
RECT      *srect,*drect;

ret = s_copy(flags,fnum,dframe,drect,sframe,srect);
ret = __osif(F_COPY,&parmblk);
```

parmblk:



Parameters:

flags — Bit map of planes to copy

bit 0: 1 = Copy character plane
0 = Do not copy character plane

bit 1: 1 = Copy attribute plane
0 = Do not copy attribute plane

bit 2: 1 = Copy extension plane
0 = Do not copy extension plane

bits 3-15 are reserved.

fnum	Console screen or border file number
dframe	Address of destination FRAME; NULLPTR indicates screen or border specified by fnum.
drect	Address of destination RECT description
sframe	Address of source FRAME; NULLPTR indicates screen or border specified by fnum.
srect	Address of source RECT description

Return Code:

ret	Error Code

Description: The COPY SVC copies the specified plane contents from one rectangle to another on the same or different FRAMEs. The drect and srect rectangles are defined in the form of RECT structures. If either the dframe or sframe FRAME specifications are 0, the file number in fnum indicates the proper FRAME. The RECT and FRAME data structures are described in Section 3.

The source and destination rectangles do not need to be the same size and can overlap on the same screen. When the rectangles are different sizes, COPY trims off the larger. The upper lefthand corner of both rectangles is used as the point of reference.

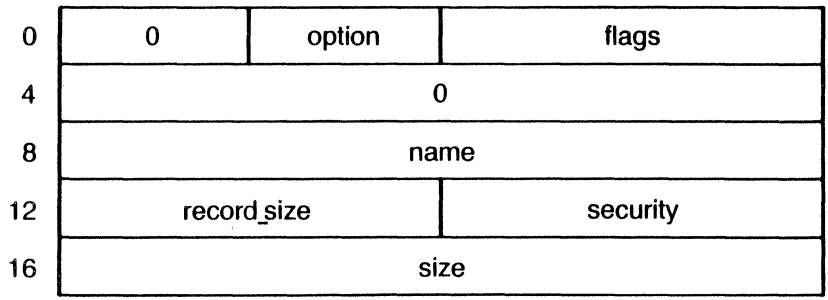
7.9 CREATE

7.9.1 Create a File, Directory, or Pipe

C Interface:

```
UWORD       flags,record_size,security;
BYTE        option,*name;
LONG        size;

fnum = s_create(option,flags,name,record_size,security,size);

ret = __osif(F_CREATE,&parmblk);
```

parmblk:



Parameters:

option
- 0 = Disk file or pipe
- 1 = Disk directory
- 2 = Virtual console (described separately)
- 3-255 Reserved

flags

bit 0: 1 = Delete file/set attributes access
0 = No delete/set access

bit 1: Reserved (must be 0)

bit 2: 1 = Write
0 = No Write

bit 3: 1 = Read
0 = No Read

bit 4: 1 = Shared
0 = Exclusive

bit 5: 1 = Allow Shared Reads if Shared
0 = Allow Shared R/W if Shared

bit 6: 1 = Shared File Pointer
0 = Unique File Pointer

bit 7: 1 = Zero Fill contiguous region*
0 = Do Not Zero Fill

bit 8: 1 = Temporary - Delete on Last Close
0 = Permanent

bit 9: 1 = Allocate space in a contiguous block*
0 = Contiguous block allocation not required

bit 10: 1 = Delete File if it already exists
0 = Return Error if file exists

bit 11 is reserved (must be zero)

bit 12: 1 = Use specified security
0 = Use default security (see ENVIRON table)

bit 13: 1 = Force Case to Media Default
0 = Do not Force Case on name

bit 14: 1 = Literal Name
0 = Prefix Substitution Allowed

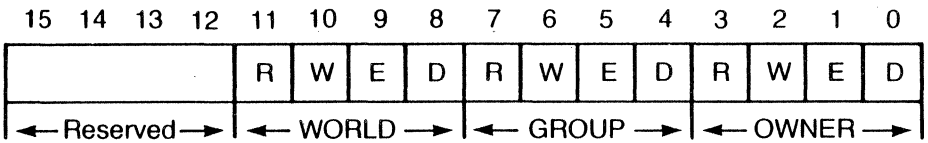
bit 15 is reserved

* Only valid if file's size value is non-zero.

name — Address of NULL-terminated name string: if file is not in default:, the string must include path specification or a previously defined logical name (maximum 128 bytes, NULL terminated)

record_size — File record size: The READ, WRITE and LOCK SVCs use this value to make sure the requested action falls on record boundaries. Use a record_size of 0 OR 1 if you want no boundary checks performed (a record_size of 0 is equivalent to a record_size of 1). FlexOS considers disk files and pipes with a record size of 1 as 8-bit files. Files with a record size of 2 are considered 16-bit files.

security — File Security Word (FSW) describing access rights for file owner, group and world. The FSW is formatted as follows:



User access is restriced according to the privilege level set for owner, group, and world users. See Section 1.4.2 for a description of the R(ead), W(rite), E(xecute), and D(elete) privileges. This value is only valid when flag bit 12 is on. Otherwise, the default security specified in the ENVIRON table is used.

size	Number of bytes to reserve for the file: For disk files, the space is contiguous only if bit 9 is set. Use a size value of 0 when you create directories and files whose size is dynamic. For pipes, the size value specifies the size of the pipe buffer. Size is not applicable to virtual consoles.

Return Code:

fnum	The file number: The file is automatically opened. The calling process must close the file if no access is needed. If the security field conflicts with the flag bits 0-6, then the file is created but not opened, and an error code is returned.
ret	Error Code

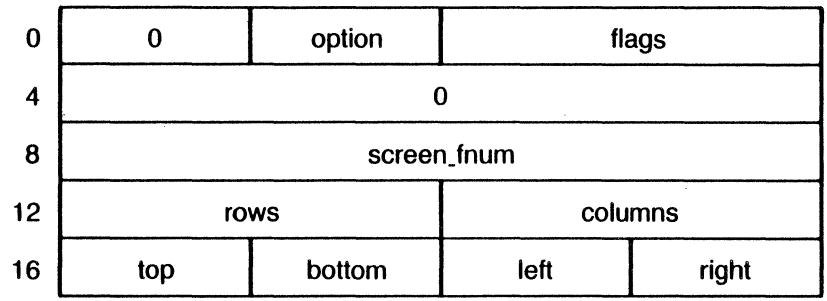
Description: CREATE option 0 adds a new disk file to a directory or a new pipe to the pipe system. CREATE option 1 makes a new directory. The record_size and size fields are only applicable to option 0; for directories, set these values to zero.

7.9.2 Create a Virtual Console

C Interface:

```
UWORD       flags;
LONG        screen_fnum;
WORD        rows,columns;
BYTE        option,top,bottom,left,right;

fnum = s_vccreate(flags,screen_fnum,rows,columns,top,bottom,left,
                  right);

ret = __osif(F_CREATE,&parmblk);
```

parmblk:



Parameters:

option
- 0 = Disk file or pipe (invalid here)
- 1 = Disk directory (invalid here)
- 2 = Virtual console (only valid choice)
- 3-255 Reserved

flags
- bit 0: 1 = Bit mapped screen
- 0 = Character mapped screen

bit 1: 1 = Bit mapped borders
0 = Character mapped borders

bit 2: 1 = Sized as specified
0 = Same size as parent

bit 3: 1 = Remove parent screen memory and restore on delete of last child's virtual console.
0 = Keep screen memory and allow writing to the parent screen

bits 4 - 7 are reserved

bit 8: 1 = Temporary - delete on last close
0 = Permanent

bits 9 - 15 are reserved

screen_fnum	File number of the parent console file on which new virtual console is based.
rows	Number of character rows in new virtual console. If flag bit 2 is zero, the number of rows is the same as the parent.
columns	Number of character columns in the new virtual console. If flag bit 2 is zero, the number of columns is the same as the parent.
top	Height of top border in characters
bottom	Height of bottom border in characters
left	Width of left border in characters
right	Width of right border in characters

Return Code:

fnum	New virtual console's file number: Use this number to GET and SET the virtual console's VCONSOLE table. Only the process that created a virtual console can change VCONSOLE values. The number returned is not the VCNUM referenced in the VCONSOLE and CONSOLE tables.
ret	Error Code

Description: This CREATE SVC option makes a new virtual console. Before you can CREATE a child virtual console, you must have at least write access to the parent console specified in screen_fnum. The row and column values do not need to be the same as the parent console's.

CREATE opens the new virtual console, which allows you to change values in the VCONSOLE table. No other process has access rights to this table. CREATE does not open the console file. Before you can read from and write to a new virtual console, you must open the console file. The name of this file is vcxxx/console where xxx is a 3-digit number indicating the virtual console's number.

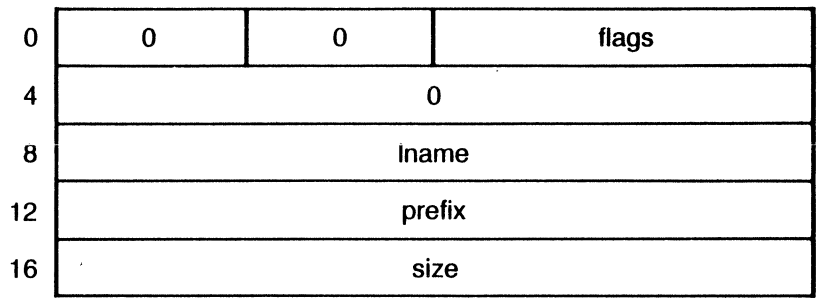
Unlike the s_create call, the s_vccreate call does not accept an option.

7.10 DEFINE

C Interface:

```
UWORD       flags;
BYTE        *lname,*prefix;
LONG        size;

ret = s_define(flags,lname,prefix,size);
ret = __osif(F_DEFINE,&parmblk);
```

parmblk:



Parameters:

flags

bit 0: 1 = Reference SYSDEF table
0 = Reference PROCDEF table

bit 1: 1 = Return prefix string
0 = Set prefix string

bits 2-13 are reserved

bit 14: 1 = Literal returned prefix
0 = Translated prefix

If bit 14 = 1, the exact prefix string is returned. Otherwise, FlexOS translates the logical name until it cannot find another entry. This is done for a maximum of 99 times, after which an error is returned.

lname	Address of logical name: lname is a NULL terminated string no longer than ten characters.
prefix	Address of prefix string buffer: If flag bit 1 = 0, prefix contents replace lname. Use NULLPTR to delete a lname. Prefix string must be NULL-terminated and cannot exceed 128 bytes. If flag bit 1 = 1, DEFINE stores the current prefix at this address.
size	Size of prefix buffer; cannot exceed 128 bytes

Return Code:

ret	Error Code

Description:

The DEFINE SVC either gives a logical name to a prefix string or returns the prefix for the specified name. Use DEFINE to redirect I/O from hardcoded filenames to other filenames or to make program-related assignments for stdin, stdout, stderr, and other logical names. The logical name cannot contain wildcard characters or path delimiters.

Logical name prefixes are kept in two tables: The SYSDEF table holds the system-wide logical name definitions and the PROCDEF table holds the process-bound logical name definitions. Child processes inherit their parent's PROCDEF table. However, DEFINE changes affect the calling process's PROCDEF table only. See Section 8 for the description of the SYSDEF and PROCDEF tables.

Only privileged processes (user and group numbers equal 0) can call DEFINE to modify SYSDEF table assignments. No special privilege is required to return a prefix from the SYSDEF table.

SVCs using names to indicate files have options to ignore prefix translation. When prefix translation is requested, the process's PROCDEF table is checked before the SYSDEF table.

When the file name specified does not include a device, FlexOS applies the special device name "default:" to the file name before attempting prefix substitution. Setting the current default directory of a process is therefore done by defining "default:". Since indirection is allowed, the user can set "default:" to another defined name such as "system:" or "home:". This implies the default directory is not necessarily legal. Programs that set the default directory should check for legality through accessing "default:" in some manner.

FlexOS does not check for loops in the DEFINE SVC. It does prevent infinite loops by only allowing 99 iterations when converting a name. FlexOS also insures that the logical name and the prefix name are not the same.

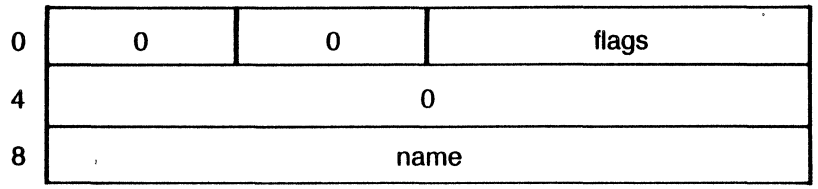
FlexOS does not check for actual device and directory names. Therefore, you can use DEFINE to store any string substitution information needed at either the system or process level. For example, you can store command macros for later translation either by the COMMAND SVC or a user interface program.

The "system:" and "boot:" logical names are initially defined at boot time by the BOOTINIT process.

7.11 DELETE

C Interface:

```
UWORD       flags;
BYTE        *name;

ret = s_delete(flags,name);

ret = __osif(F_DELETE,&parmblk);

parmblk:
```



Parameters:

flags — bits 0 - 12 are reserved

bit 13: 1 = Force case to media default
0 = Do not affect name case

bit 14: 1 = Literal name
0 = Prefix substitution allowed

bit 15 is reserved

name — Address of name of file to be deleted

Return Code:

ret	Error Code

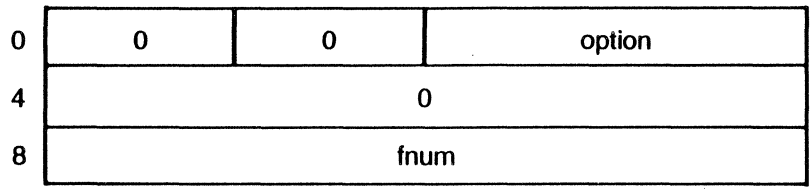
Description: The DELETE SVC removes an existing disk file, pipe, virtual console, or directory file. Before you can delete a virtual console, it must have no open files or child consoles. Before you can delete a directory file, it must be empty.

An attempt to delete an open file returns success, however, the file is not immediately deleted. Instead, FlexOS marks the file as temporary. The temporary classification results in a file that remains available until the last close, when it is deleted.

7.12 DEVLOCK

C Interface:

```
BYTE        option;
LONG        fnum;

ret = s_devlock(option,fnum);

ret = __osif(F_DEVLOCK,&parmblk);

parmblk:
```



Parameters:

option	0 - Lock for process 1 - Lock for process family 2 - Unlock
fnum	File number of the opened device

Return Code:

ret	Error Code

Description: The DEVLOCK SVC locks or unlocks a device; restricting or releasing access rights to the device. Use the option field to indicate whether you want access restricted to the calling process alone or to the calling process and other processes with the same family ID (FID).

FlexOS does not lock the device and returns an error if another process has an open file on the device. (FlexOS allows the calling process to have open files, however.) It also returns an error if the device was protected against DEVLOCK when it was installed.

The device can only be unlocked by the process that initiated the lock. The lock is automatically removed when the process terminates and when the device file is fully closed. If the lock is applied to allow only related processes access to the device, FlexOS removes the restriction when the initiating process terminates; related processes no longer have exclusive access.

7.13 DISABLE

C Interface:

```
s_disable();

ret = __osif(F_DISABLE,0);
```

Parameters:

NONE

Return Code:

NONE

Description: The DISABLE SVC suspends the calling process's program jumps to software interrupt routines (SWIs). DISABLE does not, however, suspend software interrupts generated through the EXCEPTION SVC. FlexOS triggers SWIs for events completed while software interrupts are DISABLEd when the ENABLE SVC is called.

7.14 ENABLE

C Interface:

```
s_enable();

ret = __osif(F_ENABLE,0);
```

Parameters:

NONE

Return Code:

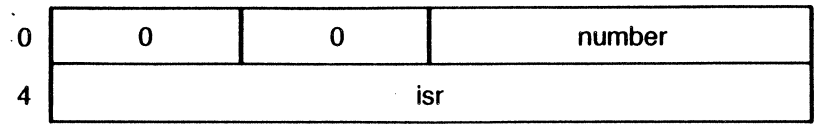
NONE

Description: The ENABLE SVC restores program jumps to software interrupt routines (SWIs). (The DISABLE SVC suspends their execution.) After ENABLE is called, FlexOS triggers the SWIs for events completed while software interrupts were DISABLEd.

7.15 EXCEPTION

C Interface:

```
WORD        number;
LONG        isr;

ret = s_exception(number,isr);

ret = __osif(F_EXCEPTION,&parmblk);

parmblk:
```



Parameters:

number	exception number
isr	Address of Interrupt Service Routine. A NULL pointer removes the software interrupt for the exception number specified.

Return Code:

ret	Error Code

Description: The EXCEPTION SVC allows a user program to trap various conditions that would otherwise result in a program abort or error.

The number parameter is a 16-bit integer specifying the exception condition to trap. Exception condition numbers are assigned as shown in Table 7-1; see the first section of the chip supplement to this book for the relationship between your microprocessor's and FlexOS's condition numbers.

Table 7-1. Exception Condition Numbers

Number	Condition
0	Non-existent memory
1	Memory boundary error
2	Illegal instruction
3	Divide by zero
4	Bound exception
5	Overflow error
6	Privilege violation
7	Trace
8	Breakpoint
9	Floating point exception
10	Stack fault
11	Emulated instruction group 0
12	Emulated instruction group 1
13	Emulated instruction group 2
14	Emulated instruction group 3
15	Emulated instruction group 4
16	Emulated instruction group 5
17	Emulated instruction group 6
18	Emulated instruction group 7
19-255	Reserved
256+n	Software interrupt n
512-64K	Reserved

Emulated instruction group 0 is reserved for software emulation of floating point hardware. Refer to programmer guide supplements for other emulation group assignments.

The Interrupt Service Routine is a machine-specific routine that must save and restore machine state if it is to return to the program causing the exception. This includes an "Interrupt Return" tailored to the CPU architecture to exit the routine. Be careful to monitor your stack utilization; your isr may have to do a stack switch for a program that is tight on stack space. This happens especially when the exception occurs in procedure code loaded through the COMMAND SVC.

7.16 EXIT

C Interface:

```
LONG        status; /*System error code or utility return code*/

s_exit(status);

ret = __osif(F_EXIT,status);
```

Parameters:

status — A 32-bit value setting exit flags in the high order word and passing completion status in the low order word.

Set exit flag bit 0 (status bit 16) to 1 to keep memory resident. Otherwise, FlexOS releases the terminating processes memory. Exit flag bits 1 - 15 are reserved and must be 0. The keep memory resident flag is only valid when the process is created with COMMAND flag bits 5 and 6 set. See Section 7.6

The completion status word is returned to terminating process's parent process.

Return Code:

NONE to calling process

Description: The EXIT SVC terminates the calling process, returns control to FlexOS, and passes back the completion status value to the parent process. Any outstanding events are cancelled and open files closed. Depending on status bit 32 (exit flag bit 16), the terminating process's memory allocation is either released or kept resident.

After a process calls EXIT, its parent's COMMAND call completes. The completion status code is placed into the low order WORD of the return code with the high order word set to 0. (If exit flag 6 was set, FlexOS resets it.) See Appendix B for utility return codes.

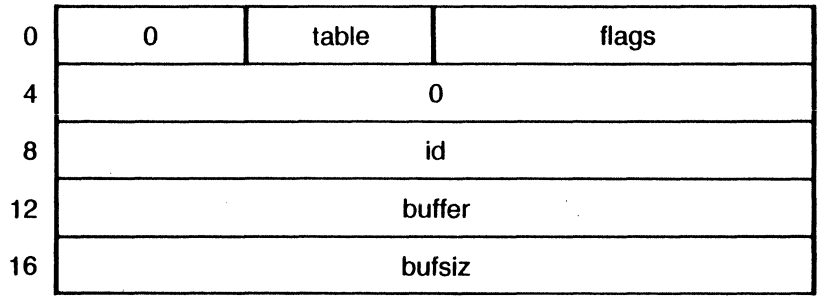
FlexOS sets the high bit of the completion status word to form a negative number when the attempt to create the process resulted in an error or the process was abnormally terminated. You can use the remainder of the bits to return a value to the parent process. By convention, a 0 value is used to indicate success while positive values indicate some form of partial completion.

7.17 GET

C Interface:

```
UWORD       flags;
BYTE        table,*buffer;
LONG        id,bufsiz;

ret = s_get(table,id,buffer,bufsiz);

ret = __osif(F_GET,&parmblk);
```

parmblk:



Parameters:

table	Table Number
flags	bits 0-7 can be used for SPECIAL devices bits 8-15 are reserved
id	File number or process ID
buffer	Address of buffer to place partial or complete table contents
bufsiz	Size of buffer in bytes

Return Code:

ret	Error Code

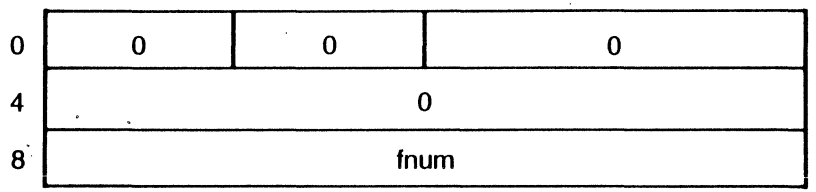
Description: The GET SVC stores partial or full table contents in the buffer specified. You specify the table by its number and, when there is more than one table with the same number, by a unique identifier. If the bufsiz is less than the size of the table structure, only the table contents up to that byte are stored.

Depending on the table type, the table ID is either a process ID or a file number. The table descriptions in Section 8 indicate what the ID is for each table. Not all tables require an ID. Use a NULL ID value for these GET calls.

7.18 GIVE

C Interface:

```
LONG        fnum;

ret = s_give(fnum);

ret = __osif(F_GIVE,&parmblk);

parmblk:
```



Parameters:

fnum	File number of virtual console whose virtual keyboard you want mapped to the physical keyboard.

Return Code:

ret	Error Code

Description: The GIVE SVC transfers access to the physical keyboard and mouse from the current virtual console to the virtual console specified by fnum. The virtual console getting ownership must be the virtual console for the process making the call or a child of that virtual console. Keyboard and mouse ownership is returned through the KCTRL SVC.

Keyboard and mouse ownership is always passed from one virtual console to another as a unit; they cannot be separated.

All characters in the previous owner's keyboard buffer are transferred into the new owner's keyboard buffer. Characters read subsequently from the physical keyboard are appended to the end of the characters in the buffer.

7.19 GSX - Perform Graphic SVC

C Interface:

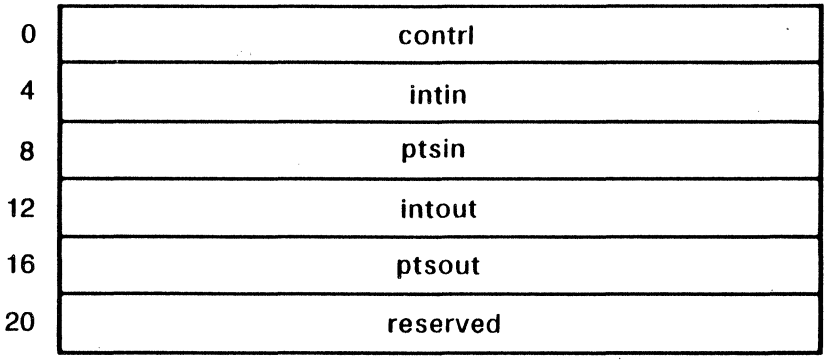
```
ret = s_gsx();

ret = __osif(F_GSX,&parmblk);

parmblk:
```

0	0	option	0
4	0		
8	fnum		
12	PB		

Parameters:

PB	Address of GSX Parameter Block

GSX Parameter Block format:



option	0	normal GSX call
	1	VDI aborting due to a CTRL-C.

Return Code:

ret	Error Code

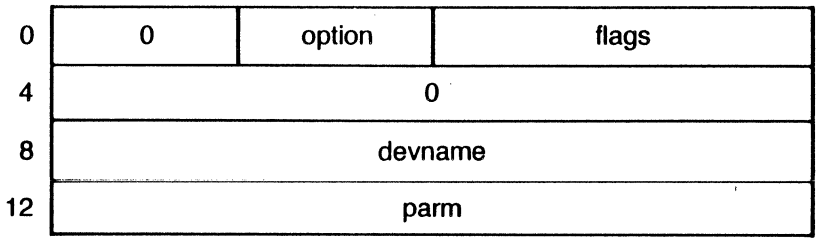
Description: The GSX SVC allows the calling process to perform a Graphics operation on the indicated file.

7.20 INSTALL

C Interface:

```
BYTE        option,*devname,*parm;
UWORD       flags;

ret = s_install(option,flags,devname,parm);
ret = __osif(F_INSTALL,&parmblk);
```

parmblk:



Parameters:

option 0 - Remove previously installed driver unit.

devname = device driver name
parm = NULL

1 - Load device driver from disk

devname = device driver name for unit 0
parm = Loadable driver disk file name

2 - Add unit to existing device driver

devname = device driver name for new unit
parm = device driver name for unit 0

3 - Link a subdriver to a device driver

devname = Front end device driver name
parm = Subdriver device driver name

flags These flags are used for options 1 and 2 only.

bit 0: 1 = Raw SET allowed
0 = Raw SET not allowed

bit 1: Reserved (must be 0)

bit 2: 1 = Raw WRITE allowed
0 = Raw WRITE not allowed

bit 3: 1 = Raw READ allowed
0 = Raw READ not allowed

bit 4: 1 = Shared access allowed
0 = Exclusive access only

bit 5: 1 = Removable device
0 = Permanent device

bit 6: 1 = DEVLOCKs allowed
0 = DEVLOCKs not allowed

bit 7: 1 = Shared access only
0 = Exclusive access allowed

bit 8: 1 = Allow device partitions
0 = Do not allow partitions

bit 9: 1 = Verify after disk writes
0 = Do not verify after disk writes

bits 10 - 12 are reserved and must be 0

bit 13: 1 = Force case to media default
0 = Do not force case

bit 14: 1 = Literal load name
0 = Prefix substitution on load name

bit 15 is reserved and must be 0

devname Address of the device name

parm Depending on the option a null pointer or the address of the loadable disk driver file name, unit 0 device name, or subdevice name.

Return Code:

ret Error Code

Description: The INSTALL SVC loads a device driver, removes a device driver, adds a unit to an existing device driver, or associates a subdevice to an existing device driver. The calling process must have group and user IDs of 0 to call INSTALL. INSTALL's devname and parm values are different for each option.

The device privileges set by the driver override those set by your INSTALL flags. This prevents you, for example, from opening a printer device with read access.

If a physical device driver has more than one unit, you must specify a unique device name to distinguish each unit. Put the unit's name in the devname field and specify the physical device driver in the parm field. devname and parm values must be null terminated and are limited to 128 bytes.

When you call option 3, both drivers must be already installed.

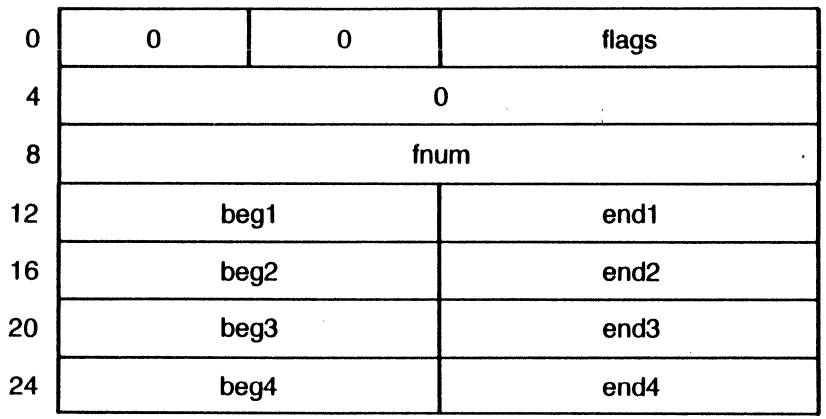
Flag bit 8 is used only by the disk resource manager and indicates whether or not the fixed disk device can have partitions. A disk with partitions cannot be formatted when installed with this bit on. Flag bit 9 is also used only by the disk resource manager. Set it when you want to verify every write to disk. Checksum verification is done.

7.21 KCTRL

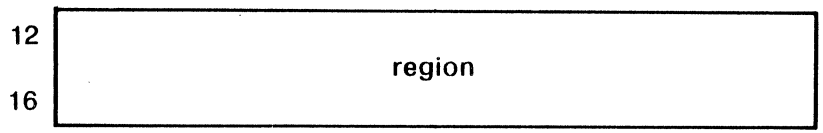
C Interface:

```
LONG        fnum;
UWORD       nranges;
UWORD       flags,beg1,beg2,beg3,beg4;
UWORD       end1,end2,end3,end4;
RECT        region;

ret = s_kctrl(fnum,nranges,beg1,end1,beg2,end2,...end4);
ret = s_mctrl(fnum,region);
ret = __osif(F_KCTRL,&parmblk);
```

parmblk:



(if mouse control)



Parameters:

flags	Reserved, must be 0 bit 0: 1 = Mouse control 0 = Character control If 0, keyboard ownership is controlled through characters typed on the keyboard and the begin range and end range parameters are required. If 1, keyboard ownership is controlled through mouse movement and a region is required.
nranges	The number of beginning and ending ranges to follow--maximum 4.
fnum	Console file number of console to get keyboard; must be console file of the parent virtual console.
begn	First character in range of characters; pressing any character in range causes keyboard to return to specified console.
endn	Last character in the range.
region	RECT structure defining a character rectangle on the parent's virtual console.

Return Code:

ret	Error Code

Description: The KCTRL SVC transfers keyboard ownership to the console file specified by fnum when a character is entered that falls within any of the four ranges specified. The initial transfer of ownership is conferred with the GIVE SVC.

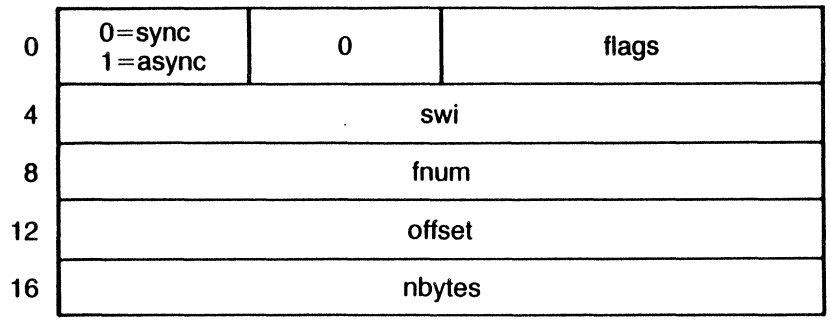
You can specify up to four character ranges. The ranges are inclusive of the first and last characters. A single character is specified by using it as the beginning and ending character. When a character falling in the range is typed, that character and all subsequent characters are diverted to the parent console file's keyboard buffer. The process controlling the virtual consoles can either give control of the keyboard to another virtual console or take some special action on behalf of the user.

You can also use mouse position to change keyboard and mouse ownership. In this case you specify a RECT (see Section 3 for the RECT description) on the parent console in which the mouse form must be resident. This region must be within the virtual console. When the mouse leaves the region, keyboard and mouse ownership go back to the parent.

7.22 LOCK

C Interface:

```
UWORD       flags;
LONG        fnum,offset,nbytes;

ret = s_lock(flags,fnum,offset,nbytes);
emask = e_lock(swi,flags,fnum,offset,nbytes);
ret = __osif(F_LOCK,&parmblk);

parmblk:
```



Parameters:

flags — bits 0 and 1 select the LOCK mode

- 0 = Unlock
- 1 = Exclusive lock
- 2 = Exclusive write lock
- 3 = Shared write lock

bits 2–3 are reserved (must be 0)

bit 4: 1 = Return error on lock conflict
0 = Wait on lock conflict

bits 5-7 are reserved (must be 0)

bits 8 and 9 determine how the offset field is interpreted

0 = Relative to beginning of file
1 = Relative to file pointer
2 = Relative to end of file

bits 10-15 are reserved (must be 0)

swi	Address of software interrupt routine
fnum	File number whose contents you want to lock and unlock
offset	Offset of region to lock in file
nbytes	Length of region to lock

Return Code:

ret	Error Code

Description: The LOCK SVC either locks or unlocks a region of a disk file, restricting or releasing access rights in the process. The disk file is specified by fnum and the area to be locked is determined by flag bits 8 and 9, offset, and nbytes. The Disk Resource Manager verifies that offset and nbytes define a region that falls on record boundaries for files created with a record size. If you specify a region that does not fall on a record boundary, no records are locked or unlocked and an error message is returned.

The lock modes selected by flag bits 0 and 1 are defined as follows:

- **1--Exclusive lock:** Prevents other processes from locking, reading from, writing to, or deleting the region.
- **2--Exclusive write lock:** Lets other processes read from the region but prevents them from locking, writing to, or deleting the region.
- **3--Shared write lock:** Allows other processes to read from and establish a shared write lock on but prevents them from writing to the region

Flag bit 4 determines what happens when the region requested is already locked in an exclusive mode. When you set bit 4 to 1, an error code is returned; when you set bit 4 to 0, LOCK waits for the region to be unlocked, locks the region, and returns.

The offset of the lock region in the file is, depending on the value in flag bits 8 and 9, relative to the beginning of the file, the current file pointer, or the end of the file. The file pointer location is modified by the READ, WRITE and SEEK SVCs. The nbytes value determines how many bytes are locked.

To unlock a region set flag bits 0 and 1 to 0. The offset indicates the first byte of the region to unlock and nbytes the number of bytes to unlock. Because the region unlocked is independent of the region initially locked, you can lock a large region of a file and then release portions as the lock becomes unnecessary so that other processes can have access. Once a region is unlocked, it can be locked by another process.

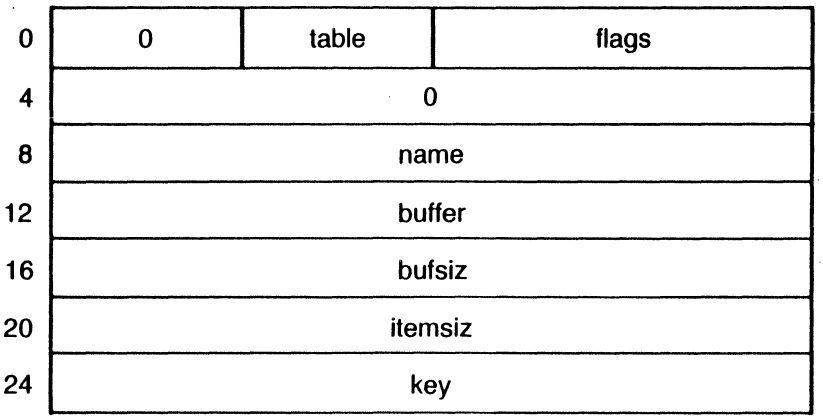
An unlock specification with flags and offset values equal to 0 and nbytes equal to 0xFFFFFFFF removes all locks on the file made by the calling process. The number of unlock calls does not have to match the number of lock calls.

7.23 LOOKUP

C Interface:

```
UWORD       flags;
BYTE        table,*name,*buffer;
LONG        key,bufsiz,itemsiz,nfound;

nfound = s_lookup(table,flags,name,buffer,bufsiz,itemsiz,key);

ret = __osif(F_LOOKUP,&parmblk);
```

parmblk:



Parameters:

table	Table Number (Table 10-1 lists the table numbers)
flags	bits 0 - 7 are dependent on table type
	bits 8 -12 are reserved (must be 0)

bit 13: 1 = Force name case to media default
0 = Do not change name case

bit 14: 1 = Literal name
0 = Prefix translation allowed

bit 15 is reserved (must be 0)

name — Address of the table name to search for; names are case sensitive.

buffer — Address of buffer to store information collected.

bufsiz — Size of buffer in bytes.

itemsiz — The number of bytes to store from each table. If itemsiz is less than the table size, only that many bytes from each table found are written in the buffer. If itemsiz is greater than the table size, the excess area is not modified.

key — Key from which to continue searching. The key value depends on the table type. Each table allowing LOOKUP specifies a key for continued search. The LOOKUP SVC continues the search from the first item after the key. A key value of 0 always starts the LOOKUP search from the beginning of the table.

Return Code:

nfound — Number of tables found. LOOKUP stops searching when the end of the buffer is reached or there are no more tables. If the last table does not fit into the remaining buffer space, it is discarded.

ret — Error Code

Description: The LOOKUP SVC searches the system tables for those matching the table and name specified. The key field is used to specify the starting point for the search. A key value of zero specifies the beginning. A table's key value is defined by the resource manager responsible for that table. When a match is found the table, or an excerpt corresponding to the itemsiz in length, is copied into the buffer. The search continues until the buffer is filled or there are no more tables.

The name specification is limited to 128 bytes and must be null terminated. You can use wildcards in the name specification. However, you are restricted to the lowest level of a path name--that is, files within a directory and devices on a node. The name "*" is translated to mean "default:*".

Table names are case sensitive and you must enter your specification with the same case letters to get a match. This is also true when you use wildcards. For example, the entry "s*" returns only those tables beginning with a lowercase s.

A return of 0 indicates success, but means that LOOKUP found no tables.

Table numbers, names, keys and the use of flag bits 0 through 7 are described in Section 8.

7.24 MALLOC

C Interface:

```
LONG        *mpbptr,mpbsiz;
BYTE        option;

ret = s_malloc(option,mpbptr);

ret = __osif(F_MALLOC,&parmblk);
```

parmblk:

0	0	option	0
4	0		
8	0		
12	mpbptr		
16	mpbsiz		

Parameters:

option	0 = Expand existing heap 1 = Allocate a new heap
mpbptr	Address of Memory Parameter Block
mpbsiz	Size of Memory Parameter Block in bytes

The Memory Parameter Block must have the following format:

0	start
4	min
8	max

start: For option equals 0, set the base address of the heap segment to be expanded in this field. MALLOC writes the base address of the added memory portion before it returns. For option equals 1, set this field to zero. MALLOC fills in the base address of the new heap here.

min: Specify the minimum number of bytes required. MALLOC fills in the actual number allocated before returning.

max: Specify the maximum number of bytes required. MALLOC does not change your entry.

Return Code:

ret Error Code

Description: MALLOC either adds contiguous memory to the end of an existing heap or allocates a new heap. Use the option field to select one or the other and the Memory Parameter Block to specify the minimum and maximum memory requirements. Set the Memory Parameter Block's start parameter to the base address of the existing heap for option 0 or to zero for option 1.

Note: Process are not automatically given an initial heap allocation. Consequently, option 1 must be called the first time heap space is needed.

When you select option 0, MALLOC extends the designated heap contiguously and modifies your Memory Parameter Block's start and min parameters to indicate the new allocation's starting address and actual allocation, respectively. The original heap's base address (which is present PROCESS table) and contents remain unchanged.

When you select option 1, the new heap may or may not be contiguous with any previously allocated heap. MALLOC modifies your Memory Parameter Block's start and min values to indicate the new heap's base address and actual allocation. These new values also appear as the PROCESS table's HEAP and HSIZE parameters. The new heap may be allocated such that an existing heap is no longer expandable.

MALLOC use is affected by the type of processor. See the supplement corresponding to your processor for more information.

7.25 MFREE

C Interface:

```
BYTE        *start;

s_mfree(start);

ret = __osif(F_MFREE,start);
```

Parameters:

start	First address in heap to free

Return Code:

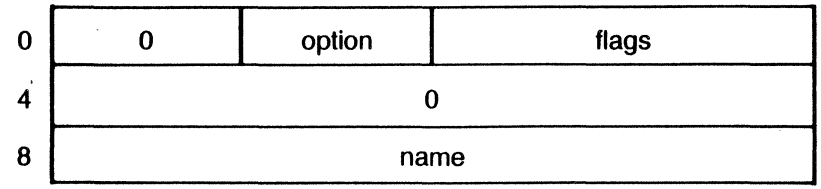
ret	Error Code

Description: The MFREE SVC releases the memory in a heap from the address specified to the end of that heap.

7.26 OPEN

C Interface:

```
UWORD       flags;
BYTE        *name;
LONG        fnum;

fnum = s_open(flags,name);

ret = __osif(F_OPEN,&parmblk);

parmblk:
```



Parameters:

option — May be used by SPECIAL devices

flags

bit 0: 1 = Delete file/set attributes access
0 = No delete/set access

bit 1: 1 = Execute access
0 = No execute access

bit 2: 1 = Write access
0 = No write access

bit 3: 1 = Read access
0 = No read access

bit 4: 1 = Shared
0 = Exclusive

bit 5: 1 = Allow shared reads if shared
0 = Allow shared R/W if shared

bit 6: 1 = Shared file pointer
0 = Unique file pointer

bit 7: 1 = Reduced access accepted
0 = Return error on reduced access

bits 8 - 12 are reserved (must be 0)

bit 13: 1 = Force case to media default
0 = Do not affect name case

bit 14: 1 = Literal name
0 = Prefix substitution allowed

bit 15 is reserved (must be 0)

name Address of file, pipe, or device name

Return Code:

fnum file number
ret Error Code

Description: The OPEN SVC opens an existing file and returns a 32-bit file number used for subsequent I/O. "File" in this context refers to disk files, pipes, and device files used to communicate with printers, mouses, consoles, and special devices. FlexOS sets the file pointer to 0 when you open the file.

Use flag bits 0 through 3 to request the file access privileges--read, write, execute, and delete/set. Use flags 4, 5, and 6 to set the access mode--shared versus exclusive, shared read only versus shared read/write when shared, and shared versus unique file pointer. The use of these flags to monitor file access differs slightly from one type of file to another. See the sections in this manual on disk file, console, pipe, and special device management for the description of flag use with these types of files.

Set flag bit 6 when you want two or more processes to share the same file pointer; this feature is only available to processes with the same family identification number (FID). Each process sharing the pointer must have this flag set. When this bit is set, the value of flag bit 1 is assumed to be 1; the actual value is ignored.

Set bit 7 to accept reduced access privileges. The file's governing privileges for owner, group, and world categories are set when it is created. Reduced access is an issue when a disk label's security flag bit is set and you request a privilege level not available to a process with your ID and group number. Set this flag to 1 if you can accept reduced access; FlexOS ANDs the file's R, W, E, and D privileges corresponding to your category with those you requested to determine the privileges you actually get. Set this flag to 0 if you cannot accept reduced access; FlexOS returns an error code when the privileges do not match.

Files can be opened any number of times. Each open returns a different file number and each must be closed. Use this technique to obtain greater access to a file without losing your previous access. The standard protection rules do not apply on multiple opens of the same file by the same process. For example, if you open a file in SHARED, READ-ONLY mode, you can later open it in EXCLUSIVE, READ-WRITE mode. The protection rules still apply, however, with respect to other processes attempting to open the file.

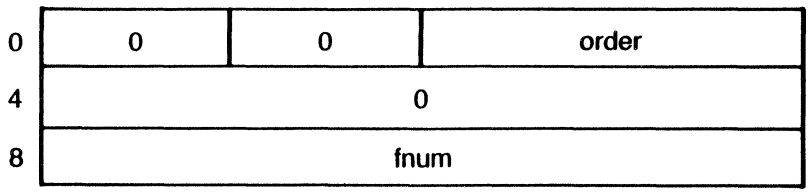
Pipe file's read and write ends are separate and independent of each other. Similarly, a console file can be opened for read and write access separately. If one process opens a console or pipe file with EXCLUSIVE, READ access, another can open it with EXCLUSIVE, WRITE access. One end of a pipe file can be opened in SHARED mode while the other is opened in EXCLUSIVE mode. For pipes, how you open the file affects the pipe's operation.

7.27 ORDER

C Interface:

```
WORD        order;
LONG        fnum;

ret = s_order(order,fnum);

ret = __osif(F_ORDER,&parmblk);
```

parmblk:



Parameters:

order — New virtual console position

- 0 = Bottom
- 1 = Next to Bottom
- 2 = 2nd from Bottom
- n = Nth from Bottom
- -1 = Top

fnum — File number of virtual console to move

Return Code:

ret — Error Code

Description: The ORDER SVC changes the position of the virtual console with file number fnum in a "stack" of sibling virtual consoles. The "order" value specifies the virtual console's new position. Use -1 to specify the top. All other positions are designated by number where 0 is the bottom console, 1 the next, then 2, and so forth. The Console Resource Manager adjusts the position numbers after you make a change.

The initial order of precedence corresponds to the order of creation.

7.28 OVERLAY

C Interface:

```
BYTE        *codeadr,*dataadr;
LONG        fnum,offset;

ret = s_overlay(fnum,codeadr,dataadr,offset);

ret = __osif(F_OVERLAY,&parmblk);
```

parmblk:

0	0	0	0
4	0		
8	fnum		
12	codeadr		
16	dataadr		
20	offset		

Parameters:

fnum	File number of the opened file containing one or more overlay procedures
codeadr	Address in calling process's code area in which to load the overlay code.

dataadr	Address in calling process's data area in which to load the overlay data.
offset	Byte offset into file of the overlay header. The header must be in the same format as the default program load image used by the COMMAND SVC.

Return Code:

ret	Error Code

Description: The OVERLAY SVC loads the code and data from the designated overlay file into the calling process's memory. The overlay file is specified by fnum and the code and data addresses by the codeadr and dataadr pointers, respectively. Use the offset value to select a specific overlay within a file containing several. Each overlay in a file must have its own header. The overlay file must be open and the calling process must have EXECUTE privilege.

An E_MEMORY error is returned if the overlay does not fit into the calling process's code or data area starting at the specified address.

When the COMMAND SVC detects overlays in the program file, it automatically keeps the file open. The file number can be found in the ENVIRON table.

7.29 READ

C Interface:

```
LONG        fnum,offset,bufsiz,nbytes;
UWORD       flags,*delimiters;
BYTE        *buffer,option;

nbytes = s_read(flags,fnum,buffer,bufsiz,offset);
emask = e_read(swi,flags,fnum,buffer,bufsiz,offset);
nbytes = s_rdelim(flags,fnum,buffer,bufsiz,offset,delimiters);

ret = __osif(F_READ,&parmblk);
```

parmblk:

0	0=sync 1=async	option	flags
4	swi		
8	fnum		
12	buffer		
16	bufsiz		
20	offset		
24	delimiters		

Parameters:

option May be used by SPECIAL devices

flags

bit 0: 1 = Read from device. On disk files internal buffers are flushed and discarded before reading. On a keyboard file, the type ahead buffer is flushed.
0 = Allow reading from internal buffers

bit 1: 1 = Read until delimiter
0 = Not delimited

bit 2: 1 = Non-destructive read: Read the internal buffer contents without removing bytes pertinent to keyboard and pipe files only; disk file reads are always non-destructive.
0 = Normal read

bit 3: 1 = Preinitialized read
0 = Normal read

bit 4: 1 = Include delimiter in buffer
0 = Exclude delimiter

bit 5: 1 = Edited read (only relevant when "Read until Delimiter" flag is on.)
0 = Normal Read

bits 6-7 are reserved (must be 0)

bits 8 and 9 determine interpretation of the offset field:

0 = relative to the beginning of file
1 = relative to the file pointer
2 = relative to the end of file

bits 10-15 are reserved (must be 0)

swi	Address of software interrupt routine
fnum	File number of file to read
buffer	Address of buffer in which to place information
bufsiz	Size of buffer in bytes
offset	Byte offset relative to the position indicated by flag bits 8 and 9. Negative offsets are allowed.
delimiters	Address of an array of WORD values. This field is ignored on non-delimited reads. The first item indicates the number of delimiters in the array; 16-bit character delimiters follow. If the file being read is an 8-bit file, the high byte of each delimiter is ignored. Disk files and pipes with a record size of 1 are considered 8-bit files; files with a record size of 2 are considered 16-bit files. If the record size is greater than 2, a record size error is returned. The keyboard mode in the CONSOLE table determines if a console file is 8-bit or 16-bit oriented. On other devices, the device driver determines if it is an 8-bit or 16-bit device.

Return Code:

nbytes	Number of bytes read
	Error Code

Description: The READ SVC extracts data from the specified file. Data can be read either sequentially or randomly. The offset field is always added to either the beginning of a file, the current file pointer, or the end of file (see flag bits 8 and 9). You can specify a negative offset; this is useful, for example, to reread the last record of a file. Set flag bits 8 and 9 to one and the file pointer to one to perform sequential I/O.

The file pointer is updated on every read to the byte position after the transferred data in the file. It is initialized to 0 at OPEN.

The READ SVC verifies that the offset and bufsiz fields are on record boundaries if the file was created with a record size. If the values do not fall on record boundaries, no characters are read and an error code is returned.

The READ SVC can be called asynchronously on character oriented devices such as keyboards and special devices if the delimited read flag is not set. In this case, the number of characters read is at least one before the event is completed. The disk system does not support asynchronous READs. The pipe system supports asynchronous undelimited READs and reads as many characters as requested.

When using the delimited read flag, READ cannot be called asynchronously. The buffer size is limited to 256 bytes. Editing is performed by keyboards on delimited reads only. Common delimiters include the <carriage return>, <line feed> and <help> keys. The standard editing characters are as follows:

LEFT ARROW	Move cursor one character to left.
RIGHT ARROW	Move cursor one character to right.
DELETE	Delete next character
BACKSPACE	Delete previous character
CTRL-B	Move cursor to beginning of line if not at beginning, otherwise move to end of line.
CTRL-X	Erase from beginning of line to cursor

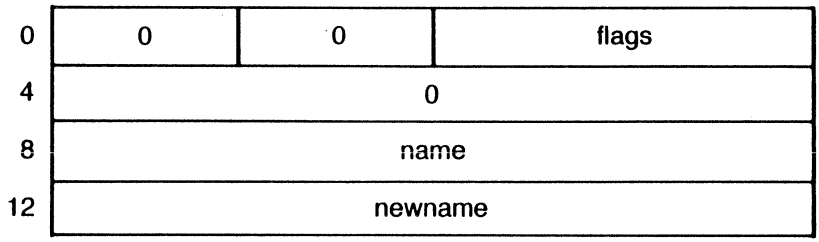
If a standard editing key is used as a delimiter, it has no effect on the returned buffer. These keys can be changed by an application program through the use of the XLAT SVC. The OEM that configures the system can also set the original editing character set.

7.30 RENAME

C Interface:

```
UWORD       flags;
BYTE        *name,*newname;

ret = s_rename(flags,name,newname);

ret = __osif(F_RENAME,&parmblk);
```

parmblk:



Parameters:

flags — bits 0 - 12 are reserved

bit 13: 1 = Force case to media default
0 = Do not affect name case

bit 14: 1 = Literal name and new name
0 = Prefix translation allowed

bit 15 is reserved

name — Address of string containing name of existing file.

newname	Address of string containing new name of file.

Return Code:

ret	Error Code

Description: The RENAME SVC renames an existing disk file or directory. If the file is currently open by another process, FlexOS does not rename the file and returns an error. For files, if the new name specifies another directory, the file is moved to that location. This feature is limited to directories on the same drive. Attributes, ownership, protection and date stamps are not changed.

7.31 RETURN

C Interface:

```
LONG        emask;

ret = s_return(emask);

ret = __osif(F_RETURN,emask);
```

Parameters:

emask	Event mask of completed event

Return Code:

ret	return code of asynchronous SVC

Description: The RETURN SVC retrieves the return code of an asynchronous SVC. If the event is not complete, FlexOS waits for it to complete before returning from the RETURN call. Use WAIT or STATUS to determine if the event has completed. The return code is the code that would have been returned if the SVC had not been called synchronously. Once the RETURN SVC has been called, the event's emask bit is cleared.

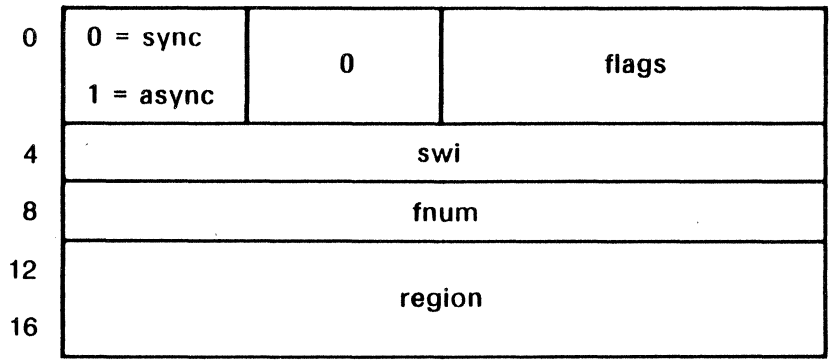
Note: You cannot use RETURN for events with a software interrupt (swi). The event's completion is provided to the swi and is not kept available to the parent process.

7.32 RWAIT

C Interface:

```
RECT        *region;

position = s_rwait(flags,fnum,region);
emask = e_rwait(swi,flags,fnum,region);

ret = __osif(F_RWAIT,&parmblk);
```

parmblk:



Parameters:

flags — bit 0: 1 = clip to current window
0 = no clip

bit 1: 1 = return on exit from rectangle
0 = return on entry to rectangle

bits 2-15 are reserved and must be 0.

fnum — File number of open mouse file

region RECT structure describing a rectangular area of the screen associated with the mouse.

0	ROW	COL
4	NROW	NCOL

Return Code:

ret Error Code

Description: The RWAIT SVC allows a process to detect the mouse entering or exiting a described region of the screen.

7.33 SEEK

C Interface:

```
LONG        fnum,offset;
UWORD       flags;

position = s_seek(flags,fnum,offset);

ret = __osif(F_SEEK,&parmblk);
```

parmblk:

0	0	0	flags
4	0		
8	fnum		
12	offset		

Parameters:

flags — bits 0-7 are reserved (must be 0)

bits 8-9 determine how to interpret the offset field

0 - Relative to beginning of file
1 - Relative to file pointer
2 - Relative to end of file

bits 10-15 are reserved

offset — Number of bytes relative to reference selected in flag bits 8 and 9

Return Code:

position — Current position of the file pointer after SEEK call.

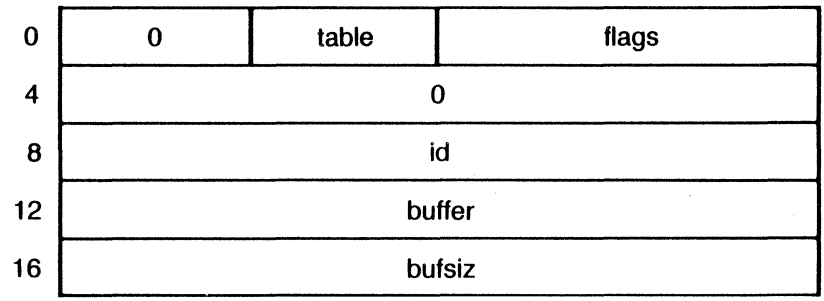
Description: The SEEK SVC either returns or changes the file pointer position of the specified file. To get the current pointer position, select the "Relative to file pointer" option in flag bits 8 and 9 and specify an offset of 0. Any other combination of values for flag bits 8 and 9 and the offset cause a change in the file pointer position. For all SEEK calls, the value returned indicates the current file pointer position.

The offset value can be positive or negative. An error is returned, however, if the new pointer position is less than 0. If the file consists of multibyte records, the offset must fall on a record boundary.

7.34 SET

C Interface:

```
BYTE        table,*buffer;
LONG        id,bufsiz;
UWORD       flags;

ret = s_set(table,id,buffer,bufsiz);

ret = __osif(F_SET,&parmblk);

parmblk:
```



Parameters:

table	Table number
flags	bits 0-7 may be used by SPECIAL drivers bits 8-15 are reserved and must be 0

id	Table identifier (required only when you have more than one table with the same number)
buffer	Address of source buffer with new table contents
bufsiz	Size of buffer in bytes

Return Code:

ret	Error Code

Description: The SET SVC changes table contents. The table is specified by the table number and, if necessary, an id. The id is table dependent; see the individual table explanations in Section 8 for the id value of a specific table. Not all tables can be modified with SET and some tables can only be modified by privileged processes.

If the bufsiz specified is less than the size of the table, the buffer contents replace the table contents starting from the beginning of the table. The remainder of the table is not changed.

7.35 SPECIAL

C Interface:

```
UWORD       flags;
LONG        fnum,dbufsiz,pbufsiz,;
BYTE        func,*databuf,*parmbuf;

ret = s_special(func,flags,fnum,databuf,dbufsiz,parmbuf,pbufsiz)
emask = e_special(swi,func,flags,fnum,databuf,dbufsiz,parmbuf,
pbufsiz);

ret = __osif(F_SPECIAL,&parmblk);
```

parmblk:

0	0=sync 1=async	func	flags
4	swi		
8	fnum		
12	databuf		
16	dbufsiz		
20	parmbuf		
24	pbufsiz		

Parameters:

func	SPECIAL function number field: Bits 6 and 7 indicate the data flow direction of the databuf and parmbuf buffers as follows: bit 7--parmbuf: 1 = write buffer, 0 = read buffer bit 6--databuf: 1 = write buffer, 0 = read buffer If no data or parameters are to be transferred, set the bits to 0. The remainder of the bits in the number are determined by the device drivers.
flags	Depends on type of file; however bits 11, 14, and 15 are always reserved and must be 0.
swi	Address of software interrupt routine
fnum	File number returned when device was opened
databuf	Address of data buffer. If dbufsize field is NULL, this field is data.
dbufsiz	Size of data buffer in bytes or NULL to indicate that data is in databuf field or that there is no data.
parmbuf	Address of parameter buffer. If pbufsiz field is NULL, this field is data.
pbufsiz	Size of parameter buffer or NULL to indicate that parameter is in the parmbuf field or that there are no parameters.

Return Code:

ret	Return code depending on type of file
ret	Error code

Description: The SPECIAL SVC provides direct access to a device. The calling process must have opened the device before a SPECIAL function can be used. The function number indicates what type of operation to perform. SPECIAL requires the driver, not the resource manager, to interpret the function number and perform the operation.

Although SPECIAL functions and return codes differ according to the driver, the SPECIAL SVC parameter block is always formatted as shown above. The format rules are as follows:

- The most significant 2 bits of the func field determine the direction of the buffer data flow as described in the func description above. The lower 6 bits of the func field and flag bits 0–10, 12, and 13 are driver dependent.
- The flags field is a bit map of flags affecting the function's mode of operation and are typically function dependent.
- The databuf and dbufsiz fields make up a buffer specifier. If dbufsiz is 0 (NULLPTR), databuf field is a 32 bit data value rather than a pointer.
- The data buffer cannot contain pointers.
- The parmbuf and pbufsiz fields are also buffer specifiers and follow the same rules as the databuf and dbufsiz fields.

There are a maximum of 64 SPECIAL function numbers. (This is because only function number bits 0 through 5 can be used.) The first 32 digits are reserved for use by Digital Research Inc. The second 32 digits are available for use by an OEM.

7.35.1 Disk Resource Manager SPECIAL Functions

The Disk Resource Manager recognizes nine SPECIAL functions for disk initialization and raw disk I/O. These SPECIAL calls allow you to access the disk directly, bypassing normal operations and restrictions. You must specify OEM-defined parameters, such as sector size or file allocation table (FAT) address, to use some functions.

The parameter blocks for the Disk Resource Manager functions adhere to the model shown above but differ in their flag use and buffer requirements. Each function description below includes the flag definitions. Buffer requirements are shown in the parameter block illustrations. If a 0 is shown in the field, there is no corresponding parameter.

The return code for all Disk Resource Manager functions is either E_SUCCESS or E_IOERRS.

Note: Before you can perform the SPECIAL disk initialization functions 1 through 3, you must open the device with exclusive access and call SPECIAL function 8. Read, write, and/or set privileges are also required as described below.

Disk Function 0: Read System Area

SPECIAL disk function 0 reads in the system area of the disk into the data buffer. GET the drive's DISK table before you call function 0 to determine if the system area exists on the disk. The size of the data buffer must be greater than or equal to the size of the system area. FlexOS requires the user to have disk read privilege to perform this function.

0	0	0	flags
4	swi		
8	fnum		
12	databuf		
16	dbufsiz		
20	0		
24	0		

Parameters:

flags Bits 0-15: Reserved

Disk Function 1: Write System Area

SPECIAL disk function 1 writes the contents of the data buffer onto the system area of the disk. GET the drive's DISK table to determine if the disk has a system area before calling function 1. The size of the data buffer must match the size of the system area. You must must open the drive in exclusive mode with write privileges and call SPECIAL disk function 8 before you can write to the system area.

0	41H	flags
swi		
fnum		
databuf		
dbufsiz		
0		
0		

Parameters:

flags Bits 0–15: Reserved

Disk Function 2: Format System Area of Disk

SPECIAL disk function 2 formats the disk's system area according to the convention of the driver. GET the drive's DISK table to determine if the system area exists before calling function 2. You must open the drive in exclusive mode with read, write, and set privileges and call SPECIAL disk function 8 before you can format the system area.

0	2	flags
swi		
fnum		
0		
0		
0		
0		

Parameter:

flags Bits 0–15: Reserved

Disk Function 3: Format Track

SPECIAL disk function 3 formats the disk media according to the specifications in the parameter buffer. You must open the drive in exclusive mode with read, write, and set privileges and call SPECIAL disk function 8 before you can use this function

0	83H	flags
swi		
fnum		
0		
0		
parmbuf		
pbufsiz		

Parameters:

flags Bit 0: Reserved

Bit 1: 1 = Mark the whole track as bad

Bit 2: 1 = Use C, H, S, and N fields

Bit 3: 1 = Use HEAD, CYLINDER, BYTESEC, and SECTOR fields

Bits 4-15: Reserved

Set bit 1 to remove tracks designated bad by the manufacturer from file system access. Set flag bit 3 rather than flag bit 2 and specify the track in the head and cylinder fields. The remainder of the fields are irrelevant in this operation.

You select the format locations and characteristics in the parmbuf (parameter buffer). The data structure provides two, mutually exclusive means for specifying the starting head, cylinder, sector number, and the number of bytes per sector for the format operation. The other fields are valid for both options. Set flag bit 2 or 3 to select one means over the other.

parmbuf Format:

HEAD	0	CYLINDER	
DENS	FILL	BYTESEC	
SECTRK		SECTOR	
C	H	S	N
:			
C	H	S	N

HEAD Starting head number

CYLINDER Starting cylinder number

DENS Format density where

0 - Single density
1 - Double density

FILL Fill character

BYTESEC	Number of bytes per sector
SECTRK	Number of sectors per track
SECTOR	Starting sector number. When you set bit 3, the format operation begins with the sector field specified here
C, H, S, & N	a variable length list of 4-byte fields where:

C is a starting cylinder number
H is a head number
S is a starting sector number
N is the number of bytes/sector

When you set bit 2, the format operation begins after the sector specified in each entry. The number of items in the list is determined by pbufsiz.

Disk Function 4: Media Check

SPECIAL disk function 4 checks to see if the media has changed or if a physical or logical error condition exists on the media. You must have opened the drive in at least GET-only mode to use this function. (GET-only mode is described in Section 2.6 above.)

0	4	flags
swi		
fnum		
0		
0		
0		
0		

Parameters:

flags Bits 0-15: Reserved

Disk Function 5: Flush Buffers

SPECIAL disk function 5 writes any updated buffers onto the disk. The user must have opened the device, however, no particular privilege is required.

0	5	flags
swi		
fnum		
0		
0		
0		
0		

Parameters:

flags Bits 0-15: Reserved

Disk Function 6: Read Physical Record

SPECIAL disk function 6 either reads data from the media into the data buffer or verifies the data is valid; flag bit 2 determines which operation is performed. The media starting point for both operations is defined in the parameter buffer by head, sector, and cylinder numbers. The dbufsiz value determines how much data is read. dbufsiz must be a multiple of the media's sector size. You must have opened the drive with at least read privilege to use function. No data is read, however, when you select the verify option.

0	86H	flags
swi		
fnum		
databuf		
dbufsiz		
head	sector	cylinder
0		

Parameters:

flags

Bits 0-1: Reserved
Bit 2: 1 = Verify media
0 = Read media
Bits 3-15: Reserved

The starting head, sector and cylinder numbers are specified in the H, S, C fields above.

Disk Function 7: Write Physical Record

SPECIAL disk function 7 writes the data buffer contents to the media. The media starting point is specified in the parameter buffer by head, sector, and cylinder number. The dbufsiz value determines how much data is written. dbufsiz must be a multiple of the media's sector size. You must open the drive in exclusive mode with write access before you can use this function.

0	C7H	flags
swi		
fnum		
databuf		
dbufsiz		
head	sector	cylinder
0		

Parameters:

flags Bits 0-15: Reserved

The starting head, sector, and cylinder numbers are specified in the H, S, C fields, above.

Disk Function 8: Initialize Format

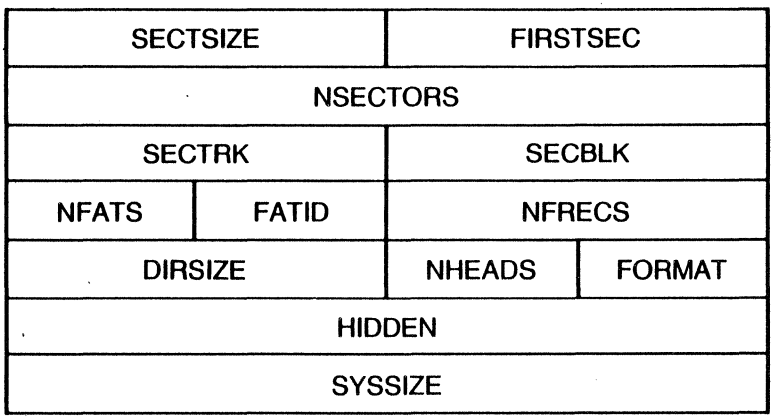
SPECIAL disk function 8 supplies the file system and the disk driver with the drive's Media Descriptor Block (MDB). This function must be called before the user calls SPECIAL disk functions 1, 2, and 3. To execute this call, the user must have opened the drive in exclusive mode with read, write and set privileges.

0	48H	flags
swi		
fnum		
databuf		
dbufsiz		
0		
0		

Parameters:

flags Bits 0 - 15: Reserved

The Media Descriptor Block is specified in the data buffer as follows:



SECTSIZE Size of sectors in bytes

FIRSTSEC First physical sector number of File Allocation Table (FAT) on track 0

NSECTORS Number of sectors in logical image of disk including FATs, directory, and boot record

SECTRK Number of sectors per track

SECBLK Number of sectors per block

NFATS Number of FATs

FATID FAT identification byte

NFRECS Number of sectors in a FAT

DIRSIZE Number of directory entries in the root directory

NHEADS Number of heads

FORMAT	Media format according to the following values 0 = RAW 1 = 1.5 byte FATs 2 = 2 byte FATs
HIDDEN	Number of sectors in partitions preceding the media's logical image
SYSSIZE	Number of bytes in the system area.

7.35.2 Miscellaneous Resource Manager SPECIAL Functions

Two SPECIAL functions are provided for accessing serial-type port devices when they are serving as subdrivers. Use these functions as you would GET and SET to determine the driver's current values and set them. The data structure used for both functions is the PORT table.

The fnum value for both calls is the file number returned when you open the subdriver's owner. See Section 6 for the description of the owner and the procedure for finding it.

Miscellaneous Device Function 0: Get Current PORT Table Values

Use this function to determine the subdriver's current PORT table values.

0	13H	0
0		
fnum		
0		
0		
parmbuf		
pbufsiz		

Parameters:

parmbuf	Address of buffer to place the PORT table.
pbufsize	Length of the buffer; if the number splits a field, that value is not copied.

Miscellaneous Device Function 1: Set Port Table Values

Use this SPECIAL function to set a subdriver's PORT table values.

0	93H	0
0		
fnum		
0		
0		
parmbuf		
pbufsiz		

Parameters:

parmbuf	Address of buffer with source PORT table values.
pbufsize	Length of the buffer; if the number splits a field, that value is not set.

7.36 STATUS

C Interface:

```
LONG        cmask;

cmask = s_status();

ret = __osif(F_STATUS,0L);
```

Parameters:

NONE

Return Code:

cmask	Bit map of completed events

Description: The STATUS SVC informs the calling process of previously initiated asynchronous events that have completed and whose return codes have not been retrieved by the RETURN SVC. If the event specified has a software interrupt (swi), the cmask value for that event is 0 rather than 1. (You do not call RETURN for events with a software interrupt.)

Note: STATUS places a heavy burden on the CPU; excessive use of STATUS impacts program performance.

7.37 SWIRET

C Interface:

```
LONG        option

s_swiret(option);

ret = __osif(F_SWIRET,option);
```

Parameters:

option 0 - return to main program at point of interruption

1 - assume process identity from main program

Return Code:

NONE

Description: The SWIRET SVC is used to return from a software interrupt routine (swi). It provides two options:

- return to the main program at the point of interruption
- retain control of subsequent program execution

"main program" means the process that made the initial asynchronous call. Both options return the registers to their values when the process was interrupted.

When you select SWIRET's second option, the software interrupt assumes the main program's process ID and environment, including the stack.

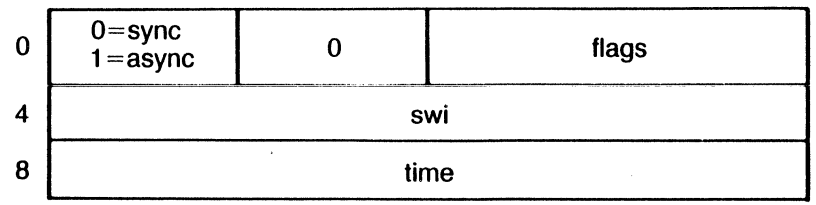
Use this option to return to a location in the main program other than the point of interruption or to assume the entire process identity without returning to the main program. Because the current condition of the stack is unknown when SWIRET is called, you should restore it to a known place before proceeding.

You can exit a program with SWIRET. Specify option 1 and call EXIT in your next instruction.

7.38 TIMER

C Interface:

```
UWORD       flags;
LONG        time;

ret = s_timer(flags,time);
emask = e_timer(swi,flags,time);

ret = __osif(F_TIMER,&parmblk);

parmblk:
```



Parameters:

flags bit 0: 1 = absolute
0 = relative

bits 1-15: Reserved

swi Address of software interrupt routine

time If bit 0 = 1 (absolute), number of milliseconds to delay after midnight. If bit 0 = 0 (relative), number of milliseconds to delay.

Return Code:

ret	Error Code

Description: The TIMER SVC delays the calling process until the specified time or the specified period of time expires. Use TIMER asynchronously with bit 0 = 0 (relative time) when you need a watchdog timer for an asynchronous SVC.

If absolute time is specified and the current time of day is beyond it, the process delays until the specified time the next day.

7.39 WAIT

C Interface:

```
LONG        events,cmask;

cmask = s_wait(events);

ret = __osif(F_WAIT,events);
```

Parameters:

events	Logical OR of emasks to wait for

Return Code:

cmask	Bit map of completed events

Description: The WAIT SVC causes the calling process to wait for an asynchronous event to occur. Specify one or more events by their emask in the WAIT events argument. FlexOS returns when one of these events has run to completion. For events that do not have a software interrupt, the cmask return code indicates which event completed. Subsequently, call the RETURN SVC to retrieve the return code of the completed event. This also releases that emask so it can be reused.

You can wait on events that have a software interrupt (swi). However, the event bit in the cmask returned is 0 rather than 1 when WAIT returns. Also, do not call RETURN to retrieve the completion code after WAIT returns--the completion is no longer available having already been provided to the swi for handling.

7.40 WRITE

C Interface:

```
LONG        fnum,bufsiz,offset,nbytes;
BYTE        option,*buffer;
UWORD       flags;

nbytes = s_write(flags,fnum,buffer,bufsiz,offset);
emask = e_write(swi,flags,fnum,buffer,bufsiz,offset);

ret = __osif(F_WRITE,&parmblk);

parmblk:
```

0	0=sync 1=async	option	flags
4	swi		
8	fnum		
12	buffer		
16	bufsiz		
20	offset		

Parameters:

option — May be used by SPECIAL devices

flags — bit 0: 1 = Flush buffers after WRITE.
This forces the data to the media.
If this is a zero length request,
the media is updated with any pending
writes.

0 = Allow optimized internal buffering

bit 1: 1 = Truncate file to size specified in
offset field. The bufsiz field
must be 0 to allow a truncate.

0 = Do not truncate

bits 2 - 7 are reserved (must be 0)

bits 8 and 9 determine how the offset field
is interpreted:

0 - Relative to beginning of file
1 - Relative to file pointer
2 = Relative to end of file

bits 10-15 are reserved (must be 0)

swi — Address of software interrupt routine

fnum — File number of file to write to

buffer — Address of buffer from which to write

bufsiz — Size in bytes of buffer

offset — Offset into file to start writing depending on bits 8 and 9.

Return Code:

nbytes — Number of bytes written. When nbytes is less than bufsize, an error ocurred during the write operation. An error code is returned only if no data was written before the error ocurred.

Error Code

Description:

The WRITE SVC places data into the specified file. Flags bits 8 and 9 determine whether the offset value is added to the beginning of file, the current file pointer, or the end of file. The offset can be a negative number, allowing a write to the last record of the file. Sequential I/O is performed by writing relative to the file pointer with an offset of 0.

The file pointer is updated on every write to be the byte position after the transferred data in the file. It is initialized to 0 at OPEN. Use the SEEK SVC to determine the current value of the file pointer.

The WRITE function verifies that the offset and bufsiz are on record boundaries if the file was created with a record size. No data is written if the values do not correspond.

The disk system has an asynchronous interface to allow for I/O redirection from the pipe or console systems. However, the disk system does not support asynchronous WRITE operations. An asynchronous WRITE to disk is slower and requires more memory than a synchronous WRITE.

7.41 XLAT

C Interface:

```
LONG        fnum,bufsiz;
UWORD       flags;
BYTE        *buffer;

ret = s_xlat(flags,buffer,bufsiz);

ret = __osif(F_XLAT,&parmblk);

parmblk:
```

0	0	0	flags
4	0		
8	0		
12	buffer		
16	bufsiz		

Parameters:

flags — bit 0: 1 = replace existing table with buffer contents
0 = add buffer contents to current table

bits 1 - 15 are reserved and must be 0.

buffer — Address of the buffer with the replacement or supplemental keystroke translations

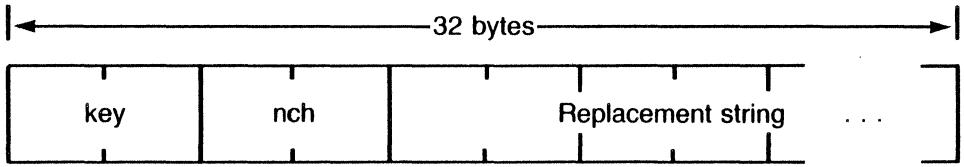
bufsiz — Size of buffer in bytes

Return Code:

ret	Error Code

Description: The XLAT SVC creates, replaces, or supplements a key translation table for the console specified by fnum. When the CONSOLE table KMODE (offset 2) bit 2 is 0, FlexOS translates characters entered from the keyboard into the string specified in the key translation table.

The key translation table consists of an unlimited number of 32 byte entries. Each entry is formatted as follows:



The fields are defined as follows:

- key: The 16-bit character to be translated; fill the high byte with a 0 for 8-bit input.
- nch: A WORD value indicating the number of 16-bit replacement characters; the maximum number of replacement characters is 14
- replacement: The replacement string; all characters are 16-bit

The key translation table is maintained on a per process basis. Child processes inherit their parent's table and share it until either process makes a change. This allows a parent to set up the keyboard environment before an application is run. When XLAT is called to change a table shared by two processes, FlexOS makes a separate copy for the calling process so that the modifications do not affect the other process.

There is no inherent limit to the number of translated keys supported for each process. The space for these keys are taken out of the Transient Program Area (TPA).

End of Section 7

System Tables

System status and parameter values are available to applications through the GET, SET, and LOOKUP SVCs which operate on a set of formalized data structures that comprise FlexOS's system tables. This section presents descriptions of the system tables in alphabetical order.

The GET SVC transfers the table to a buffer in the application's memory space. The SET SVC changes values in a table. For both SVCs, the table is identified by its number and, when that table type has more than one version, a unique ID number. The LOOKUP SVC searches for and retrieves tables of the same type. Each table that can be accessed with LOOKUP has a key value field; use this field to specify a starting point for the search.

The GET, SET, and LOOKUP SVCs will not access all of the system tables. Table 8-1 lists each of the system tables and the SVCs used to access them. Also listed in Table 8-1 are each table's number, ID, and key value.

Table 8-1. System Table Access

Table No. & Name	GET	SET	Unique ID	LOOK UP	Key	Description
0H PROCESS	X	X	pid	X	pid	Process information
1H ENVIRON	X	X	0			Process environment
2H TIMEDATE	X	X	0			System time of day
3H MEMORY	X		0			System memory use
10H PIPE	X		fnum	X	key	Pipe information
20H DISKFILE	X	X	fnum	X	key	Disk file information
21H DISK	X	X	fnum			Disk device information
30H CONSOLE	X	X	fnum			Console file information
31H PCONSOLE	X	X	fnum			Console device information
32H VCONSOLE	X	X	fnum	X	VCNUM	Console information
40H SYSTEM	X	X	0			Global system information
41H FILNUM	X		fnum	X	fnum	File number's table
42H SYSDEF				X	key	System logical name table
43H PROCDEF				X	key	Process logical name table
44H CMDENV	X		pid			Command environment
45H DEVICE				X	key	Device information
46H PATHNAME				X	none	Full path name
71H PRINTER	X	X	fnum			Printer device information
81H PORT	X	X	fnum			Port device information
82H+ SPECIAL	X	X	fnum			Special device information

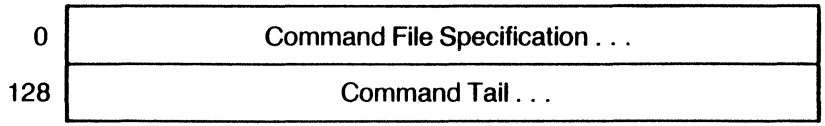
In the following system table descriptions, only those fields marked **R/W** are read-write; all other fields are read-only. In all bit-mapped values the bits for which there are no options are reserved and must be 0.

Note: FlexOS does not maintain memory representations for the tables described in this section. The corresponding resource manager or driver constructs them only when you call the GET, SET, or LOOKUP SVCs.

8.1 CMDENV Table

Number	GET?	SET?	LOOKUP?
44H	Yes	No	No

ID: 0 or process ID
Key: none



256 = Length in bytes.

The CMDENV table contains a process's command file specification and command tail. The strings are set by the COMMAND SVC. Both fields are 128 bytes in length and the strings are NULL terminated. The file specification includes the full pathname.

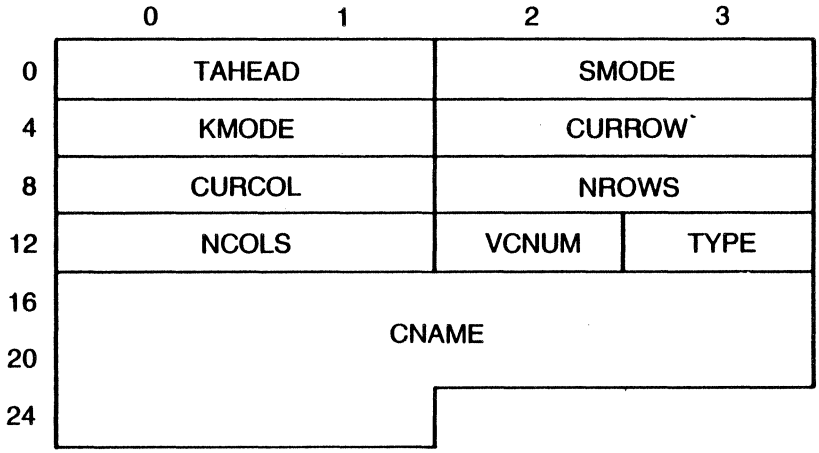
You can get the CMDENV table for the calling process or another process. For the calling process, specify an ID of 0 in the GET ID field. Otherwise, put the process ID of the target in the ID field.

8.2 CONSOLE Table

Number	GET?	SET?	LOOKUP?
30H	Yes	Yes	No

ID: File number of the console file
Key: none

The CONSOLE table describes the screen and keyboard of a con file.



26 = Length in bytes

- **TAHEAD:** Number of characters waiting in type-ahead buffer
- **SMODE (R/W):** Screen modes

  bit 0: 1 = Disable escape sequence decoding
  0 = Select Escape sequences supported

  bit 1: 1 = Characters are 16-bit values
  0 = Characters are 8-bit values

  bit 2: 1 = Convert <LF> to <CR><LF>
  0 = Do not convert <LF> or <CR>

- **KMODE (R/W)**: Keyboard mode

  bit 0: 1 = Disable Control-C
  0 = Control-C attempts external abort

  bit 1: 1 = Disable Control-S/Control-Q
  0 = Allow Control-S/Control-Q

  bit 2: 1 = Disable keyboard translation
  0 = Translate keys

  bit 3: 1 = Disable ESC sequence decoding
  0 = Support ESC sequence

  bit 4: 1 = Characters are 16-bit values
  0 = Characters are 8-bit values

  bit 5: 1 = Disable echo
  0 = Echo input characters on screen

  bit 6: 1 = Disable CTRL-Z
  0 = CTRL-Z = end of file

  bit 7: 1 = Enable toggle characters
  0 = Disable toggle characters

  bit 8: 1 = Convert <LF> or <CR> to <CR><LF>
  0 = Do not convert <LF> or <CR>

  bit 9: 1 = Do not echo carriage returns
  0 = Echo carriage returns

  bit 10: 1 = Do not echo <CR> on any delimiter
  0 = Echo <CR> on any delimiter

- **CURROW (R/W)**: Current cursor row position
- **CURCOL (R/W)**: Current cursor column position

- **NROWS:** Height of virtual screen in character rows
- **NCOLS:** Width of virtual screen in character columns
- **VCNUM:** Decimal number of virtual console
- **TYPE:** Type of virtual console

  bit 0: 1 = Graphics capability
  0 = Character only

  bit 1: 1 = No numeric keypad
  0 = Keypad

  bit 2: 1 = Mouse support
  0 = No mouse support

  bit 3: 1 = Color
  0 = Black and white

  bit 4: 1 = Memory-mapped video
  0 = Serial device

  bit 5: 1 = Currently in graphics mode
  0 = Currently in character mode

- **CNAME:** Physical console device name

Each console file opened has a corresponding CONSOLE table. The TAHEAD, CURROW, and CURCOL values are initialized to 0 when the console file is opened. NROWS and NCOLS correspond to the rows and columns set in the virtual console. SMODE and KMODE are initialized to 0; TYPE and CNAME are inherited from the parent console.

GET and SET the CONSOLE table using as the ID the file number returned when you OPENed the file vcxxx/console. Do not use the file number returned when you CREATEd the virtual console. For most applications, this file number is contained in the stdout--the screen file number--and stdin--the keyboard file number--in the ENVIRON table. Stdin and stdout can have the same or different file numbers.

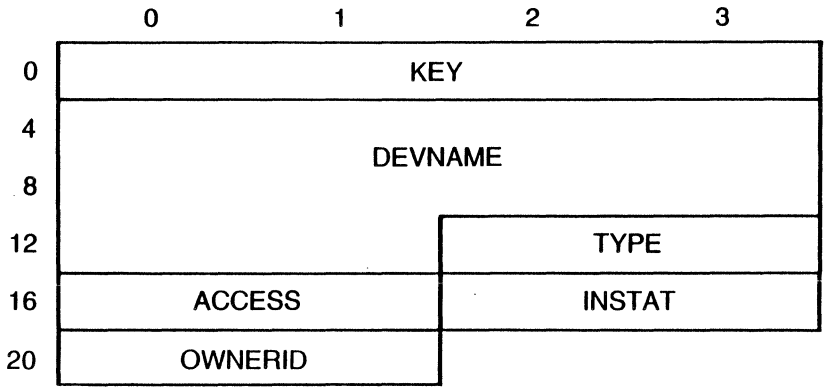
Use SET to change the cursor position and the screen and keyboard modes.

8.3 DEVICE Table

Number	GET?	SET?	LOOKUP?
45H	No	No	Yes

ID: none
Key: Key value assigned by resource manager

This table describes a physical device. Each device installed has a DEVICE table. All fields are read-only.



22 = Length in bytes

- **KEY:** Key field for LOOKUP
- **DEVNAME:** 10-byte device name
- **TYPE:** Type of device

  0xH - Kernel drivers
  2xH - Disk drivers
  3xH - Console drivers
  5xH - Extension drivers
  6xH - Network drivers
  7xh - Miscellaneous drivers
  80-FFH - Special drivers

- **ACCESS**: Access modes

  bit 0 1 = Delete allowed
  0 = Delete not allowed

  bit 1 Reserved

  bit 2 1 = Raw write allowed
  0 = Raw write not allowed

  bit 3 1 = Raw read allowed
  0 = Raw read not allowed

  bit 4 1 = Shared access allowed
  0 = Exclusive access only

  bit 5 1 = Removeable device
  0 = Permanent device

  bit 6 1 = Device lock (DEVLOCK) allowed
  0 = Device lock not allowed

  bit 7 1 = Shared access only
  0 = Exclusive access allowed

  bit 8* 1 = Device partitions allowed
  0 = Device partitions not allowed

  bit 9* 1 = Verify disk writes
  0 = Do not verify disk writes

  bits 10-15 reserved

  * Applicable to disk devices only.

- **INSTAT:** Installation status

  0x00 - Not installed
  0x01 - Requires subdriver
  0x02 - Owned by the Miscellaneous Resource Manager
  0x03 - Owned by another driver

- **OWNERID:** Significant 16 bits of the key field of the owner's DEVICE table entry. Use this value with a LOOKUP to find the driver that owns this subdriver. This field is only valid when INSTAT has a value of 0x03.

The DEVNAME, TYPE, ACCESS, and KEY values are established when the device is installed and do not change. The ACCESS flags override conflicting requests made by programs when they open the device.

The INSTAT and OWNERID values are also static except for subdrivers assigned to different drivers. In this case, the current values are subject to change as the driver is linked and unlinked to different owners.

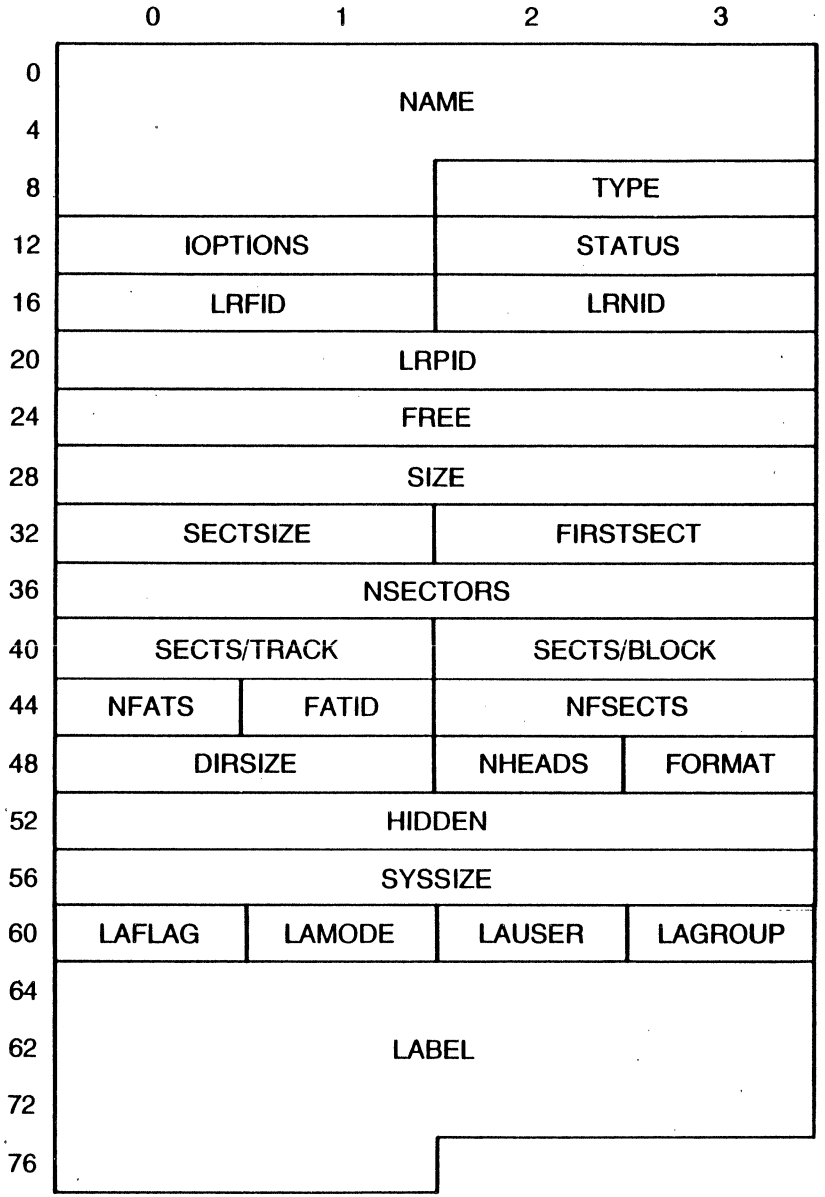
You must use the LOOKUP SVC to get DEVICE tables. Wildcards can be used in the LOOKUP device name specification.

8.4 DISK Table

Number	GET?	SET?	LOOKUP?
21H	Yes	Yes	No

ID: File number returned by OPEN
Key: none

The DISK table describes a disk driver. All fields are read-only except the label options.



78 = Length in bytes

- **NAME:** Disk device driver name
- **TYPE:** Media type

  bit 0: 1 = Removable media
  0 = Permanent media
- **IOPTIONS:** Install options

  bit 0: 1 = Set allowed
  0 = Set not allowed

  bit 1: Reserved

  bit 2: 1 = Raw write allowed
  0 = Raw write not allowed

  bit 3: 1 = Raw read allowed
  0 = Raw read not allowed

  bit 4: Reserved

  bit 5: 1 = Removable device driver
  0 = Permanent device driver

  bit 6: 1 = DEVLOCKs allowed
  0 = DEVLOCKs not allowed

  bit 7: Reserved

  bit 8: 1 = Partitioned disk driver
  0 = Non-partitioned disk driver

  bit 9: 1 = Verify after writes
  0 = Do not verify after writes

- **STATUS:** Disk status

  bit 0: 1 = Disk locked to process
  0 = Disk not locked to process

  bit 1: 1 = Disk locked to family
  0 = Disk not locked to family

  bit 2: 1 = Disk opened for exclusive access
  0 = Disk not opened for exclusive access

  bit 3: 1 = Disk currently in use by other processes
  0 = Disk not currently in use by other processes

  bit 4: 1 = Disk currently in use by processes in other families
  0 = Disk not currently in use by processes in other families

  bit 5: 1 = Disk activated for file system access
  0 = Disk not activated

  bit 6: 1 = File system files currently open
  0 = No open file system files

- **LRFID:** Family ID of locking process
- **LRNID:** Network node ID of locking process
- **LRPID:** Process ID of locking process
- **FREE:** Number of bytes of free space on disk/partition
- **SIZE:** Size in bytes of total file space on disk/partition
- **SECTSIZE:** Sector size in bytes
- **FIRSTSEC:** First sector of logical media
- **NSECTORS:** Number of sectors on disk
- **SECTS/TRACK:** Number of sectors per track
- **SECTS/BLOCK:** Number of sectors per block
- **NFATS:** Number of File Allocation Tables (FATs)

- **FATID:** Implementation-dependent value indicating media format
- **NFSECTS:** Number of sectors per FAT
- **DIRSIZE:** Maximum number of directory entries in root directory
- **NHEADS:** Number of heads on disk
- **FORMAT:** FAT format

  0 - Raw
  1 - 1 1/2 byte FATs
  2 - 2 byte FATs

- **SYSSIZE:** Size of system area in bytes
- **HIDDEN:** Number of hidden sectors on partitioned' disk
- **LAFLAG:** Label flag

  0 - Label does not exist on media
  1 - Label exists
  2 - Return device error on attempts to read label

- **LAMODE (R/W):** Label mode

  bit 0: 1 = File security enabled
  0 = No file security

  bit 1: 1 = Upper and lower case file names
  0 = Upper case file names only

- **LAUSER (R/W):** Label maker's User ID
- **LAGROUP (R/W):** Label maker's Group ID
- **LABEL (R/W):** 11-character label (also referred to as volume) name. Bytes 12 through 14 in LABEL data block are ignored. The name does not need to be null-terminated.

Most of the DISK table's read-only fields are static. The exceptions are:

- STATUS which changes as processes lock, unlock, open, and close files.
- FREE and DIRSIZE which increase and decrease as files are removed and added.
- LRFID, LRNID, and LRPID which change with each change in the locking process.

Use the file number returned by OPEN as the ID in your GET and SET calls.

All of the label-related fields are read/write. However, once they have been set, only the superuser (user and group IDs 0) or the original label setter can make any changes.

8.5 DISKFILE Table

Number	GET?	SET?	LOOKUP?
20H	Yes	Yes	Yes

ID: File number returned by CREATE or OPEN
Key: Key number assigned by resource manager

The DISKFILE table describes a disk file. The disk file must be open before you can GET its table. To SET values in the table, the calling process must have the same USER and GROUP IDs or have GROUP and USER numbers 0. Files do not need to be open for the LOOKUP SVC. LOOKUP flag bits determine the type of file to search for and are used as follows:

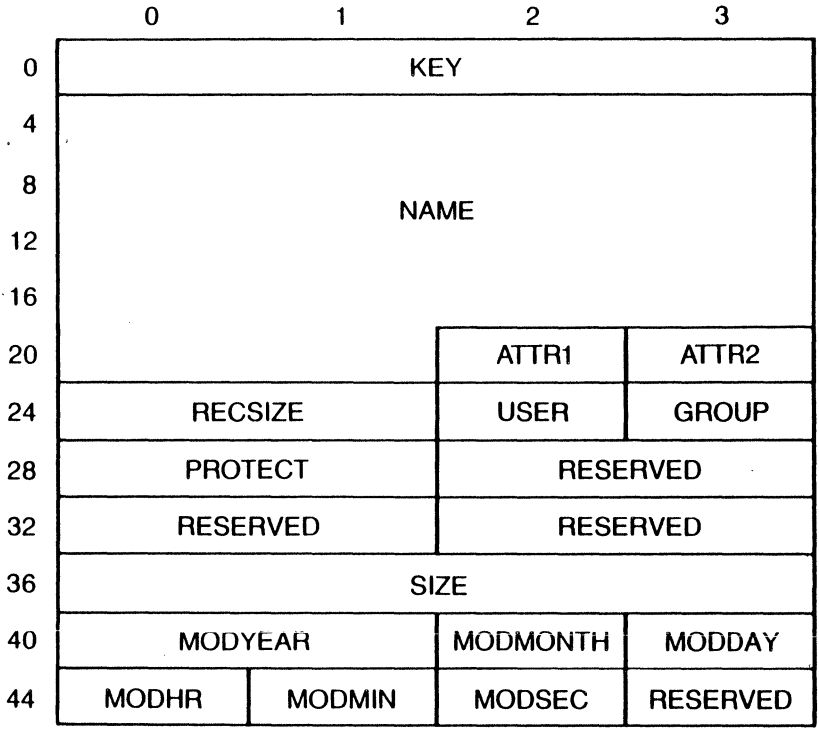
bit 0: 1 - Include HIDDEN files
0 = Exclude HIDDEN files

bit 1: 1 = Include SYSTEM files
0 = Exclude SYSTEM files

bit 2: 1 = Include VOLUME label
0 = Exclude VOLUME label

bit 3: 1 = Include directories
0 = Exclude directories

bit 4: 1 = Exclude normal files
0 = Include normal files



48 = Length in bytes

- **KEY:** Key field for LOOKUP
- **NAME:** Disk file name

- **ATTR1 (R/W):** The sum of the following file attributes:

  01H Read-only
  02H Hidden
  04H System
  08H Volume label
  10H Subdirectory
  20H Archive attribute
  40H Reserved
  80H Reserved

- **ATTR2 (R/W):** The sum of the following file attributes:

  01H Security enabled (label only)
  02H Disk supports uppercase and lowercase file names (if not set, disk supports uppercase file names only)

  04H through 80H are reserved.

- **RECSIZE:** Record size
- **USER (R/W**):** User ID of owner
- **GROUP (R/W**):** Group ID of owner
- **PROTECT (R/W):** File Security Word
- **SIZE:** File size
- **MODYEAR (R/W):** Year of last modification
- **MODMONTH (R/W):** Month of last modification (1 - 12)
- **MODDAY (R/W):** Day of last modification (1 - 31)
- **MODHR (R/W):** Hour of last modification (0 - 23)
- **MODMIN (R/W):** Minute of last modification (0 - 59)
- **MODSEC (R/W):** Second of last modification (0 - 59)

** These fields are read/write to a superuser only.

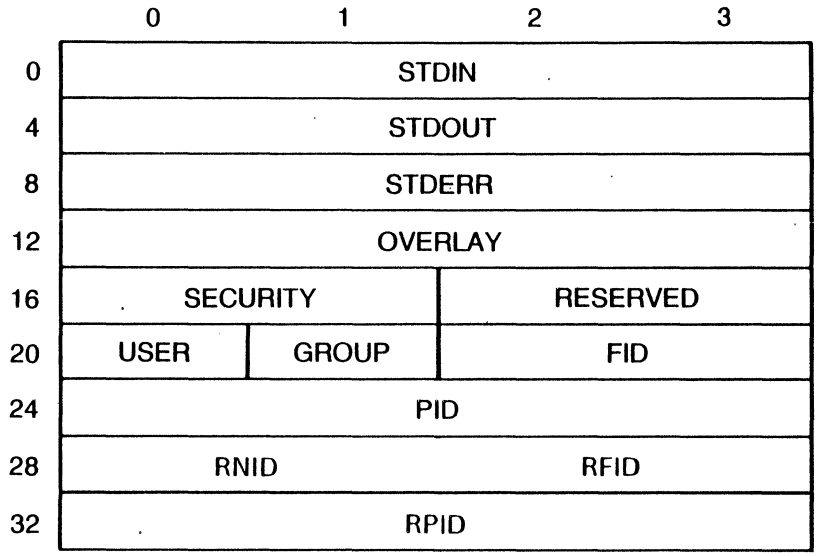
All DISKFILE values are set and updated by the Disk Resource Manager. This does not preclude setting these values yourself. However, you should exercise caution when modifying the attributes and record size.

8.6 ENVIRON Table

Number	GET?	SET?	LOOKUP?
1H	Yes	Yes	No

ID: 0
Key: none

The ENVIRON and PROCESS tables describe the calling process's environment. Although there is some overlap between the two, the standard input, output, error, and overlay file numbers, file security word, and requester node numbers are unique to the ENVIRON table.



36 = Length in bytes

- **STDIN (R/W):** Process's standard input file number
- **STDOUT (R/W):** Process's standard output file number
- **STDERR (R/W):** Process's standard error file number

- **OVERLAY (R/W)**: Current program's file number
- **SECURITY (R/W)**: Default file security word for CREATE
- **USER (R/W**)**: Current User ID
- **GROUP (R/W**)**: Current Group ID
- **FID**: Calling process's family ID
- **PID**: Calling process's ID
- **RNID (R/W*)**: Requester process's remote node ID
- **RFID (R/W*)**: Requester process's family ID
- **RPID (R/W*)**: Requester process's process ID

**These fields are read/write for a superuser only.

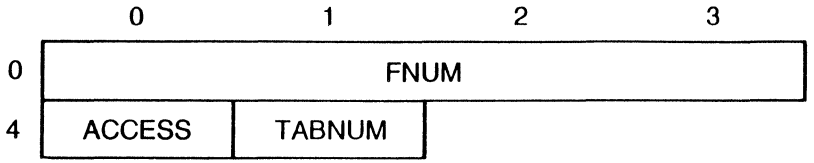
The RNID, RFID and RPID fields are used by network server processes only. See the FlexNet Network Operating System OEM and Programmer's Guide for the description of their use.

8.7 FILNUM Table

Number	GET?	SET?	LOOKUP?
41H	Yes	No	Yes

ID: File number returned by CREATE or OPEN
Key: File number returned by CREATE or OPEN

The FILNUM table provides the table number for a given file number. For example, the Console Resource Manager returns the VCONSOLE table number when you GET the FILNUM table for a virtual console file.



6 = Length in bytes

- **FNUM:** File number and key field for LOOKUP
- **ACCESS:** Access privileges returned from OPEN call

  bit 0: 1 = Delete/set access
  0 = No delete/set access

  bit 1: 1 = Execute access
  0 = No execute access

  bit 2: 1 = Write access
  0 = No write access

  bit 3: 1 = Read access
  0 = No read

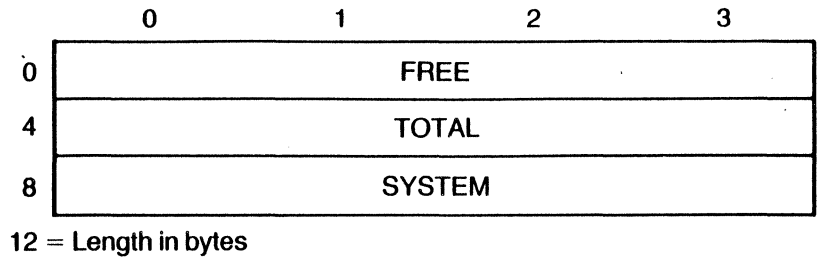
- **TABNUM:** table number for that type of file's table

8.8 MEMORY Table

Number	GET?	SET?	LOOKUP?
3H	Yes	No	No

ID: 0
Key: none

The MEMORY table indicates the system memory usage. The FREE and SYSTEM values change as processes use and release memory and the resource managers take up transient program area.



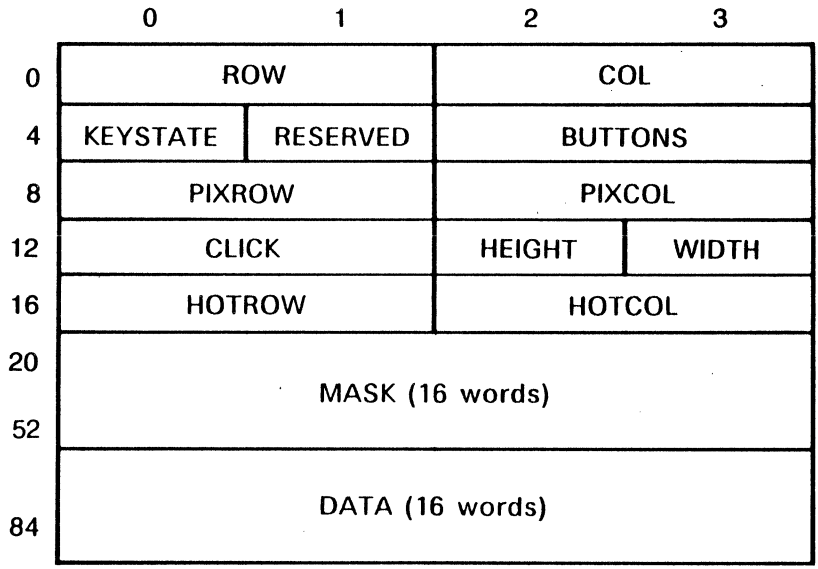
- **FREE:** Total free memory in bytes
- **TOTAL:** Total memory in bytes
- **SYSTEM:** Size of system memory in bytes

8.9 MOUSE Table

Number	GET?	SET?	LOOKUP?
33H	Yes	Yes	No

ID: File number returned by OPEN
Key: none

The MOUSE table describes a pointing device. Every installed pointing device has a MOUSE table. The initial values are set by the driver and you can set all of them except for the PIXROW and PIXCOL.



- **ROW (R/W)**: Current row position of mouse
- **COL (R/W)**: Current column position of mouse

- **KEYSTATE**: Keyboard state of the right Shift, left Shift, Control, and Alt keys

  Bit 0 right Shift key
  Bit 1 left Shift key
  Bit 2 Control key
  Bit 3 Alt key

  0 - up position
  1 - down position

- **PIXROW**: Number of mickeys per pixel for rows
- **PIXCOL**: Number of mickeys per pixel for columns
- **CLICK (R/W)**: Click interval in milliseconds
- **HEIGHT (R/W)**: Height of mouse form
- **WIDTH (R/W)**: Width of mouse form
- **HOTROW (R/W)**: Hot row of mouse form
- **HOTCOL (R/W)**: Hot column of mouse form
- **MASK (R/W)**: On a bit map screen, a 16 x 16 pixel rectangle that masks the effect of the DATA rectangle.
- **DATA (R/W)**: On a bit map screen, a 16 x 16 pixel rectangle to "BLT" to the screen given the mask.

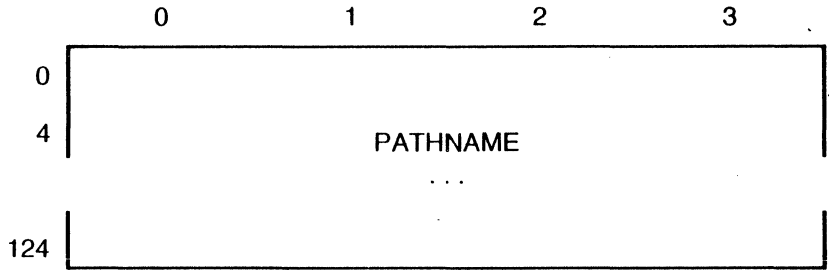
The ROW and COL values are updated by the Console Resource Manager to indicate the current mouse location. You can, however, set these values to move the mouse form to a location without device input. The HEIGHT and WIDTH values have a maximum value of 4, but can be less. If either is less, the length of the MASK and DATA fields is not affected.

8.10 PATHNAME Table

Number	GET?	SET?	LOOKUP?
46H	No	No	Yes

ID: none
Key: none

The PATHNAME table contains the fully-expanded path name for a defined symbol. LOOKUP is the only way to retrieve a PATHNAME table; you cannot SET or GET a PATHNAME.



128 = Length in bytes.

The PATHNAME table consists of a single 128 byte field. Only one path is ever returned when you lookup a defined symbol. If the symbol specified starts with a defined name, the prefix is substituted for the symbol. If the first name in the prefix is itself a defined symbol, the substitution is made again. The search and substitute routine is repeated until no prefix is found for the starting name.

The SYSDEF and PROCDEF tables are searched when you lookup the PATHNAME table. (DEFINE only looks in one or the other.) These tables are searched for the first name in the specification only.

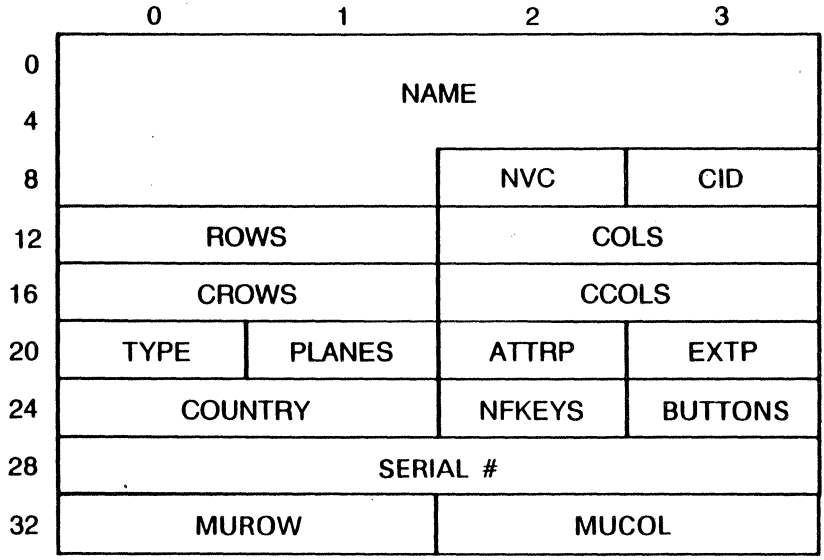
Wildcard characters can be used but they are not expanded; for example, as asterisk is interpreted only as an asterisk.

8.11 PCONSOLE Table

Number	GET?	SET?	LOOKUP?
31H	Yes	Yes	No

ID: File number returned by OPEN
Key: none

The PCONSOLE table describes a physical console device. Each console installed has its own PCONSOLE table. All parameters are read-only except the country code.



36 = length in bytes

- **NAME:** Console device name
- **NVC:** Current number of virtual consoles
- **CID:** Physical console ID number

- **ROWS:** On graphic console devices, this is the number of rows of pixels. On character console devices, this is the number of character rows and is the same as CROWS.
- **COLS:** On graphic console devices, this is the number of pixels in a row. On character console devices, this is the number of character columns and is the same as CCOLS.
- **CROWS:** The number of rows of characters
- **CCOLS:** The number of columns of characters
- **TYPE:** Type of console

  bit 0: 1 = Graphics capability
  0 = Character only

  bit 1: 1 = No numeric keypad
  0 = Keypad

  bit 2: 1 = Mouse supported
  0 = No mouse supported

  bit 3: 1 = Color
  0 = Black and white

  bit 4: 1 = Memory-mapped video
  0 = Serial device

  bit 5: 1 = Currently in graphics mode
  0 = Currently in character mode

- **PLANES:** Planes supported

  Bit 0: 1 = Character plane supported
  0 = No character plane

  Bit 1: 1 = Attribute plane supported
  0 = No attribute plane

  Bit 2: 1 = Extension plane supported
  0 = No extension plane

- **ATTRP:** Bit map of attribute plane bits supported
- **EXTP:** Bit map of extension plane bits supported
- **COUNTRY (R/W):** Country code; in applications that support multiple character sets, use this value to select a specific set. Appendix C lists the country codes.
- **NFKEYS:** Number of function keys supported
- **BUTTONS:** Number of mouse buttons supported
- **SERIAL #** Mouse serial number
- **MUROW** Mouse sensitivity in mickey units per row
- **MUCOL** Mouse sensitivity in mickey units per column

The PCONSOLE values are set by the driver. The Console Resource Manager updates the NVC value as you create and delete virtual consoles on this console.

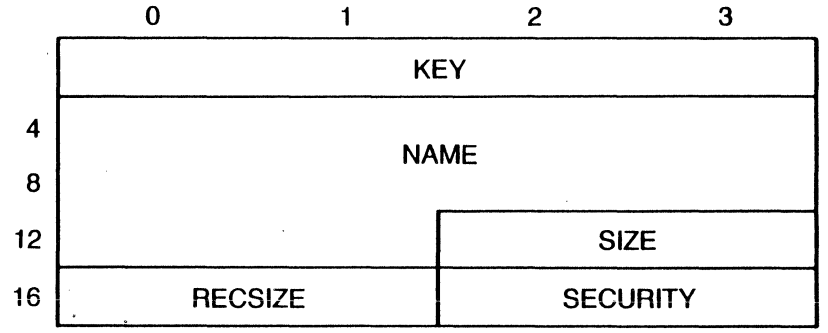
To GET and SET a PCONSOLE table (LOOKUP cannot be used), OPEN the device and use the file number returned as the GET and SET ID number. In your OPEN call, the only access mode flag bit you can set is bit 0 and you only need set it if you want to change the country code.

8.12 PIPE Table

Number	GET?	SET?	LOOKUP?
10H	Yes	No	Yes

ID: File number returned by CREATE or OPEN
Key: Key number assigned by resource manager

The PIPE table describes a pipe. All fields are set when you create the pipe and are read-only.



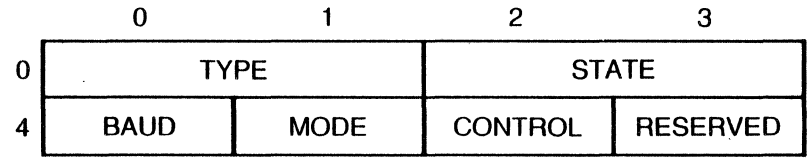
- **KEY:** Key field for LOOKUP
- **NAME:** 10-byte pipe name
- **SIZE:** Internal buffer size of pipe
- **RECSIZE:** Record size
- **SECURITY:** File security word

You can retrieve a pipe table with GET or LOOKUP. Use the the file number returned when you CREATEd or OPENed the pipe as your GET ID number. In a LOOKUP call, use the pipe name.

8.13 PORT Table

Number	GET?	SET?	LOOKUP?
81H	Yes	Yes	No

ID: File number returned by OPEN
Key: None



8 = Length in bytes

- **TYPE:** Type of port

  0 = Undefined
  1 = Standard serial driver
  2 = Character I/O device
  4 = Standard parallel driver

- **STATE (R/W):** Current state of port

  0 = Transmit enable
  1 = Character has been received
  2 = Change in Data Set Ready or Data Carrier Detect
  3 = Parity error
  4 = Overrun error
  5 = Framing error
  6 = Carrier present (Data Carrier Detect)
  7 = Data Set Ready (DSR) active

- **BAUD (R/W):** A value selecting the baud rate

0 = 50 baud	6 = 600 baude	12 = 4800 baud
1 = 75 baud	7 = 1200 baud	13 = 7200 baud
2 = 110 baud	8 = 1800 baud	14 = 9600 baud
3 = 134.5 baud	9 = 2000 baud	15 = 19200 baud
4 = 150 baud	10 = 2400 baud	
5 = 300 baud	11 = 3600 baud	

- **MODE (R/W):** Bit-mapped description of the word length, parity and stop bits

Value	Bits 0-1 Bits/word	Bits 2-3 Stop Bits	Bits 4-5 Parity
0	5	None	None
1	6	1	Odd
2	7	1.5	
3	8	2	Even

- **CONTROL (R/W):** Bit-mapped description of serial port control parameters

  0 = Enable character transmission
  1 = Force Data Terminal Ready low
  2 = Enable character reception
  3 = Force break signal
  4 = Reset error
  5 = Force Request to Send low

Use the GET and SET SVCs to retrieve and set PORT table values. The ID is the file number returned when the device was opened. When the port is a subdriver, you cannot access the table directly with GET or SET. Instead, use SPECIAL functions 13H and 93H, respectively.

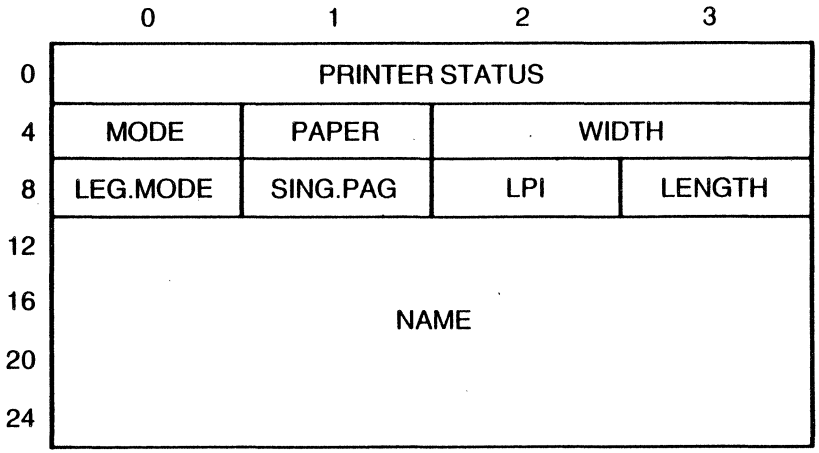
For standard parallel drivers, the STATE, BAUD, MODE, and CONTROL fields are meaningless.

8.14 PRINTER Table

Number	GET?	SET?	LOOKUP?
71H	Yes	Yes	No

ID: File number returned by OPEN
Key: None

The PRINTER table describes an installed printer driver. The printer driver may drive the physical I/O port directly or require a subdriver to conduct character I/O. For all bit maps in this table, the least significant bit is rightmost.



28 = Length in bytes

- **PRINTER STATUS:** Bit map indicating current status; bits are assigned as follows:

Bit	Set Definition	Bit	Set Definition
0	Off line	4	Illegal mode requested
1	Out of paper	5	Framing error
2	Select error	6	Internal buffer full
3	Initialization error	7	Waiting for XON

- **MODE (R/W):** A bit map used to select the current typeface; the bits are assigned as follows:

Bit	Typeface Selected	Bit	Typeface Selected
0	Boldface	4	Superscript
1	Graphics	5	Condensed
2	Italic	6	Elongated
3	Subscript	7	Letter quality

- **PAPERTYP (R/W):** A bit map indicating the current paper type; the default is 8 1/2 x 11. The bits are assigned as follows:

Bit	Paper Type
0	Wide paper
1	Letterhead
2	Labels

- **WIDTH (R/W):** Width of paper in columns for all modes except graphics; in dots if graphics mode.
- **LENGTH (R/W):** Length of paper in lines
- **LEG.MODE:** Bit map of modes available; bit assignments are the same as MODE above.
- **SING.PAG (R/W):** Single-page paper feed select; non-zero when single-page feed mechanism selected.
- **LPI (R/W):** Number of lines printed per inch
- **NAME:** 16-byte field for the brand and mode of the printer in ASCII.

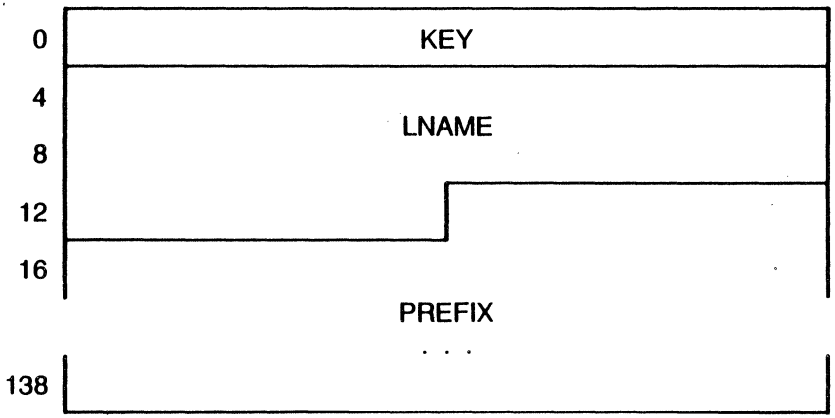
To retrieve a PRINTER table, use the file number returned when the device was opened as the GET ID number.

8.15 PROCDEF Table

Number	GET?	SET?	LOOKUP?
43H	No	No	Yes

ID: none
Key: Key number assigned by resource manager

The PROCDEF table shows the prefix defined for a logical name by the calling process. The LNAME and PREFIX fields are set by the DEFINE call; the key value is set by the resource manager when the name is defined. All fields are read-only.



142 = Length in bytes

- **KEY** Key field for LOOKUP.
- **LNAME:** 10-byte, null terminated logical name string
- **PREFIX:** 128-byte, null terminated prefix substitution string

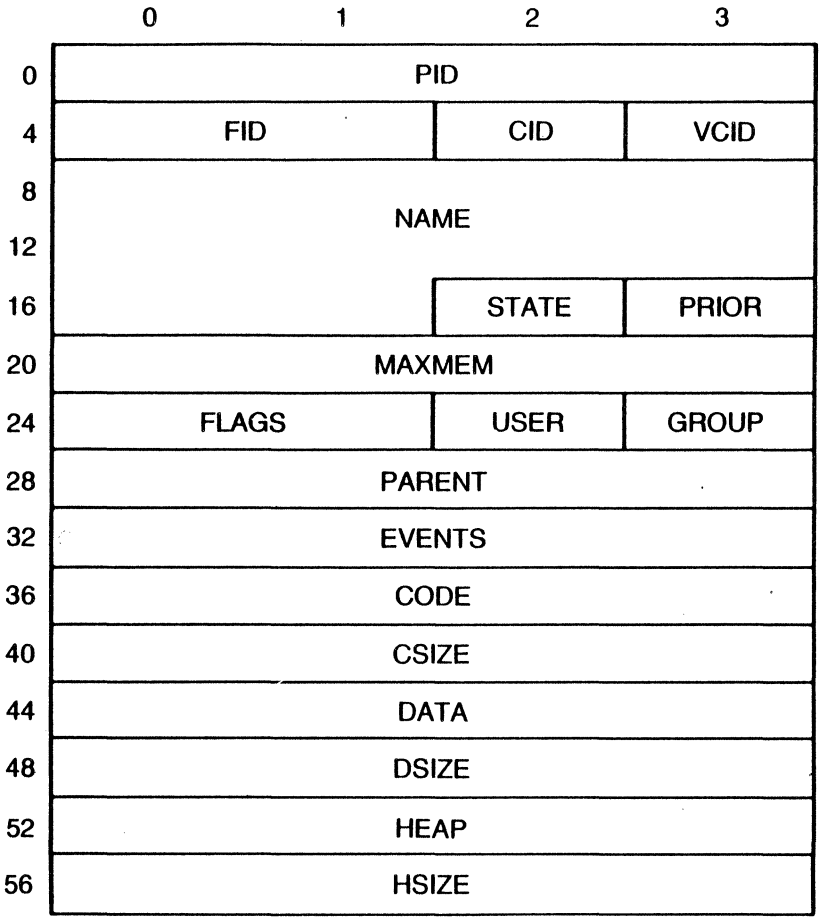
Use LOOKUP to get a PROCDEF table. Use the logical name (wildcards can be used) or key value to specify a table. The maximum name and prefix length is 9 and 127 characters, respectively; the null character is always included in the specification.

8.16 PROCESS Table

Number	GET?	SET?	LOOKUP?
0H	Yes	Yes	Yes

ID: Process ID
Key: Process ID

The PROCESS and ENVIRON tables combine to describe a process's environment. The PROCESS table values are set when the process is created (see the COMMAND SVC description) and maintained by the resource managers. All values are read-only except the priority. This value can be set by a process with the same USER and GROUP numbers or USER and GROUP numbers 0.



60 = Length in bytes

- **PID:** Process ID
- **FID:** Process's family ID
- **CID:** Physical console device number
- **VCID:** Process's virtual console number--only filled after console is opened
- **NAME:** Process name
- **STATE:** Process state

  0 - Run
  1 - Waiting
  2 - Terminating

- **PRIOR (R/W):** Priority
- **MAXMEM** Maximum memory allowed
- **FLAGS:**

  bit 0: 1 = System process
  0 = User process

  bit 1: 1 = Locked in memory
  0 = Swappable

  bit 2: 1 = Running in SWI context
  0 = Running in main context

  bit 3: 1 = Originally a privileged process[1]
  0 = Not originally a privileged process

- **USER:** User number
- **GROUP:** Group number

[1] A privileged process, also called a superuser, is one with USER and GROUP numbers 0.

- **PARENT:** Parent process ID
- **EVENTS:** Bit map of events that have completed but whose return values have not been retrieved.
- **CODE:** Start of code area in user space
- **CSIZE:** Size in bytes of code area
- **DATA:** Start of data area in user space
- **DSIZE:** Size in bytes of data area
- **HEAP:** Start of heap area in user space[2]
- **HSIZE:** Size in bytes of most recently allocated heap area

Use the process ID as the ID in GET and SET calls and as the key value in LOOKUP calls. You can also use the process NAME with LOOKUP. A process ID of 0 selects the calling process.

[2]Information passed to the process in the COMMAND SVC is stored at this location.

8.17 SPECIAL Table

Number	GET?	SET?	LOOKUP?
82H-FFH	Yes	Yes	No

ID: File number returned by OPEN
Key: Key number assigned by resource manager

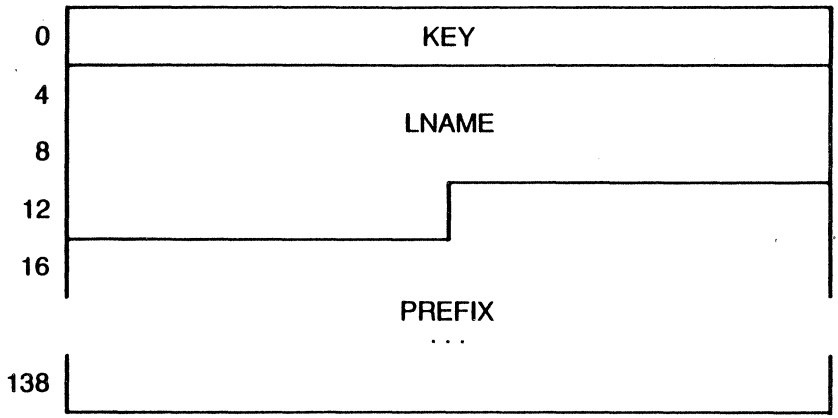
SPECIAL tables describe special devices installed. The format and size of a SPECIAL table is defined by the OEM and set by the device driver. There are two rules, however, for all SPECIAL tables: the first word indicates the size of the table and table number is the same number as the device type.

8.18 SYSDEF Table

Number	GET?	SET?	LOOKUP?
42H	No	No	Yes

ID: none
Key: Key number assigned by resource manager

The SYSDEF table describes the system's logical names. Logical name assignments are made with the DEFINE SVC by privileged (USER and GROUP numbers are 0) processes. Privileged processes can also change existing assignments.



- **KEY:** Key field for LOOKUP.
- **LNAME (R/W):** 10-byte, null terminated Logical name string.
- **PREFIX (R/W):** 128-byte, null terminated prefix substitution string.

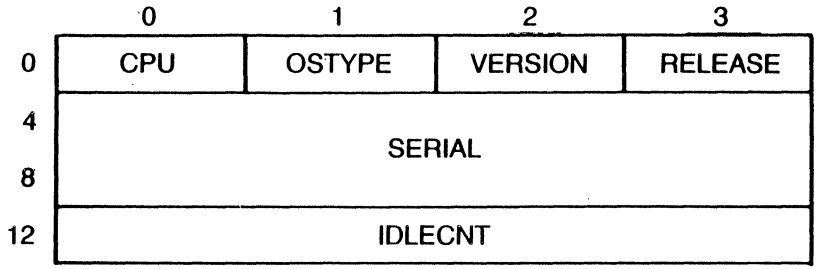
Use LOOKUP to get a SYSTEM table. Use the logical name or key value to specify a table. The maximum name and prefix length is 9 and 127 characters, respectively; the null character is always included in the specification.

8.19 SYSTEM Table

Number	GET?	SET?	LOOKUP?
40H	Yes	Yes	No

ID: 0
Key: none

The SYSTEM table describes the CPU and operating system. All fields are read-only except the IDLECNT. This field is read-write to processes with USER and GROUP numbers 0 only.



16 = Length in bytes

- **CPU**: Type of CPU

  1 - Intel 8080
  2 - Intel 8085
  3 - Zilog Z80
  4 - Intel 8086
  5 - Zilog Z8000
  6 - Motorola 68000
  7 - Motorola 68010
  8 - Motorola 68020
  9 - Intel 80286
  10 - Intel 80386
  11 - Intel 80186
  12 - 255 Reserved

- **OSTYPE**: Type of Operating System

  0 FlexOS
  1-255 Reserved

- **VERSION**: Operating system version number

- **RELEASE:** Release level of version
- **SERIAL:** 8-byte, operating system serial number
- **IDLECNT (R/W for privileged user only):** CPU System idle count

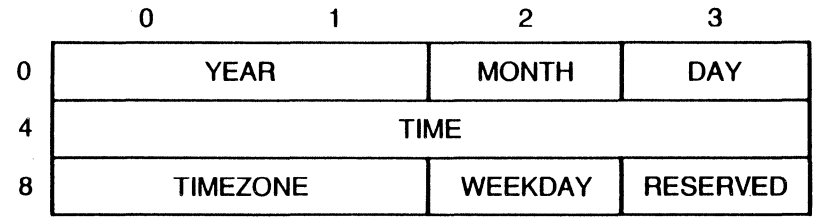
IDLECNT is a value incremented by the CPU when no process is running. Monitor CPU utilization by setting this value to 0 and after a known period of time, GETting the count.

8.20 TIMEDATE Table

Number	GET?	SET?	LOOKUP?
2H	Yes	Yes	No

ID: 0
Key: none

The TIMEDATE table contains the system time of day. All fields are read/write except WEEKDAY. The time is maintained by the kernel once the starting is set. Use SET to establish the starting time.



12 = Length in bytes

- **YEAR (R/W):** Year; a literal value (for example, 1987 = 1987)
- **MONTH (R/W):** Month; 1 - 12
- **DAY (R/W):** Day of the month; 1 - 31
- **TIME (R/W):** Number of milliseconds since midnight
- **TIMEZONE (R/W):** Minutes from Universal Coordinated Time
- **WEEKDAY:** Day of the week; 0 = Sunday, 6 = Saturday

You use an ID value of 0 to GET and SET the TIMEDATE table.

8.21 VCONSOLE Table

Number	GET?	SET?	LOOKUP?
32H	Yes	Yes	Yes

ID: File number returned by CREATE
Key: VCNUM assigned when virtual console created

The VCONSOLE table describes a virtual console. Table values are established when you CREATE the console. Use read/write fields to modify window size, location on the virtual console, and placement on the parent console.

	0	1	2	3
0	KEY			
4	MODE		VCNUM	TYPE
8	VIEWROW		VIEWCOL	
12	NROW		NCOL	
16	POSROW		POSCOL	
20	ROWS		COLS	
24	TOP	BOTTOM	LEFT	RIGHT

28=Length in bytes

- **KEY:** Key field for LOOKUP

- **MODE (R/W):** Window mode

  bit 0: 1 = Freeze borders
  0 = Synchronize borders (See Note 1, below)

  bit 1: 1 = Allow auto view change (See Note 2, below)
  0 = Keep view fixed

  bit 2: 1 = Keep cursor on edge on auto view change
  0 = Center cursor on auto view change

  bit 3: 1 = Auto view change on output
  0 = Auto view change on input

- **VCNUM:** Decimal virtual console number

- **TYPE:** Type of console.

  bit 0: 1 = Graphics capability
  0 = Character only

  bit 1: 1 = No numeric keypad
  0 = Keypad

  bit 2: Reserved

  bit 3: 1 = Color
  0 = Black and white

  bit 4: 1 = Memory-mapped video
  0 = Serial device

  bit 5: 1 = Currently in graphics mode
  0 = Currently in character mode

- **VIEWROW (R/W):** Row coordinate on the virtual console of window's upper lefthand corner

- **VIEWCOL (R/W):** Column coordinate on the virtual console of the window's upper lefthand corner

- **NROW (R/W)**: Number of character rows in the window
- **NCOL (R/W)**: Number of character columns in the window
- **POSROW (R/W)**: Row coordinate on parent console of window's upper lefthand corner
- **POSCOL (R/W)**: Column coordinate on parent console of window's upper lefthand corner
- **ROWS**: Number of character rows in the virtual console
- **COLS**: Number of character columns in the virtual console
- **TOP**: Height in character rows of the top border
- **BOTTOM**: Height in character rows of the bottom border
- **LEFT**: Width in character columns of the left border
- **RIGHT**: Width in character columns of the right border

Notes:

1. Use bit 0 to freeze a border so that intermediate states are not displayed when you make changes to the border file contents. Before you change the border file contents, set this bit. After you have completed the changes, reset the bit. Normally, keep this flag at 0 so that the borders change as you make changes to the window dimensions and location.
2. Bits 1 through 3 determine whether the window view changes to keep the cursor on-screen or the view remains fixed on the same virtual console coordinates regardless of cursor location. If the cursor leaves the window and bit 2 = 1, bit 3 determines whether the view changes when the cursor leaves the view (output) or when the application READs the keyboard.

Use the file number returned by the CREATE call to GET or SET the VCONSOLE table. Alternatively, use the key value in a LOOKUP call. Changes made to the VIEWROW, VIEWCOL, NROW, and NCOL immediately affect the shape and position of the window on the virtual console. Border files are automatically adjusted accordingly as well. Changes to POSROW and POSCOL are immediately reflected on the parent console.

End of Section 8

Character Sets and Escape Sequences

This appendix describes the Console Resource Manager's built-in escape sequences and character sets. The presentation begins with the description of the 8-bit escape sequences (a superset of the VT-52 escape sequences), continues with the description of the 16-bit output character set, and concludes with the description of the 16-bit input character set.

Output escape sequence decoding is only available with the WRITE SVC. You cannot use COPY or ALTER to output escape sequences.

The descriptions below cross-reference bits in the CONSOLE table's SMODE and KMODE fields. See Section 8, "System Tables," for the complete description of these fields.

A.1 Escape Sequences

You select escape sequence decoding to manipulate the screen display by setting bits 0 and 1 of the CONSOLE table's SMODE word to 0. Escape sequence decoding of keyboard input is selected by setting bits 3 and 4 of the CONSOLE Table's KMODE word to 0.

An escape sequence consists of at least two 8-bit characters, where the first is always an ESC (ASCII character 1B hex). The second character selects a function. Three functions require additional numeric values to select a foreground or background color or set the cursor position. Table A-1 lists the functions supported and the escape sequence that invokes it.

Table A-1. Escape Sequence Functions

ESC Sequence	Description
<ESC>A	**Cursor Up:** Move cursor up to beginning of previous line.
<ESC>B	**Cursor Down:** Move cursor down one line without changing column position.
<ESC>C	**Cursor Right:** Move cursor one character position right.
<ESC>D	**Cursor Left:** Move cursor one character position left.
<ESC>H	**Cursor Home:** Move cursor to first column of first line.
<ESC>I (uppercase i)	**Reverse Index:** Move cursor up one line without changing column position.
<ESC>j	**Save Cursor Position:** Store current cursor position for subsequent restore.
<ESC>k	**Restore Cursor Position:** Move cursor to position previously saved.
<ESC>Y(c_1)(c_2)	**Set Cursor Position:** Move cursor to specified coordinates; first character is the ASCII equivalent of the row number + 31D and second character is ASCII equivalent of column number + 31D.
<ESC>E	**Clear Display:** Erase entire screen and home cursor.
<ESC>J	**Erase to End of Display:** Erase from cursor to end of display.
<ESC>K	**Erase to End of Line:** Erase from cursor to end of line.

Table A-1. (Continued)

ESC Sequence	Description
<ESC>l (lowercase L)	**Erase Entire Line:** Erase current line contents.
<ESC>d	**Erase Beginning of Display:** Erase from beginning of display through cursor.
<ESC>o	**Erase Beginning of Line:** Erase from beginning of line through cursor.
<ESC>L	**Insert Blank Line:** Move current and all subsequent lines down one line; keep cursor on current line.
<ESC>M	**Delete Line:** Remove current line from display and add blank line at bottom.
<ESC>N	**Delete Character:** Remove character at cursor.
<ESC>b(n)	**Set Foreground Color:** Set character color for current cursor position where n is a decimal value that determines the color as follows: 0 - Black 1 - Blue 2 - Green 3 - Cyan 4 - Red 5 - Magenta 6 - Brown 7 - Light Gray 8 - Dark gray 9 - Light blue 10 - Light green 11 - Light cyan 12 - Light red 13 - Light magenta 14 - Yellow 15 - White

Table A-1. (Continued)

ESC Sequence	Description
<ESC>c(n)	**Set Background Color**: Set screen color for current cursor position where n is a decimal value that determines the color as follows: 0 - Black 1 - Blue 2 - Green 3 - Cyan 4 - Red 5 - Magenta 6 - Brown 7 - Light Gray 8 - 15 are the same as 0 - 7, except the foreground blinks.
<ESC>e	**Enable Cursor**: Show cursor.
<ESC>f	**Disable Cursor**: Remove cursor.
<ESC>p	**Enter Reverse Video Mode**: Swap foreground and background colors.
<ESC>q	**Exit Reserve Video Mode**: Return to original foreground and background color scheme.
<ESC>r	**Enter Intensify Mode**: Turn on the console's intensity option.
<ESC>u	**Exit Intensify Mode**: Turn off the console's intensity option.

Table A-1. (Continued)

ESC Sequence	Description
<ESC>s	**Enter Blink Mode:** Start character blinking for all characters.
<ESC>t	**Exit Blink Mode:** Stop character blinking.
<ESC>@	**Enter Insert Mode:** Insert subsequent characters from current cursor position, moving existing characters over; characters pushed off end of line are lost.
<ESC>O	**Exit Insert Mode:** Replace existing characters with characters entered.
<ESC>V	**Wrap at End of Line:** Automatically scroll cursor to beginning of next line when end of line reached.
<ESC>W	**Drop Characters at End of Line:** Ignore characters entered after end of line reached.

A.2 16-bit Output Character Set

When SMODE bit 2 is 1, the Console Resource Manager accepts 16-bit characters output with the WRITE SVC. Table A-2 defines the 16-bit output character set.

Table A-2. Output 16-bit Character Set

Range	Description
00xxH	Same as 8-bit; each character takes one character position in FRAME. Characters in the range 80H-FFH are defined on a per country basis.
01xxH - 0FxxH	Alternate character sets provided by the system implementer; each character takes one character position where the low byte is stored in the Character Plane and the low nibble of the high byte is stored in the low nibble of Extension Plane.
1xxxH	Non-visible characters (take no space).
2xxxH	Editing characters functionally equivalent to the VT-52 ESC sequences defined above: 2040 Enter insert character mode 2041 Cursor up 2042 Cursor down 2043 Cursor right 2044 Cursor left 2045 Clear display 2048 Cursor home 2049 Reverse index 204A Erase to end of display 204B Erase to end of line

Table A-2. (Continued)

Range	Description
	204C Insert blank line
	204D Delete line
	204E Delete character
	204F Exit insert character mode
	2064 Erase beginning of display
	2065 Enable cursor
	2066 Disable cursor
	206A Save cursor position
	206B Restore cursor position
	206C Erase entire line
	206F Erase beginning of line
	2070 Enter reverse video mode
	2071 Exit reverse video mode
	2072 Enter intensify mode
	2073 Enter blink mode
	2074 Exit blink mode
	2075 Exit intensify mode
	2076 Wrap at end of line
	2077 Discard at end of line
3xxxH	Set cursor to row xxx (0 origin)
4xxxH	Set cursor to column xxx (0 origin)
51xxH	Set background color to xx (see <ESC>c above)
52xxH - 7xxxH	Non-visible characters (take no space)
8000H - FCFCH	16-bit language; each character takes two character positions on FRAME (the corresponding Extension Plane bytes are modified to indicate byte order).

A.3 16-bit Input Character Set

When the CONSOLE table's KMODE bit 4 is 1, the Console Resource Manager accepts 16-bit characters input with the READ SVC. In a 16-bit character, the low byte contains the ASCII character code. The high byte is used as shown in Figure A-1. Table A-3 defines the 16-bit character set.

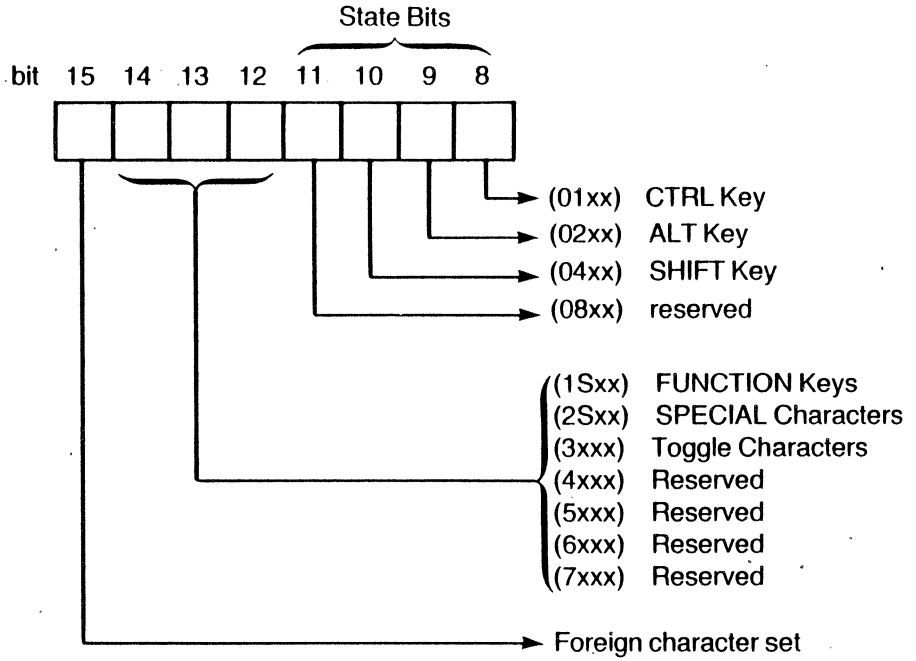


Figure A-1. High Byte Bit Usage of 16-bit Input Character

Table A-3 lists the 16-bit characters. The "S" in the table represents the value of CTRL, ALT and SHIFT state bits 8, 9, and 10. If these keys are depressed when another key is pressed, the corresponding bits come on. If the ASCII standard includes this character, the standard ASCII character is generated instead of the state value.

Table A-3. 16-bit Input Character Set

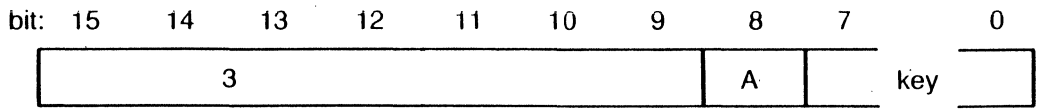
Range	Description	
0000 - 00FFH	ASCII character set	
1SxxH	Function keys	
2SxxH	Special keys defined as follows:	
	2S00	HELP
	2S01	WINDOW
	2S02	NEXT
	2S03	PREVIOUS
	2S04	PRINT SCREEN
	2S05	BREAK
	2S06	REDRAW (screen)
	2S07	BEGIN
	2S08	END
	2S09	INSERT
	2S0A	DELETE
	2S0B	SYS REQ
	2S10	Cursor up
	2S11	Cursor down
	2S12	Cursor left
	2S13	Cursor right
	2S14	Page up
	2S15	Page down
	2S16	Page left
	2S17	Page right
	2S18	Home
	2S19	Reverse tab

Table A-3. (Continued)

Range	Description
2S30	Numeric keypad 0
2S31	Numeric keypad 1
2S32	Numeric keypad 2
2S33	Numeric keypad 3
2S34	Numeric keypad 4
2S35	Numeric keypad 5
2S36	Numeric keypad 6
2S37	Numeric keypad 7
2S38	Numeric keypad 8
2S39	Numeric keypad 9
2S3A	Numeric keypad A
2S3B	Numeric keypad B
2S3C	Numeric keypad C
2S3D	Numeric keypad D
2S3E	Numeric keypad E
2S3F	Numeric keypad F
2S40	Numeric keypad ENTER
2S41	Numeric keypad COMMA
2S42	Numeric keypad MINUS
2S43	Numeric keypad PERIOD
2S44	Numeric keypad PLUS
2S45	Numeric keypad DIVIDE
2S46	Numeric keypad MULTIPLY
2S47	Numeric keypad EQUAL

Table A-3. (Continued)

Range	Description
3xxxH	Toggle character where xxx defines a toggle key as follows:



A - Action 0 - OFF
1 - ON

key
0 - Caps Lock
1 - Shift Lock
2 - Scroll Lock
3 - Num Lock
10 - Right Shift
11 - Left Shift
12 - Insert
13 - Control
14 - Alternate

When the user presses and releases keys 0 - 3 a single character is sent. For keys 10 - 14, a character is sent when the key is pressed and another is sent when it is released.

Toggle characters are only available if the hardware supports them.

Table A-3. (Continued)

Range	Description
4xxxH - 7xxxH	Reserved
8xxxH - FCxxH	15-bit Foreign language character sets including KANJI.

End of Appendix A

System Return and Error Codes

All FlexOS SVCs return 32 bit values. A negative number--the high order bit is set--indicates an error occurred. The remainder of the value is allocated as shown in Figure B-1.

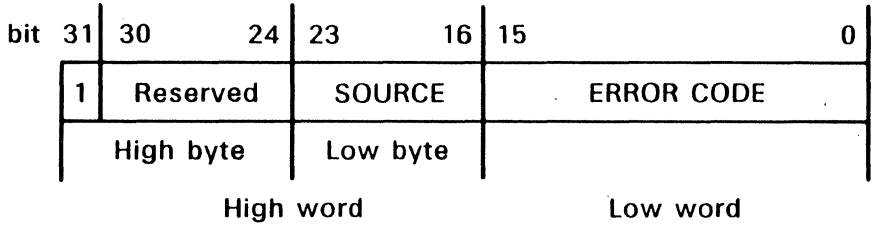


Figure B-1. Error Code Conventions

In the high order word, only the low byte is significant; the high byte is reserved. The low byte indicates the source of the error as indicated in Table B-1. By convention, operating system resource managers and modules have a zero-value in the low order nibble.

Table B-1. Error Source Codes--High Order Word

Value	Source
00H	Kernel or Supervisor
10H	Pipe Resource Manager
20H	Disk Resource Manager
21H - 2FH	Disk Drivers
30H	Console Resource Manager
31H - 3FH	Console Drivers
40H	Command/Load
50H	OEM Extension Resource Manager
51H - 5FH	OEM Extension Drivers
60H	Network Resource Manager
61H - 6FH	Network Drivers
70H	Miscellaneous Resource Manager
71H - 7FH	Miscellaneous Drivers
81H - FEH	Special Drivers

The low order word indicates the error condition. The codes are assigned in ranges of values again to indicate the source. Table B-2 lists the ranges and their corresponding source.

Table B-2. Low-order Word Error Code Ranges

Error Code Range	Source
0000 - 3FFF	Drivers
4000 - 407F	Errors Common to All Resource Managers
4080 - 40FF	Supervisor
4100 - 417F	Memory
4180 - 41FF	Kernel
4200 - 427F	Pipe and Miscellaneous Resource Managers
4280 - 42FF	Console System
4300 - 437F	File System
4400 - FFFF	Reserved

For the source of one of the common error codes, see the low byte in the high order word. The remaining tables in this appendix list define the error messages by their source. No error codes are currently associated with the Pipe, Console and Miscellaneous Resource Managers.

Table B-3. Driver Error Codes

Mnemonic	Code	Description
E_WPROT	0x00	Write protect violation
E_UNITNO	0x01	Illegal unit number
E_READY	0x02	Drive not ready
E_INVCMD	0x03	Invalid command issued
E_CRC	0x04	CRC error on I/O
E_BADPB	0x05	Bad parameter block
E_SEEK	0x06	Seek operation failed
E_UNKNOWNMEDIA	0x07	Unknown media present
E_SEC_NOTFOUND	0x08	Requested sector not found
E_DKATTACH	0x09	Attachment did not respond
E_WRITEFAULT	0x0A	Write fault
E_READFAULT	0x0B	Read fault
E_GENERAL	0x0C	General failure
E_RES1	0x0D	Reserved
E_RES2	0x0E	Reserved
E_RES3	0x0F	Reserved

Table B-4. Error Codes Shared by Resource Managers

Mnemonic	Code	Description
E_SUCCESS	0x0L	No Error
E_ACCESS	0x4001	Cannot access file--ownership differences
E_CANCEL	0x4002	Event Cancelled
E_EOF	0x4003	End of File
E_EXISTS	0x4004	File (CREATE) or device (INSTALL) exists
E_DEVICE	0x4005	Device does not match; for RENAME, on different devices
E_DEVLOCK	0x4006	Device is LOCKed
E_FILENUM	0x4007	Bad File Number
E_FUNCNUM	0x4008	Bad function number
E_IMPLEMENT	0x4009	Function not implemented
E_INFOTYPE	0x400A	Illegal Infotype for this file
E_INIT	0x400B	Error on driver initialization
E_CONFLICT	0x400C	Cannot access file due to current usage; for DELETE on open file or directory with files; for INSTALL, attempted replacement of driver in use
E_MEMORY	0x400D	Not enough memory available
E_MISMATCH	0x400E	Function mismatch--file does not support attempted function; for INSTALL, mismatch of subdrive type
E_NAME	0x400F	Illegal file name specified
E_NO_FILE	0x4010	File not found; for CREATE, device or directory does not exist
E_PARM	0x4011	Illegal parameter specified; for EXCEPTION, an illegal number
E_RECSIZE	0x4012	Record Size does not match request
E_SUBDEV	0x4013	INSTALL only: Sub-drive required
E_FLAG	0x4014	Bad Flag Number
E_EMEMACCESS	0x4015	Non-existent memory

Table B-4. (Continued)

Mnemonic	Code	Description
E_BOUNDS	0x4016	Memory bound error
E_EINSTRUCT	0x4017	Illegal instruction
E_EDIV0	0x4018	Divide by zero
E_EBOUNDEX	0x4019	Bound exception
E_OVERFLOW	0x401A	Overflow exception
E_PRIV	0x401B	Privilege violation
E_ETRACE	0x401C	Trace
E_EBREAK	0x401D	Breakpoint
E_EFLOAT	0x401E	Floating point exception
E_ESTACK	0x401F	Stack fault
E_EGENERAL	0x4020	General Exception

Table B-5. Supervisor and Memory Error Codes

Mnemonic	Code	Description
E_ASYNC	0x4080	Function does not allow asynchronous I/O
E_LOAD	0x4082	Bad load format
E_LOOP	0x4083	Infinite recursion (99 times) on prefix substitution; for INSTALL, subdrive type mismatch
E_FULL	0x4084	File number table full
E_DEFINE	0x4085	DEFINE only: illegal name
E_UNIT	0x4086	Too many driver units
E_UNWANTED	0x4087	Driver does not need subdriver
E_DVRTYPE	0x4088	Driver returns bad driver type
E_LSTACK	0x4089	Stack not defined in load header
Memory Error Codes		
E_POOL	0x4100	Out of memory pool
E_BADADD	0x4101	Specified bad address to free

Table B-6. Kernel Error Codes

Mnemonic	Code	Description
E_OVERRUN	0x4180	Flag already set
E_FORCE	0x4181	Return code of aborted process
E_PDNAME	0x4182	Process ID not found on abort
E_PROCINFO	0x4183	COMMAND only: no procinfo specified
E_LOADTYPE	0x4184	COMMAND only: invalid loadtype
E_ADDRESS	0x4185	CONTROL only: invalid memory access
E_EMASK	0x4186	Invalid event mask
E_COMPLETE	0x4187	Event has not completed
E_STRL	0x4188	Required SRTL could not be found
E_ABORT	0x4189	Process cannot be terminated
E_CTRLC	0x418A	Process aborted by Ctrl-C
E_CONTROL	0x418B	Slave process running
E_SWIRET	0x418C	Not in SWI context
E_UNDERRUN	0x418D	Flag already pending
E_SPACE	0x4300	Insufficient space on disk or in directory
E_MEDIACHANGE	0x4301	Media change occured
E_MEDCHGERR	0x4302	Detected media change after a write
E_PATH	0x4303	Bad path
E_DEVCONFLICT	0x4304	Devices locked exclusively
E_RANGE	0x4305	Address out of range
E_READONLY	0x4306	RENAME or DELETE on R/O file
E_DIRNOTEMPTY	0x4307	DELETE of non-empty directory
E_BADOFFSET	0x4308	Bad offset in read, write or seek
E_CORRUPT	0x4309	Corrupted FAT
E_PENDLK	0x430A	Cannot unlock a pending lock
E_RAWMEDIA	0x430B	Not FlexOS media
E_FILECLOSED	0x430C	File closed before asynchronous lock could be completed
E_LOCK	0x430D	Lock access conflict

Utility return codes follow the same format of operating system error return codes, as illustrated in Figure B-1, with the following exceptions:

- Utility return codes are positive numbers (LONGS) because the high order bit (31) is always zero.
- When possible, you should use the error codes listed in Table B-7 in the error code field (bits 0-15).
- You can designate given modules within an application in the source field (bits 16-23).

To return errors generated within your application, OR the source field (module) with the error code field. For example, to indicate that an application has detected a parameter error, use:

```
return( UR_SOURCE | UR_PARM );
```

Do not OR a source field value with UR_SUCCESS, which is a LONG of zeroes.

Table B-7. Utility Return Codes

Mnemonic	Code	Description
UR_SOURCE	(LONG)0	Utility return
UR_SUCCESS	(LONG)0	Success
URPARM	0x0001	Parameter error
UR_CONFLICT	0x0002	Contention conflict
UR_UTERM	0x0003	Terminated by user
UR_FORMAT	0x0004	Data structure format error
INTERNAL	0x0005	Internal utility error
UR_UR_DOSERR	0x0006	PC DOS error

End of Appendix B

Country Codes

All FlexOS console drivers indicate the country code that is currently supported. These country codes are used by applications to distinquish character sets, accounting practices, currency symbols presentation, date presentation and many other country or region dependent practices.

Code	Country or Region
10	Afghanistan
20	Albania
30	Algeria
40	Andorra
50	Angola
60	Antigua
70	Argentina
80	Austria
90	Australia
100	Bahama Islands
110	Bahrein
120	Bangladesh
130	Barbados
140	Belgium
150	Bermuda Islands
160	Bhutan
170	Bolivia
180	Botswana
190	Brazil
200	British Honduras
210	Brunei
220	Bulgaria
230	Burma
240	Burundi
250	Cameroun
260	Canada
270	Central African Republic

Code	Country or Region
280	Ceylon
290	Chad
300	Chile
310	China
320	Colombia
330	Congo
340	Costa Rica
350	Cuba
360	Cyprus
370	Czechoslovakia
380	Dahomey
390	Denmark
400	Dominica
410	Dominican Republic
420	East Germany
430	Ecuador
440	Egypt
450	El Salvador
460	Equatorial Guinea
470	Ethiopia
480	Fiji
490	Finland
500	France
510	French Guiana
520	French Somaliland
530	Gabon
540	Gambia
550	Ghana
560	Greece
570	Greenland
580	Grenada
590	Guadeloupe
600	Guatemala
610	Guinea
620	Guyana

Code	Country or Region
630	Haiti
640	Honduras
650	Hong Kong
660	Hungary
670	Iceland
680	Indonesia
690	India
700	Iran
710	Iraq
720	Ireland
730	Israel
740	Italy
750	Ivory Coast
760	Jamaica
770	Japan
780	Jordan
790	Kenya
800	Khmer Republic
810	Kuwait
820	Laos
830	Lebanon
840	Lesotho
850	Liberia
860	Libya
870	Liechtenstein
880	Luxembourg
890	Malagasy Republic
900	Malaysia
910	Malawi
920	Malaysia
930	Maldive Islands
940	Mali
950	Malta
960	Mauritania

Code	Country or Region
970	Mauritius
980	Mexico
990	Moldavian SSR
1000	Monaco
1010	Mongolia
1020	Morocco
1030	Mozambique
1040	Nepal
1050	Netherlands
1060	New Caledonia
1070	New Guinea
1080	New Hebrides
1090	New Zealand
1100	Niger
1110	Nigeria
1120	Nicaragua
1130	North Korea
1140	Norway
1150	Oman
1160	Pacific Islands
1170	Pakistan
1180	Panama
1190	Papua
1200	Paraguay
1210	People's Democratic Republic of Yemen
1220	Peru
1230	Philippines
1240	Poland
1250	Portugal
1260	Portuguese Guinea
1270	Puerto Rico
1280	Qatar
1290	Rhodesia
1300	Rumania

Code	Country or Region
1310	Rwanda
1320	St. Kitts-Nevis-Anguilla
1330	St. Lucia
1340	St. Vincent
1350	San Marino
1360	Saudi Arabia
1370	Senegal
1380	Sierra Leone
1390	Sikkim
1400	Singapore
1410	Somalia
1420	South Africa
1430	South Korea
1440	South-West Africa
1450	Spanish Sahara
1460	Spain
1470	Sudan
1480	Surinan
1490	Swaziland
1500	Sweden
1510	Switzerland
1520	Syria
1530	Tahiti
1540	Taiwan
1550	Tanzania
1560	Thailand
1570	Tibet
1580	Timor
1590	Togo
1600	Trinidad and Tobago
1610	Tunisia
1620	Turkey
1630	Uganda

Code	Country or Region
1640	Union of Soviet Socialist Republics
1650	United Arab Emirates
1660	United Kingdom
1670	United States of America
1680	Upper Volta
1690	Uruguay
1700	Vatican City
1710	Venezuela
1720	Vietnam
1730	West Germany
1740	Western Samoa
1750	Yemen Arab Republic
1760	Yugoslavia
1770	Zaire
1780	Zambia

End of Appendix C

A

B

C

E

F

G

S

T