GST Computer Systems Limited

# 68K/OS

# PROGRAMMER'S

# REFERENCE MANUAL

GST Computer Systems Limited

CONTENTS

1 **INTRODUCTION**
1.1 Purpose
1.2 Scope
1.3 Audience
1.4 Copyright
1.5 References

2 **SYSTEM OVERVIEW**
2.1 68K/OS Main Features
2.2 Asynchronous Components
2.3 Synchronous Call Components
2.4 Synchronous Trap Components
2.5 Applications Program Interface

3 **INPUT/OUTPUT SUBSYSTEM FUNCTIONS**
3.1 IOSS Interfaces
3.2 Standard Device Drivers
3.3 IOSS Channels
3.4 Device Independence and Redirectable I/O
3.5 Path Names
3.6 Path Name Defaults
3.7 Access Type
3.8 Access Mode
3.9 Calling IOSS Routines
3.10 Default String Functions
3.11 Open an IOSS Channel
3.12 Close an IOSS Channel
3.13 Procedure Handling Functions
3.14 File Delete and Rename Functions
3.15 Update Directory
3.16 Read Directory Information
3.17 Reading from IOSS Channels
3.18 Writing to IOSS Channels
3.19 File Positioning
3.20 Polling an Input Channel
3.21 Mounting and Dismounting Directories
3.22 Device Driver Special Function

4 OPERATING SYSTEM FUNCTIONS
4.1 Overview of OS Functions
4.2 Calling OS Routines
4.3 Program Manager Functions
4.4 Initial Program State
4.5 Starting a New Child Program
4.6 Determine Program Status
4.7 Wait for a Program to Finish
4.8 Force Program Termination
4.9 Memory Manager Functions
4.10 Allocate extra RAM to a Program
4.11 Change Ownership Information
4.12 Release Memory by Ownership Information
4.13 Release Memory by Address Range
4.14 The Menu Manager
4.15 Menu Data Structures
4.16 Display Fixed Menu Data
4.17 Read User Input to Menu
4.18 Read a Variable Field
4.19 Update a Variable Field
4.20 Timing Services
4.21 Passive Delay
4.22 Read Binary Time and Date
4.23 Set Binary Time and Date
4.24 Heap Allocation
4.25 Allocate a Heap Record
4.26 Deallocate a Heap Record
4.27 Determine the Free Stack/Heap Space
4.28 User Trap Handler

5 DISPLAY FILE MANAGER
5.1 Outline Description
5.2 Partitions
5.3 Virtual Screens
5.4 Windows
5.5 Display Files
5.6 Extended Display Files
5.7 Cursor, Action Pointer and Markers
5.8 Console Display File Interface and IOSS
5.9 Display File Binary Commands
5.10 Single Line Menu
5.11 Calling DFM Routines
5.12 Initialisation Routines
5.13 Termination Routines
5.14 Display File Control Routines
5.15 Space Allocation Routines
5.16 Line Manipulation Routines
5.17 Character Manipulation Routines
5.18 String Manipulation Routines
5.19 Cursor Routines
5.20 Marker Position Routines
5.21 Update Single Line Menu
5.22 Install User Hook Routine

6 **GRAPHICS ROUTINES**
6.1 General Description
6.2 Coordinate System
6.3 Colour Definitions and Stipple Patterns
6.4 Aspect Ratio
6.5 Calling Graphics Routines
6.6 Graphics Figures

7 **CREATING PROGRAMS AND PROCEDURES**
7.1 Overview
7.2 Position Independence
7.3 Reentrant Code
7.4 Procedure Header Block
7.5 Program Memory Requirements
7.6 Program Memory Layout
7.7 Data Area Pointers
7.8 Special Conditions at Start of Program
7.9 Program and Procedure Exit
7.10 Passing Status Parameters

8 **SYSTEM DATA STRUCTURES**
8.1 Scope
8.2 Notation
8.3 Directory Entry Buffer
8.4 Directory Information Buffer
8.5 Menu Fixed Data Structure
8.6 Procedure Entry Control Block
8.7 Program List Element
8.8 Standard Parameter String
8.9 Standard Text String

## APPENDICES

A I/O SUB-SYSTEM CALLS
A.1 IOSS Register Conventions
A.2 Detailed IOSS Function Specification

B OPERATING SYSTEM CALLS
B.1 Detailed OS Function Specification

C DISPLAY FILE MANAGER CALLS
C.1 Detailed DFM Function Specification

D GRAPHICS ROUTINES
D.1 Detailed Graphics Function Specification

E STATUS CODES
E.1 Format
E.2 Alphabetical List of Status Codes

F CHARACTER CODES
F.1 General
F.2 Changes from Standard US ASCII
F.3 QL ASCII Decode Table
F.4 Summary of System Mode Commands
F.5 Summary of User Mode Commands
F.6 Display File Manager Commands
F.7 DFM and Screen Driver Commands

G DEVICE DRIVERS
G.1 Overview
G.2 Keyboard Driver
G.3 Screen Driver
G.4 Microdrive Filing System
G.5 RS232 Output Driver
G.6 RS232 Input Driver
G.7 ROM Driver

# SECTION 1:

# INTRODUCTION

# 1 INTRODUCTION

## 1.1 Purpose

This manual describes the 68K/OS operating system for the Sinclair QL and other personal computers, intelligent terminals and workstations based on the the Motorola M68000 series processors. Sufficient details of system call interfaces and data structures are provided for the production of advanced assembler level applications software.

## 1.2 Scope

This edition of the 68K/OS Programmer's Reference Manual defines both the portable and implementation dependent areas of the system, namely:

(a) Chapter 6 and appendices D, F and G refer to facilities available on the Sinclair QL that may not be available or may have different interfaces on later implementations of the operating system.

(b) The remaining material defines the interfaces to the portable sections of the operating system that should remain unchanged on later implementations of the operating system.

This manual provides details of 68K/OS interfaces and internal data structures necessary for production of applications software. Details of 68000 architecture and instruction syntax are available from Motorola, details of the 68K/ASM assembler and a 68000 programming primer are available from GST (see paragraph 1.5).

Systems programming interfaces and data structures are provided in a separate manual (see paragraph 1.5). Certain systems programming facilities provided by the operating system require that the programmer has detailed documentation of the QL hardware. GST do not supply this documentation and cannot guarantee that such documentation will be made available by Sinclair Research Limited or any third party.

## 1.3 Audience

68K/OS is a small but advanced operating system aimed at the following market sectors:

(a) OEM suppliers of 68000-based terminals and workstations,

(b) Independent 68000 software developers,

(c) Computer science students and advanced home users with 68000-based personal computers.

This manual therefore assumes that the reader has a reasonable working knowledge of programming, the 68000 processor and operating system theory.

### 1.4 Copyright



### 1.5 References

| | |
|---|---|
| 8290.6 GST 68 | 68K/ASM Assembler Reference Manual |
| 9992.1 GST 54 | 68K/OS Systems Programmer's Reference Manual |
| Motorola M68000UM | M68000 Programmer's Reference Manual |
| Addison-Wesley | Programming the M68000 (Tim King & Brian Knight) |

# SECTION 2:

# SYSTEM OVERVIEW

## 2 SYSTEM OVERVIEW

### 2.1 68K/OS Main Features

68K/OS is a single-user multitasked system using conventional operating system software techniques typical of those found on many minicomputer systems, with the addition of sophisticated screen window management software. The main features of the system are as follows:

(a) **Operating System:** 68K/OS is a true operating system in the sense that it has both asynchronous and synchronous components, unlike a monitor system (such as CP/M) with only synchronous components.

(b) **Multitasked:** the system shares its time and memory resources between several 'concurrent' programs, with a program scheduler that arbitrates between them.

(c) **Priority Scheduling:** the 68K/OS scheduler uses a priority-based algorithm to determine which program to invoke in response to a real-time event. The highest priority 'ready' program is invoked.

(d) **Event Driven:** the scheduler is invoked by a real-time event, which is either a return from interrupt or a system trap or call.

(e) **Programs and Reentrant Procedures:** a 68K/OS program consists of a program control block (PCB), a data area for its stack and heap, and at least one procedure. Procedures must be both reentrant and position independent. Only one copy of a procedure will ever be loaded at any given time, even though it may be shared by several concurrent programs.

(f) **Semaphore Communication:** concurrent programs communicate using general semaphores. This is the only standard method of program communication provided, though semaphore control is transparent when using piped I/O.

(g) **Device Independent I/O:** with the exception of screen window updates, all input/output of applications software within 68K/OS uses the device independent I/O sub-system (IOSS). The IOSS provides a standard calling interface to I/O functions and allows complete run-time I/O redirection.

(h) **Screen Window Control:** the display file manager (DFM) supports 'simultaneous' screen updates by concurrent programs in variable sized screen partitions, and allows programs to divide their screen partitions into windows dynamically. Each window is associated with a display file whose text is maintained by DFM independently of the window and can be scrolled through the window both vertically and horizontally.

As a real-time multitasked system, 68K/OS strongly resembles operating systems such as RSX or UNIX and provides a powerful subset of the features to be found on these much larger systems. In addition, 68K/OS provides unique screen window handling facilities, yet the entire operating system will fit into 32Kb of ROM or EPROM.

## 2.2 Asynchronous Components

The operating system contains software processes that run (either entirely or partly) asynchronously with respect to applications programs. Two of these have special status and execute in supervisor mode with interrupts disabled:

(a) **Interrupt Handler:** a single interrupt routine is responsible for handling all system interrupts and vectoring (by software) to the individual service routines. Facilities are provided for systems programmers to add extra interrupt service routines to the system. The interrupt handler runs with all interrupts disabled and in supervisor mode.

(b) **Scheduler:** the scheduler is responsible for maintaining the queue of PCBs and, whenever a real-time event occurs, for finding and invoking the highest priority ready program. The scheduler runs with all interrupts disabled and in supervisor mode.

The remaining asynchronous system processes are all programs controlled by the scheduler. These run in user mode with interrupts enabled and have the same status as user programs:

(c) **Null Program:** this program has the lowest possible priority and is responsible for soaking up all spare CPU cycles when no other program is ready.

(d) **Disk Program:** this is responsible for maintaining an ordered list of memory block addresses to enable intelligent seek optimisation to be performed on a random (or pseudo-random) access device such as a disk or microdrive.

(e) **Interrupt Poll Control Program:** the IPC program is invoked by a 50/60Hz clock interrupt and calls a number of hardware poll routines for devices that are not driven by interrupt. Facilities are provided for systems programmers to add extra hardware poll routines to the system.

(f) **Undertaker Program:** this program is invoked whenever a program terminates (either voluntarily or as a result of an error trap or a program kill system call) and is responsible for releasing all system resources owned by the terminated program, these being open channels, system memory, screen partition and, recursively, those resources owned by any child programs.

Only systems programmers will require the asynchronous facilities provided in the interrupt handler and IPC program. Applications programmers should regard the whole of paragraph 2.2 as containing background information only.

## 2.3 Synchronous Call Components

The majority of 68K/OS functions are provided by the synchronous components of the operating system. These are invoked by applications software through subroutine calls via four sets of vectored entry points. This software is executed in user mode with interrupts enabled and is logically an extension of the calling applications program, and subject to the usual rules of priority scheduling and program status.

Subroutine call entry points are provided for the system components defined below:

(a) **I/O Sub-System:** this provides a device independent input/output calling mechanism for data transfer to and from files and devices. Facilities are provided to load user defined IOSS device drivers to enable applications software to access plug-in devices via the standard IOSS calling mechanism.

(b) **Program Manager:** this provides a standard method to start applications programs, determine their status, wait for their completion or to force their termination.

(c) **Memory Manager:** this provides functions to enable programs to obtain and release extra RAM memory in 1Kb units.

(d) **Display File Manager:** this software provides a comprehensive set of functions to create, update and delete information in screen windows and to manipulate the data in the display files associated with them.

(e) **Menu Manager:** this software enables the display of complex menus and forms in a screen window and will handle data entry and data capture for a complete form without the need for intervention by the applications software.

(f) **Timing Functions:** facilities are provided to invoke both passive and active program delays and to read or set the internal time-of-day clock.

(g) **Heap Allocation:** routines are provided to grab and release space from applications program heap storage and to determine the available heap space.

(h) **Graphics Primitives:** software is provided to draw points, lines, blocks, simple figures and conic sections in any screen window, with automatic clipping at window edges.

The graphics functions provided under 68K/OS for the Sinclair QL are specific to the QL hardware and are not guaranteed to be provided in the same format (or at all) on later hardware implementations.

## 2.4 Synchronous Trap Components

System trap entry points are vectored into the synchronous regions of the scheduler. These are executed in supervisor mode with interrupts disabled and, following completion of the requested function, may cause a system reschedule. Traps are are provided for the following systems programming functions:

(a) **Semaphore Control:** general functions to signal, poll and wait on semaphores permit low-level program communication.

(b) **Program Status Control:** functions that directly alter the program status of either the caller or a target program are provided.

TRAP 0 is provided to terminate an applications program. TRAPs 1 to 3 are reserved for the operating system and are fully defined in the Systems Programmer's Reference Manual.

A function is provided to enable an applications program to redefine (for its own exclusive use) those trap vectors that are not reserved for the system. These may be vectored to user defined trap handling routines, one of which is entered in system mode, the remainder in user mode.

## 2.5 Applications Program Interface

The normal interface from applications software to 68K/OS is via four general call vectors, entered with a function code in D0 and returning a status code in D0:

(a) **IOSS Vector:** all input/output sub-system calls.

(b) **DFM Vector:** all display file manager calls.

(c) **OS Vector:** all other hardware independent system calls including program manager, memory manager and menu manager functions, plus timing and heap allocation routines.

(d) **SP Vector:** all hardware dependent system calls including graphics primitive routines.

These functions are sufficient for all normal applications software. Systems programs will require details of 68K/OS internal facilities such as system traps, data structures and device driver installation. These can be found in the System Programmer's Reference Manual.

# SECTION 3:

# INPUT/OUTPUT

# SUBSYSTEM FUNCTIONS

# 3 INPUT/OUTPUT SUSBSYSTEM FUNCTIONS

## 3.1 IOSS Interfaces

The IOSS has two major interfaces: that which it presents to calling programs and that which it presents to device drivers.

A user program calls the IOSS as a subroutine via the entrypoint IOENTRY. The IOSS decides which device driver should be used to implement the function requested and calls the relevant driver as a subroutine. All these calls take place synchronously under the control of the calling user program and any memory which IOSS requires to perform the requested function is allocated in the user program's heap. IOSS is responsible for freeing any user program heap it allocates.

Some device drivers may need to operate to some extent asynchronously with respect to the user program in order to operate synchronously with some hardware device. In this case the driver will consist of a separate concurrently running program and/or interrupt routine in addition to the subroutines called directly from IOSS, communication between components of the driver being achieved with semaphores. This operation is transparent to the user program, which remains blissfully ignorant of the relative complexities of various IOSS drivers.

## 3.2 Standard Device Drivers

Standard IOSS device drivers are provided for:

(a) Keyboard (KEY:)

(b) Screen (SCREEN:)

(c) Microdrive (MD:)

(d) Pipe (PIPE:)

(e) Serial Transmit (TX1: and TX2:)

(f) Serial Receive (RX1: and RX2:)

(g) ROM Directory (ROM:)

Note that use of the SCREEN: device is a simple method of screen output that takes default paths through DFM and requires no explicit DFM calls from the applications software. Note also that reading lines from KEY: has the usual line editing screen interaction that one would expect from a console device.

## 3.3 IOSS Channels

All IOSS input/output takes place through channels which are assigned and controlled by the IOSS. A channel is an input/output route attached by IOSS to a file or device and owned by a specific program.

## 3.4 Device Independence and Redirectable I/O

Because the calling interfaces to IOSS routines are identical for all devices, IOSS is defined to be device independent. Applications programs can usually perform channel I/O without needing to know whether the channel is attached to a microdrive file, a pipe, a serial communications line or an IOSS compatible add-on device.

Applications software can be written to enable the actual I/O devices that will be used by the program to be specified by the user when the program is run, providing redirectable I/O.

## 3.5 Path Names

All sources and destinations of IOSS channels (devices and files) are identified by a path name which has the general format:

DEVICE:DIRECTORY/FILENAME.EXTENSION

where there may be any number of directory components, each followed by a slash, and provided that the total length of the pathname does not exceed 44 characters. Each component must be between one and eight characters long, and may be a mixture of numeric and alphabetic characters of either case, the case being non-significant.

IOSS devices differ in their requirements for path name components, the full path names for each standard device are defined below:

(a) KEY:

(b) SCREEN:

(c) MD:DIRECTORY/FILENAME.EXTENSION

(d) PIPE:FILENAME.EXTENSION

(e) TX1:

(f) TX2:

(g) RX1:

(h) RX2:

(i) ROM:FILENAME.EXTENSION

Note that where a FILENAME component is specified the EXTENSION component is optional.

The IOSS performs syntax analysis of a path name and extracts the device name component to decide which device driver to call. It is legal (though rot necessarily sensible) to append filename components to a pathname for a device that is not file structured (TX1:FILENAME), this will be ignored by IOSS and all standard IOSS drivers.

## 3.6 Path Name Defaults

A system of path name defaults is provided by IOSS to supply any path name components not specified by the program. Four user-specified default strings are maintained which are used by IOSS to complete partial pathnames:

(a) Default program device

(b) Default program directory prefix

(c) Default data device

(d) Default data directory prefix

where the device name is null or a device name component ending in a colon, and the directory prefix is null or one or more directory components each ending in a slash.

Each IOSS function that requires a pathname has a parameter indicating whether program or data default strings should be used. The program supplied pathname is examined and any missing components are inserted in the pathname by IOSS using the program or data strings as follows:

(a) If the path name does not contain either a colon or a slash then the relevant default directory prefix is added.

(b) If the pathname does not contain a colon then the relevant default device is added.

Note that the two different sets of default strings are provided so that programs can be loaded from one device and data files can be accessed on another device with no device or directory names needing to be specified by the user.

Note also that these rules apply to path names as supplied to IOSS and that particular applications programs may apply additional rules, such as appending standard extension names to input filenames to construct default output filenames.

## 3.7 Access Type

Data access type refers to read/write access permission at four levels:

(a) **Device:** each device has a fixed access type which usually refers to some physical restriction. For example, you cannot read from the screen or write to the keyboard.

(b) **Directory:** each directory has an access type which is the same as or more restrictive than the device access type. For example, a write-protected microdrive imposes directory level restrictions on the access type.

(c) **File:** each file has an access type which is the same as or more restrictive than the directory level access type. Note that a file may have write permission regardless of the fact that the microdrive it resides on has been write-protected, in this case the access type of the directory overrides the file access type.

(d) **Channel:** each channel has an access type which is defined when the channel is opened which must be the same as or more restictive than the combined access type of the components of the path name. Thus any attempt to open a channel to write to the keyboard will fail immediately (on grounds of incompatible access type) before any write attempts are made.

IOSS will always choose the most restrictive access type of the four levels when deciding whether a data transfer request is legal. For example, a channel may be opened for reading only on a file which could otherwise be written to: read calls will be permitted but any attempts to write to that channel are failed.

## 3.8 Access Mode

The access mode of data is either random or sequential and is defined at two levels:

(a) **Device:** each device has a fixed access mode which usually refers to some physical restriction. For example, it is not possible to perform random access on a serial line.

(b) **Channel:** when a channel is opened its required access mode is specified. If random access is requested and the device has random access permission then the channel will be given random access permission.

IOSS will always choose the more restrictive access mode of the two levels when deciding if a data transfer request is legal. For example, if a channel is opened with sequential access only to a file on a random access device, then all random access calls on that channel will be failed.

### 3.9 Calling IOSS Routines

IOSS routines are called by applications software via a single entry point with the function code in D0:

```
MOVEQ    #IOFUNC,D0
JSR      IOENTRY
```

On return D0 contains a status code.

Descriptions of each IOSS routine follow below, and precise details of each IOSS call are given in Appendix A.

### 3.10 Default String Functions

These functions are used to set and read the current program and data default strings used by the calling program.

When a program is created it inherits its parent program's default strings. **IOSETDEF** is used to change either the program or data default strings, and requires a string parameter of the form DEVICE:DIRECTORY/ where both components are optional, and the directory component may be repeated. A null string is valid and has no effect on the current program or data default strings.

**IOGETDEV** will return the current program or data default device.

**IOGETPRE** will return the current program or data default directory prefix.

### 3.11 Open an IOSS Channel

The **IOOPEN** routine is the means of creating IOSS channels through which input and output operations can be performed. An IOSS channel open operation will follow the general pattern outlined below:

(a) The path name is extended using the program or data default strings as necessary and the syntax of the resulting path name is checked for plausibility.

(b) The device name is extracted from the path name and the requested access type and mode are compared with those legal for the device.

IOSS calls the device driver to perform device specific checks. Where this is a filing system, checks are made for the existence of the specified file, the access types of the directory and file and whether the user has specified double buffering for file I/O. Checks for incompatible multiple uses of devices or files are usually generated by the device driver, but devices which can only be used by one channel at cnce are protected by IOSS itself.

If there are no status codes generated from any of these operations, IOSS will open a channel.

## 3.12 Close an IOSS Channel

The **IOCLOSE** call closes a channel. This can be a fairly lengthy process for some devices (such as an output microdrive file) but has no complications of interest to the user. After this call the channel number on which this channel was open has no further validity.

## 3.13 Procedure Handling Functions

The **IOLOAD** call loads position independent, reentrant procedures from a directory structured device such as a disk or microdrive or the ROM: device. Path name validation follows the method used in IOOPEN, using program or data default strings as required. If the procedure exists it may be placed in RAM depending on the following criteria:

(a) If the procedure is not already loaded and is held on a disk or microdrive, then it is read into RAM, the RAM address is returned to the caller, and a system procedure table entry is created.

(b) If the procedure is not already loaded and is held in the ROM: device, then the ROM address is supplied to the caller and a system procedure table entry is created.

(c) If the procedure is already loaded, then its address (ROM or RAM) is supplied to the caller and the use count field of the system procedure table entry is incremented.

When loading a procedure, two data structures are required:

(d) A procedure entry control block. This passed by IOSS to the device driver which places the procedure entrypoint into the control block. If the procedure cannot be loaded, IOSS will supply the address of a program termination routine instead, as a precaution against calling a non-existent procedure.

(e) A procedure list element. This is grabbed by IOSS from the calling program's heap and is chained to its PCB (enabling program termination software to unload procedures owned by a program).

Note that procedures are owned by programs and that this ownership can be shared. When its final owner is terminated the procedure is automatically unloaded by the system.

The **IODEFPRO** call defines an entrypoint specified by the caller as being a procedure. The calling program fills in the procedure entry control block before calling the routine, and the path name supplied must refer to a directory structured device to pass IOSS validation.

The **IOUNLOAD** call is used to indicate that a program no longer requires to use a procedure. The procedure has its procedure list element removed from its owner's PCB chain and its use count in the system procedure table is decremented. If the use count drops to zero, the memory that the procedure occupied is freed. (Note that the IOLOAD and IOUNLOAD calls can be used as an overlay mechanism).

### 3.14 File Delete and Rename Functions

The **IODELETE** call deletes the file specified by the given path name combined with program or data default strings as necessary. In order to carry out the delete function IOSS ensures:

(a) The device component of the full path name allows directory operations.

(b) The file defined by the path name exists.

(c) The access type for the device, directory and file components of the path name allow write access.

If these three conditions are satisfied, the file is deleted.

The **IORENAME** call changes the filename and/or the extension components of the file specified by the given path name combined with program or data default strings as necessary. IOSS will ensure that conditions (a) to (c) above are met for the existing file and that the following conditions are met for the new file:

(d) The device and directory components are the same as those specified for the existing file.

(e) The new file does not already exist.

If these five conditions are satisfied, the file is renamed.

### 3.15 Update Directory

The **IOPUTDIR** call enables a program to update the directory information for a given filename by supplying IOSS with a directory entry buffer. Only three fields can be changed by this call:

(a) File access type (read/write permission)

(b) Date and time last modified (this is set to the current date/time)

(c) User comment

Any other fields supplied in the directory entry buffer are ignored and the original values retained in the directory entry.

This call is designed to be used after obtaining the directory entry buffer from an IOGETDIR call and changing the relevant entries, however it is legal for the user program to construct its own directory entry buffer.

3.16 **Read Directory Information**

The **IOGETDIR** call allows a user program to read the directory entry for a given filename into a directory entry buffer and can be repeated to fetch each of the directory entries for a set of filenames that match a given pattern.

The path name used in this call may include **?** and ***** as wild card characters in the filename and extension components, as follows:

(a) A **?** in the body of a component matches any single alphanumeric character. For example, FI?E matches FILE and FIRE.

(b) Each **?** at the end of a component matches zero or one alphanumeric characters. For example, FILE?? matches FILE, FILE1 and FILE10.

(c) A * at the end of the component is equivalent to extending the component length to eight characters by appending ? characters. Thus FILE* is equivalent to FILE???? and matches FILE, FILE1, FILE10, FILE100 and FILE1000, for example.

Note that *.* will match any filename and extension combination.

The IOGETDIR call searches the directory indicated by the device and directory fields of the path name (extended as necessary by default strings in the usual manner) until it finds a match, in which case the directory entry buffer information is returned to the user.

The specified directory is searched from a starting position which depends on a magic number passed to the routine in D1. On the initial call to IOGETDIR for a given directory this number must be zero. On subsequent calls this parameter may either be zero (to rescan the directory from the beginning) or the magic number returned by the previous call of IOGETDIR (in which case the directory scan continues from where it left off). The effect of supplying any other magic number is undefined and likely to be unhelpful.

Directory entries are retrieved by IOGETDIR in no defined order. If the user program requires directory entries in any particular order then it must sort them itself.

The **IODIRINF** call returns information on a whole directory, whose pathname (without filename or extension fields) is supplied by the caller. The data is returned in a directory information buffer. This contains three fields:

(a) Maximum number of directory entries

(b) Maximum space available in Kb

(c) Current space available in Kb

Thus this command can be used to determine free space on disk or microdrive, and how many sort records will be required when sorting the directory entries.

## 3.17 Reading from IOSS Channels

Three IOSS calls are provided for reading data from channels previously opened succesfully for reading with IOOPEN. In all cases data is read into a user supplied buffer of a length assumed to be large enough to accommodate the requested number of bytes. If double-buffering was requested with IOOPEN (and this is supported by the device) then the system will perform read-ahead operations into system 'slave blocks' to improve performance, and if so this is transparent to the user program.

The **IOGETSEQ** call will attempt to read the defined number of bytes from the specified channel which must have been opened with read access type.

The **IOGETRAN** call is identical to IOGETSEQ except that a file position is provided by the calling program and IOSS also checks that the channel was opened with random access type.

The **IOGETLIN** call is identical to IOGETSEQ except when a newline character is detected during data transfer. In this case the transfer will stop and the actual number of bytes read (including the newline) will be returned to the user.

Note that an IOGETLIN call from a channel connected to the device KEY: has a special effect. All standard ASCII characters received from the keyboard will be reflected in the default SCREEN: window for the calling program (which will be created automatically if necessary). Keystroke reflection includes backspace, backspace-delete and delete line keystrokes which provide line editing functions internal to IOSS and with no user program intervention.

All the calls to read data may succeed only partially if end of file is reached. For this reason the actual number of bytes read is returned to the user along with a status code to say what happened. End of file is a device dependent condition, but in general, if there are N bytes left in a sequential input file and a read request for N bytes is made then the call will succeed, end of file status being returned on the next call.

When reading from files, the channel's file position pointer on entry to the read routine will be incremented by the number of bytes actually read to give a new position in the file. This ensures that subsequent read calls will advance through the file sequentially unless the file position is changed explicitly.

Note that it is legal to mix all three types of read from a single random access channel, though the user program must ensure sensible positioning of the file pointer to avoid silly answers.

3.18 **Writing to IOSS Channels**

Three IOSS calls are provided to write data to channels previously opened succesfully for writing with IOOPEN. In all cases data is written from a user supplied buffer assumed to contain the requested number of bytes. If double-buffering was requested with IOOPEN (and this is supported by the device) then the system will perform write-behind operations from system 'slave blocks' to improve performance, and this is transparent to the user program.

The **IOPUTSEQ** call will attempt to write the defined number of bytes to the specified channel which must have been opened with write access type.

The **IOPUTRAN** call is identical to IOPUTSEQ except that a file position is provided by the calling program and IOSS also checks that the channel was opened with random access mode.

The **IOPUTLIN** call is identical to IOPUTSEQ except when a newline character is detected during data transfer. In this case the transfer will stop, and the remainder of the buffer contents is ignored.

Note that that first IOPUTSEQ or IOPUTLIN call will start at file position zero and that if the file was opened for sequential output only, then it will have been truncated to zero length by IOOPEN.

When writing to files, the channel's file position pointer on entry to the write routine will be incremented by the number of bytes actually written to give a new position in the file. This ensures that subsequent write calls will advance through the file sequentially unless the file position is changed explicitly.

When writing sequentially, or when a random write would exceed the end of file position, the end of file pointer is set to point to a position one byte greater than the last byte written. If a random write starts at a position beyond end of file then the file is padded with nulls up to the start position.

Note that it is legal to mix all three types of write to a single random access channel, though the user program must ensure sensible positioning of the file pointer to avoid silly answers.

## 3.19 File Positioning

Every channel has a current position pointer (starting at position 0) which is the byte address in the file at which input and output takes place. This current position is moved automatically by the reading and writing routines described above. It can also be moved and interrogated directly by the user program using the routines described in this section.

Each file on disk or microdrive has a size in bytes equal to the position of the end of file pointer. Channels which are not attached to disk or microdrive files will not have sensible size data available.

The **IOSETPOS** call can be used to set the current file position pointer for a file (with random access mode only) as follows:

(a) If the requested file pointer is less than or equal to the end of file position when called, the file position is updated as requested.

(b) If the requested file pointer is greater than the current end of file position and the channel is open for reading only, then the new file pointer is set to the current end of file position and a status code is returned.

(c) If the requested file pointer is greater than the current end of file position and the channel is open for writing, then the new file position is set to the current end of file position and nulls are written to the file until the file position is equal to the desired value.

The **IOTRUNC** call truncates a file by setting its end of file pointer to be equal to the current position. Because it does not make sense to truncate a sequential output channel (it is always positioned at end of file) this condition is trapped and ignored by IOSS.

The **IOGETPOS** call returns the current position of the channel to the calling program. This call is valid for sequential channels although the information acquired cannot be used in a call of IOSETPOS.

The **IOEOF** call determines whether the channel is positioned at end of file and returns a yes or no answer to the calling program. For a sequential output file the answer is always yes. For any random access channel the answer is yes if the current position is equal to the end of file pointer and no otherwise. For a sequential input channel the answer is yes if end of file was encountered on the last read and no otherwise.

The **IOSIZE** call returns the size of the file attached to the channel in bytes. This information is always available for random files, and for sequential output channels the size is equal to the current position because writing always takes place at end of file. Devices that do not maintain end of file pointers will return an error status code.

3.20 Polling an Input Channel

The **IOREADY** call determines whether any input is immediately available from the given channel without suspending the calling program and allows user programs to react to real-time events, such as keystrokes.

The results obtained from this call are device dependent as follows:

(a) If the device is a time dependent device with input arriving outside the control of the operating system the answer is yes if a read call for a single byte would be satisfied immediately and no if such a call would have to wait for something to happen.

(b) Otherwise the answer is yes, unless the channel is positioned at end of file in which case it is no.

The devices which would give category (a) response include keyboard, serial communications and pipes. Devices in category (b) include disk or microdrive files, even though reading the next character from the file might take an appreciable length of time.

3.21 **Mounting and Dismounting Directories**

For disk and microdrive devices it is necessary to tell the device driver explicitly that a particular directory is available before it can be used, and to tell the driver that a particular directory is no longer required and can be removed from the system.

The **IOMOUNT** call passes a device dependent unit number to the device driver, typically a small integer specifying a drive or port number. For this call to succeed a variety of device specific conditions may need to be met, which might include, for example:

(a) The unit does not already contain a mounted disk or capsule.

(b) The disk or capsule is physically present in the drive.

IOSS (and the device driver) attempt to mount whatever is found on the specified device. Some device drivers will be capable of automatically dismounting anything which is already using that unit.

IOSS checks that the directory found matches the directory specified in the supplied path name, though if this was null then any directory found will be successfully mounted. The name of the directory found will be returned to the user as a string.

The **IODISMOU** call ends the association between the directory and the unit number specified in the IOMOUNT call. For this call to succeed it is usually a requirement that there are no files currently open on the directory (though this is strictly a device specific condition).

The directory can either be dismounted by name (in which case the unit number is ignored) or, if the name is null, by unit number (in which case any directory found on the specified unit is dismounted).

## 3.22 Device Driver Special Function

The **IOSPECIA** call is provided to perform any peculiar function which is applicable to a single device and not appropriate to supply as a general IOSS function, such as setting a serial line baud rate.

The path name and program/data indicator identify the device: what the device does, what the parameters mean and what results are returned are entirely up to the device. The IOSS performs no action at all on this call apart from checking that special operations are actually allowed on this device and passing the data to and from the device driver.

# SECTION 4:

# OPERATING SYSTEM FUNCTIONS

# 4 OPERATING SYSTEM FUNCTIONS

## 4.1 Overview of OS Functions

The functions provided in this category fall under six main headings, each consisting of a group of related routines:

(a) **Program Manager:** these routines perform create and delete operations on applications programs.

(b) **Memory Manager:** these routines provide applications software with facilities for the allocation and release of system memory.

(c) **Menu Manager:** this software provides facilities for the display, data entry and data capture of complex forms.

(d) **Timing Services:** routines are provided to perform timed delays and to read and set the internal calendar clock.

(e) **Heap Allocation:** these routines perform user heap management.

(f) **User Trap Handler:** this allows applications programs to redirect certain trap vectors to user written routines.

These groups of functions are not related in any structural fashion but instead form a conveniently sized set of entry points to be assigned to a single vector routine.

OS routines are called synchronously as subroutines of the calling applications program and either act as straightforward subroutines or communicate with an asynchronous sytem component using semaphores. In the latter case, the complexities are transparent to the applications program.

## 4.2 Calling OS Routines

OS routines are called by applications software via a single entry point with the function code in D0, as follows:

```
MOVEQ    #OSFUNC,D0
JSR      OSENTRY
```

On return D0 contains a status code.

Descriptions of each OS routine follow below, and precise details of each OS routine are given at Appendix B.

### 4.3 Program Manager Functions

The program manager is a collection of system subroutines and system programs that performs a variety of tasks concerned with the creation, deletion and examination of applications programs.

A program is an asynchronous process that consists of at least one procedure plus a program control block and a data area that contains its stack and heap.

A program can own other programs and it keeps a list of these (the program list) in its PCB. Note that program ownership can be nested to any level, enabling the formation of family trees of related programs.

A program is only permitted to use program manager functions on its own child programs. The program manager functions will fail if any attempt is made to operate on other programs. However, a program kill function applied to a child program will recursively be applied to the entire family sub-tree of programs owned by the child.

Program manager functions include facilities to:

(a) Start a new program.

(b) Investigate the state of a program.

(c) Terminate a program, tidying up all its resources.

The termination function is highly complex, having impact on a number of system functions. In principle, the program manager can cope with both normal termination and program aborts (normally invoked by system error traps), provided that any abnormal termination has not involved destruction of any system data structures. The major requirement in either case is to release all of the resources owned by the terminated program.

### 4.4 Initial Program State

When a newly created applications program is set into run state by the scheduler, the following values are present in its registers (and the corresponding locations in the PCB) immediately prior to executing the first instruction:

A1 Address of the parameter string passed from the parent

A5 Address of the program's PCB

A7 User stack pointer

This is usually the only information that an applications program will require to perform normal functions under 68K/OS. Systems programs may need more details of the initial state of the program control block and other system data structures (as found in the Systems Programmer's Reference Manual).

## 4.5 Starting a New Child Program

The **OSSTART** function performs the actions necessary to start a program:

(a) Load the procedure specified and set up the PCB.

(b) Chain a program list element to the calling program's PCB.

(c) Grab the greater of the RAM memory requirements specified by the caller or in the procedure entry control block.

(d) Allocate the program a priority less than the caller.

(e) Allocate an initial program state (either suspended or ready).

(f) Pass the program the address of a parameter string.

The created program becomes a child of the calling program.

## 4.6 Determine Program Status

The **OSSTATUS** function enables a program to find out whether a child program is still running or has finished.

## 4.7 Wait for a Program to Finish

The **OSWAIT** function waits for a child program to finish and returns its program list element to the parent program, so that the parent program may examine the results.

The program list element contains two status codes and a return string which provide the caller with information concerning the termination of the child program. When the caller has finished with the program list element it should return it to the heap using OSHEAPDE.

## 4.8 Force Program Termination

The **OSKILL** function causes a child program to stop by diverting its program counter to a TRAP #0 instruction. Control returns to the user before the child program stops, so the caller must use OSSTATUS to check the status of the program or OSWAIT to wait for it to actually stop.

The child program is allowed to finish any critical system code that it is executing, prior to having its program counter diverted. It will terminate in the same way as if it had voluntarily executed TRAP #0.

Note that OSKILL cannot be carried out by the caller on itself. A program must terminate itself by executing either a TRAP #0 or an RTS.

## 4.9 Memory Manager Functions

The memory manager is a set of subroutines which controls the allocation of RAM memory to programs, slaved microdrive or disk blocks and other system components.

Functions are provided for the following:

(a) Allocation of memory for use by a program.

(b) Change of memory ownership information.

(c) Deallocation of memory by ownership identifier.

(d) Deallocation of memory by address range.

Applications software will usually only need to grab extra memory, because it will be released automatically when the program terminates. The remaining functions are provided for systems programming use.

## 4.10 Allocate Extra RAM to a Program

The **OSMEMALL** function will attempt to allocate a contiguous area of RAM of the specified size (in units of 1Kb), and if successful will store the supplied ownership information in the system memory map entries corresponding to the RAM allocated. The ownership identifier should normally be set equal to the calling program's PCB address, because this will ensure automatic memory release on program termination.

## 4.11 Change Ownership Information

The **OSMEMOWN** function sets a given value in the ownership field of the memory map entry for a given range of blocks which were allocated with the OSMEMALL routine. This can be used by system programmers to transfer memory resources from one program to another or to retain memory after a program is terminated.

## 4.12 Release Memory by Ownership Information

The **OSMEMDA** function deallocates all memory blocks with a specified value of the ownership information field.

## 4.13 Release Memory by Address Range

The **OSMEMDS** function deallocates a specified number of 1Kb memory blocks whose start address must be explicitly identified by the calling program.

### 4.14 The Menu Manager

The Menu Manager is a set of subroutines that interface between applications software and display file manager routines, that are provided to simplify form filling and menu selection operations and to provide a consistent user interface for menu driven applications software.

A menu consists of one or two display files which are shown in different screen windows:

(a) The menu window contains a form which is constructed from protected heading fields, variable message fields and variable input fields. When this window is displayed on the screen the user can tab between the input fields, enter and edit data, and select options using function keys.

(b) The (optional) list selection window displays a scrollable list of items from which the user can select an item and copy it into any input field in the menu window.

Note that these display files and windows are not initialised by the menu manager and must be set up by the applications program explicitly using standard display file manager initialisation routines.

Two cursors are used, one in the menu window which may be moved between variable input fields by means of the TAB key, and a second in the list window which may be moved up and down the list with the cursor keys.

### 4.15 Menu Data Structures

Two data structures are required and maintained by the menu manager:

(c) The menu fixed data structure is used to specify field definitions including protection status, ink and paper colours and any fixed heading text that must be displayed. This data structure is static and can be held in ROM if required.

(d) The menu variable data structure is initially created from the fixed data structure and represents (in compact form) the current state of the menu display file shown in the menu window. The applications program need not know the detailed format of the data structure because menu manager routines are provided to read and update specific menu fields.

The menu variable data structure is initially presented to the menu manager as an empty string which must be large enough to hold the menu. The memory required for this must be obtained and disposed of by the applications program.

For details of the menu fixed data structure see section 8.

## 4.16 Display Fixed Menu Data

The **OSMENDIS** function clears the specified display file, copies the fixed data to the display file (and hence the screen) and initialises the fields in the variable data structure. This routine is called once for each new menu displayed.

## 4.17 Read User Input to Menu

The **OSMENRD** function interacts with the operator when he fills in the form or selects menu options, as follows:

(a) CHARACTER keystrokes are echoed at the cursor position in the current variable input field of the menu.

(b) The TAB and BACKTAB keystrokes move the menu window cursor between the variable input fields.

(c) The CURSOR LEFT, BACKSPACE-DELETE and DELETE LINE keystrokes are used to edit the contents of a variable field.

(d) The CURSOR UP and CURSOR DOWN keystrokes move the list selection cursor up and down the list window.

(e) The ESCAPE keystroke copies an item from the list window to the a menu window input field. The item and field are specified by the positions of the two cursors.

(f) The FUNCTION CODE and ENTER keystrokes return control to the user after copying the data from the variable input fields into the menu variable data structure.

(g) Other keystrokes are ignored.

The list selection window is optional: where none is displayed, the keystrokes in (d) and (e) are ignored.

Up to fifteen FUNCTION CODEs can be used, these plus ENTER are returned to the calling program as bits in a sixteen-bit word.

## 4.18 Read a Variable Field

The **OSMENGET** function extracts the contents of the specified field from the menu variable data structure and returns it to the user as a string of characters.

## 4.19 Update a Variable Field

The **OSMENPUT** function is the complement of CSMENGET. The string supplied by the user updates the contents of the specified field in the menu variable data structure. This will subsequently be displayed on the screen after the next call of OSMENRD.

## 4.20 Timing Services

These fall into two distinct categories (representing the two hardware clocks supported):

(a) Passive real-time clock delay routine

(b) Hardware calendar clock support routines

The real-time clock is mandatory but may operate at either 50Hz or 60Hz depending on the mains supply. The hardware calendar clock is optional and may not be present in some implementations of 68K/OS.

## 4.21 Passive Delay

The **OSDELAY** function suspends the calling program for the specified number of 50/60Hz real-time clock ticks, allowing other programs to run in the meantime.

This function does not provide a very accurate timing mechanism, for a number of reasons:

(a) The request to start the delay can occur at any time during the clock cycle, so a request to delay for one clock period actually causes the program to wait for any time from zero to one cycle.

(b) When the processor is heavily loaded clock ticks may be ignored altogether by both hardware and software at various levels, thus under these conditions it is possible for a program to be delayed for longer than specified.

(c) When the delay software wakes the program up it may take some time before it resumes running because high priority system processes are also invoked periodically on clock ticks.

If the number of ticks on entry is zero or negative, the calling program is delayed for one clock tick.

## 4.22 Read Binary Time and Date

The **OSBINCLK** function reads the hardware calendar clock and returns the time and date as a binary value. This is defined to be the number of seconds that have elapsed since 00:00 hours on 1st January 1983.

## 4.23 Set Binary Time and Date

The **OSSETCLK** function sets the hardware calendar clock with a binary value representing the time and date. This is defined to be the number of seconds that have elapsed since 00:00 hours on 1st January 1983.

## 4.24 Heap Allocation

All applications programs must have an area of storage called a heap which is used to allocate variable sized records for a variety of purposes on an ad hoc basis. Programs are allocated heap storage when they are started by the program manager, and this is subsequently used transparently by a large number of 68K/OS system calls.

To enable applications programs to allocate and deallocate records from their own heap, routines are provided that perform the heap management functions required.

## 4.25 Allocate a Heap Record

The **OSHEAPAL** function allocates a record of the specified size from the heap and returns its address to the calling program.

## 4.26 Deallocate a Heap Record

The **OSHEAPDE** function returns the specified record to the heap free pool. Adjacent free records are coalesced. If the record lies outside the address range of the calling program's heap, the call is ignored.

## 4.27 Determine the Free Stack/Heap Space

A program's stack and heap share the same area of memory but grow from opposite ends of this area, the stack growing down from the high address and the heap growing up from the low address.

**OSAVAIL** allows a program to enquire about the free space remaining and returns three values:

(a) The size of the largest free heap record.

(b) The total size of all free heap records.

(c) The size of the gap between the top of stack and the top of heap.

It follows that the largest possible heap record available to the user program is the greater of (a) and (c).

## 4.28 User Trap Handler

By default, those exception trap vectors not used by 68K/OS address a routine which will terminate the calling program, since in most cases accidental invocation of a trap is caused by the program running wild.

**OSTRAP** allows user programs to change the contents of the following exception trap vector to address a user trap routine:

| | |
|---|---|
| EAADDRES | Odd address |
| EAILLEGA | Illegal instruction |
| EADIVIDE | Divide by zero |
| EACHKINS | Array bound violation |
| EATRAPV | Arithmetic overflow |
| EAPRIV | Privileged instruction |
| EATRACE | Trace mode exception |
| EAALINE | A-line exception |
| EAFLINE | F-line exception |
| EATRAP4 | User trap 4 |
| EATRAP5 | User trap 5 |
| EATRAP6 | User trap 6 |
| EATRAP7 | User trap 7 |
| EATRAP8 | User trap 8 |
| EATRAP9 | User trap 9 |
| EATRAP10 | User trap 10 |
| EATRAP11 | User trap 11 |
| EATRAP12 | User trap 12 |
| EATRAP13 | User trap 13 |
| EATRAP14 | User trap 14 |
| EATRAP15 | User trap 15 |

Note that user trap 4 and trace mode exceptions are special cases that vector to the user defined routine in supervisor mode, all other traps will vector in user mode.

# SECTION 5:

# DISPLAY FILE MANAGER

# 5 DISPLAY FILE MANAGER

## 5.1 Outline Description

The Display File Manager is a set of subroutines that controls access to the screen by applications programs. DFM permits concurrent programs to share the available screen area between them, and will ensure that their screen areas do not interact.

DFM operates on a logical screen which is a mapping onto a physical screen. This mapping depends on the hardware implementation and/or the screen mode selected (TV or monitor).

Physical screen output is achieved using the screen driver, which is called synchronously from within DFM. The screen driver should not be called direct by applications software under any cicumstances whatever.

Although 68K/OS graphics software calls the screen driver direct, a DFM window is supplied as a parameter to each graphics routine and figures drawn will be clipped at window boundaries. In this case DFM has an indirect effect on the integrity of the screen.

## 5.2 Partitions

If a program requires an area on the screen it is allocated a partition by DFM. A partition is a variable sized horizontal slice of the logical screen which is divided from other screen partitions by a single pixel high rule. The screen may be divided into any number of partitions provided that each displays at least one line.

The size of partitions is under direct user control from the keyboard, and any partition can be grown or shrunk by any amount provided that no partition is reduced to less than one line.

Partitions are owned by programs and can only be updated by their owners. When a program is terminated its partition is deleted and the other partition(s) will expand to fill the space released.

## 5.3 Virtual Screens

Because a screen partition is under direct operator control and competes for screen resources with other partitions, a program cannot know the size of its partition (which may only display a portion of the logical area that the program wishes to display). This problem is solved by the maintenance of a virtual screen for each program.

A virtual screen defines the program's logical screen dimensions and its division into windows. It is not a separate physical copy of the screen but a complex data structure which maintains the text associated with windows in a set of linked lists known as display files.

A virtual screen can be scrolled through a partition by DFM or by user keyboard control. This is termed metascrolling.

## 5.4 Windows

Intially a virtual screen consists of a single rectangular window whose size is identical to the requested screen partition size. The initial window can be subdivided by a program by creating new windows.

A new window is created by splitting an existing rectangular window either vertically or horizontally into two smaller rectangular windows. This process can be repeated recursively to divide the virtual screen into several windows, but will always ensure that:

(a) All windows are rectangular

(b) There are no gaps of any shape

A program can create and delete windows dynamically provided that windows are deleted in reverse order of their creation. While a window exists, its size and position within the virtual screen are static.

## 5.5 Display Files

Each window is associated with a display file which holds an internal representation of the display text and is potentially far larger than the window itself. The display file is independent of the window and is not deleted if the window is removed from the virtual screen.

The display file can be scrolled through the window either vertically or horizontally by DFM. This scrolling is distinct from metascrolling.

The display file holds details of default ink and paper colours for the text and the window background colour. Special commands are provided to change ink and paper colours and character fount (see 5.9 and F.8).

## 5.6 Extended Display Files

Each display file is allocated an area of memory in which to store text which it organises as a heap. This area cannot be expanded dynamically, and there is a possiblity that the display file will be filled up and exhaust the heap.

To overcome this problem DFM allows the calling program to install a user written subroutine (the 'user hook' routine) that will be called by DFM whenever the display file is full. This routine could:

(a) Output the top line of the display file to an IOSS channel, providing a log facility.

(b) Maintain two IOSS channels, one for each end of the display file which are attached to disk or microdrive files, providing extended scrolling onto files. (This is how the GST screen editor works.)

Alternatively, DFM can be instructed to throw away the top line of the display file when this becomes full.

## 5.7 Cursor, Action Pointer and Markers

Each display file maintains two pointers into the text called the cursor and the action pointer. The cursor can be moved by DFM to point to any display character in any line of the display file, and when the cursor is moved the action pointer is set to the same value. The action pointer can be moved within the current cursor line and can point to both commands and display characters.

When displayed, the position of the cursor is represented on the screen by an inverse video block. If the display file is associated with a window then DFM will always ensure that the cursor is visible in the window, but cannot always guarantee that the window line containing the cursor is visible in the partition.

If a program has several windows it can define one of these to be fixed in the partition. The cursor in this window will flash and DFM will always ensure that it is visible in the partition, first by scrolling the display file, and if necessary by metascrolling the virtual screen.

The display file manager maintains up to eight position markers in each display file. These can be set by the user, and the cursor can be moved to a marked position.

## 5.8 Console Display File Interface and IOSS

Display files can be accessed via the console display file interface. This allows a subset of display file operations to be performed by programs which have simple requirements.

A special interface between IOSS and DFM is provided to enable console display files to be accessed through IOSS without calling DFM directly. If a program calls IOOPEN to open a sequential output channel to the device SCREEN: or calls IOGETLIN to read a line from device KEY: then the system will, if necessary, create a display file and an associated window. Further calls to IOSS have the following effects:

(a) IOPUTSEQ and IOPUTLIN calls to SCREEN: will output data to the display file's screen window.

(b) IOGETLIN calls to KEY: will reflect each keystroke in the display file's screen window (see also section 3.17).

A program can select an existing display file for use with the console interface and this display file will be used by IOSS when required.

## 5.9 Display File Binary Commands

The display file manager maintains data in the display file connected with colour, founts, underlining and spacing which is interpreted by the screen driver (see F.7).

This data is included in a display file by inserting a binary command consisting of two bytes, the first a command code and the second a parameter, both having the top bit set to distinguish them from text.

A subset of these are available as user defined commands that programs may insert in display files for their own private use. These will be ignored by DFM and the screen driver.

## 5.10 Single Line Menu

The bottom screen line is maintained separately by DFM and is never allocated to screen partitions. This line is used by applications and system programs to display program identification, single line menus, messages or actions assigned to the function keys.

The user can select which partition (and hence which program) he wishes to receive input from the keyboard. This program has exclusive use of the single line menu.

The keyboard can be used to talk directly to the operating system by switching into system mode. This allows the user to grow, shrink and metascroll partitions, select current programs and change their status. When in system mode, the operating system itself uses the single line menu to display actions assigned to the function keys.

## 5.11 Calling DFM Routines

DFM routines are called by applications software via a single entry point with a function code in D0:

```
MOVEQ     #DMFUNC,D0
JSR       DMENTRY
```

On return D0 contains a status code.

Descriptions of each DFM routine follow below, and precise details of each DFM call is given in Appendix C.

## 5.12 Initialisation Routines

The **DMINITVS** call creates an (initially empty) virtual screen for the program and will allocate a screen partition of the defined size. The new partition appears at the bottom of the screen, but above the single line menu.

The **DMINITDF** call will perform all the initialisation required to create a new empty display file, including allocation of space from the calling program's heap (if required) and the initialisation of all internal data structures.

The **DMNEWWIN** call will add a new window to the virtual screen and will display the associated display file on the screen.

The **DMRESET** routine will delete all the text in a display file and reset the data structures to their initialised state.

## 5.13 Termination Routines

The **DMFLUSH** call will empty the display file by repeated calls of the user hook routine which should write this to the top output file.

The **DMKILWIN** call will remove a window from the calling program's virtual screen, if this was the last window created.

The **DMKILLDF** call will release the display file data area after first calling DMFLUSH to write out the data to the top file.

## 5.14 Display File Control Routines

The **DMTTYSEL** call will select the specified display file as the current console window to be used for IOSS output to the SCREEN: device and for keyboard reflection using IOGETLIN with the KEY: device.

The **DMFIXDF** call specifies which window should always be kept visible within the partition and will cause the cursor in that window to flash.

The **DMDISABL** call will forbid screen update for the specified display file until reenabled. This is required when a complex operation such as paragraph reformat takes place to avoid both the time overheads of intermediate line repaints and the resulting unpleasant visual effects.

The **DMENABL** call will reenable screen update that has been disabled.

## 5.15 Space Allocation Routines

These calls are normally made direct from within DFM, but are provided to enable the user hook routine to share the same display file heap.

The **DMALLOC** call grabs a record of specified size from the display file heap, **DMRELEAS** will return a record to the heap.

## 5.16 Line Manipulation Routines

The **DMINSLIN** call inserts a line into the display file immediately above the line in which the cursor is positioned. If this is on the screen then lines below it will be automatically scrolled down by the display file manager.

The **DMDELLIN** call removes the line in which the cursor is positioned from the display file and returns it to the heap. The rest of the window will be scrolled up and the cursor is left at the start of the next line.

The **DMJOIN** call joins the line containing the cursor with the following line. Further lines are scrolled up.

The **DMSPLIT** call will split the current line into two immediately before the cursor position. Further lines in the window are scrolled down.

## 5.17 Character Manipulation Routines

The **DMRDBYT** call will move the action pointer by the specified amount and return the display file byte referenced to the calling program, which need not be a display character. When the action pointer is at the end of line, a newline code is returned.

The **DMWRBYT** call will replace the display file byte referenced by the action pointer by the byte specified, which need not be a display character. This routine must not be called if the pointer is at the end of line.

The **DMINSCHR** call will insert the display character specified at the cursor position, shifting the remainder of the line to the right.

The **DMDELCHR** call deletes the display character referenced by the cursor. The rest of the line is shifted left.

## 5.18 String Manipulation Routines

The **DMINSSTR** call inserts a string (whose length is passed in the first two bytes) into the display file at the cursor position. Note that the string may contain newline characters, in which case the string will be inserted in sections, DMSPLIT being called internally to start new lines. The cursor is left on the character after the inserted string. The string can contain binary data and display characters.

The **DMINSBLK** call is identical to DMINSSTR, but the data and bytecount are passed separately.

The **DMDELCMD** call deletes a two byte binary command from a display file. The action pointer must be pointing to the first byte of the command.

## 5.19 Cursor Routines

The **DMMOVECU** call allows the program to move the cursor (together with the action pointer) around the display file. The movement is specified as up, down, left, right or a number in the range 0-7 indicating a predefined marker position. If the cursor is moved out of the window, the window will be scrolled until the cursor is visible.

The **DMPUTCUR** call will position the cursor and action pointer at a specified line number and character position within the line.

The **DMGETCUR** call returns the current position of the cursor to the calling program as a line number and character position.

The **DMCURDIS** call disables the cursor in the specified window. This can be used when the cursor is not required (eg, in a help window) or temporarily to hide the cursor to improve screen appearance.

The **DMCURENA** call reenables the cursor in the specified window.

## 5.20 Marker Position Routines

The **DMMARK** call sets the specified marker to reference the current cursor position. Up to eight marker positions can be used.

The **DMMKPOS** call returns the position of the specified marker.

## 5.21 Update Single Line Menu

The **DMUMENU** call permits a program to display a line of text in the single line menu area at the bottom of the screen. It will only be displayed when the program is selected as the current program for keyboard input.

## 5.22 Install User Hook Routine

The **DMHOOK** call defines to DFM the address of a user routine to provide scrolling for an extended display file on and off a backing medium such as disk or microdrive. This routine will be called by DFMwhen:

(a) DFM requires to write a line to backing store. This line may be written from the top or bottom of the display file.

(b) DFM requires to read a line from backing store. This line may be read into the top or bottom of the display file.

The hook routine must, in principle, maintain two files to cope with the data from either end of the file, though for some applications where the display file is only scrolled in only one direction a single file or sequential output channel will be sufficient. No hook routine is required if a simple console window is selected because DFM will dispose of lines itself.

SECTION 6:

GRAPHICS ROUTINES

# 6 GRAPHICS ROUTINES

## 6.1 General Description

68K/OS graphics routines provide a mechanism for applications software to draw medium-resolution graphics figures. These figures are drawn in display file windows and are positioned relative to the 'origin' of the display files, and will be clipped according to the current window boundaries when the figure is drawn.

Although graphics and text may be mixed in the same display file window and use the same coordinate system, the system does not maintain graphics display files outside of the window currently visible on the screen. Thus it is not possible to scroll graphics through a window (in the same way as text) as an automatic system function, although this can be achieved by applications software if desired. If a graphics figure is scrolled out of and back into a window, DFM will repaint the scrolled area in the current window background colour.

Because 68K/OS does not contain internal trigonometric functions, the range of figures available is restricted, and two-dimensional figures must be drawn with orthogonal axes.

Up to eight colours are supported and these can be mixed in four-pixel block patterns to form a variety of stipple effects, producing a large number of pseudo-colours. Note that the QL hardware only supports four colours in 85, 80 and 60 column modes (blue is suppressed).

## 6.2 Coordinate System

The graphics coordinate system is relative to three separate screen origin offset mechanisms:

(a) The logical screen origin may be displaced in both axes from the physical screen origin if a TV compatible mode is selected.

(b) The display file window origin will be relative to the positions of the partition and the virtual screen.

(c) The applications software may define a graphics window whose origin is relative to the display file origin.

Note that in each case the origin is the top left-hand corner of the item described, and that once the origin offset and window size has been defined, the positioning, scrolling and clipping of graphics figures within the coordinate system is automatic.

Coordinates in both the X and Y axes are defined in screen pixels where the full physical screen is 512x256 pixels, regardless of whether the QL is in four or eight colour mode. In the latter, the bottom bit of the X coordinate is ignored.

The dimensions and origin position of the graphics window are defined in character units. This enables graphics windows to map directly onto display file windows.

## 6.3 Colour Definitions and Stipple Patterns

Colour definitions for graphics routines are defined in a word, the upper byte of which is set non-zero if the figure is to be drawn in XOR ink. The lower byte is defined as follows:

Bits 7-6 Stipple (0 = Q, 1 = H, 2 = V, 3 = C)
Bits 5-3 XOR of mixer colour and base colour
Bits 2-0 Base colour

where the stipple codes refer to a 2x2 pixel block, as follows:

Q = quarter mixer, three-quarters base
H = horizontal stripes of base and mixer
V = vertical stripes of base and mixer
C = checkerboard of base and mixer

Note that if bits 5-3 are zero then the plain base colour is drawn and the defined stipple pattern has no effect.

Colours are specified as numbers in the range 0-7 as follows:

| | | | |
|---|---|---|---|
| 0 | Black | 4 | Green |
| 1 | Blue | 5 | Cyan |
| 2 | Red | 6 | Yellow |
| 3 | Magenta | 7 | White |

In four-colour mode (85, 80 or 60 columns), blue is suppressed, giving the following:

| | | | |
|---|---|---|---|
| 0-1 | Black | 4-5 | Green |
| 2-3 | Red | 6-7 | White |

## 6.4 Aspect Ratio

Because the y-dimension of the physical pixel exceeds the x-dimension by a factor of approximately 3:2, the QL screen aspect ratio is non-square and will vary depending on the particular monitor or television used. Because the coordinate system is based on physical pixels it will be necessary to set the x-dimension some 25% to 35% larger than the y-dimension in order to draw circular ellipses or square blocks.

## 6.5 Calling Graphics Routines

Because the graphics interface may change with later implementations of 68K/OS on different machines, these routines are called via the system dependent SPENTRY vector with a function code in DO:

```
MOVEQ   #SPFUNC,DO
JSR     SPENTRY
```

On return, DO will be destroyed, all other registers are preserved.

## 6.6 Graphics Figures

The following graphics figures can be drawn:

(a) **SPPOINT** draws a single pixel

(b) **SPLINE** draws a straight line

(c) **SPELLIPS** draws an orthogonal ellipse

(d) **SPBLOCK** draws an orthogonal filled rectangular block

(e) **SPTEXT** draws a text string horizontally in various sizes

(f) **SPPAINT** fills an area to an unspecified border

(g) **SPFILL** fills an area to a specified border

All these figures are clipped to window boundaries.

# SECTION 7:

# CREATING PROGRAMS

# AND PROCEDURES

# 7 CREATING PROGRAMS AND PROCEDURES

## 7.1 Overview

This section defines the general programming requirements and specific entry and exit requirements for programs or procedures, and the initial values of pointers and registers when a program is started by the OSSTART command.

## 7.2 Position Independence

All procedures written to run under 68K/OS must be position independent because the user has no control over the position in which his code is loaded into memory. This requirement restricts the addressing modes available to the programmer when making references internal to his program. In general, all internal references to addresses must use (or be derived from) PC relative mode. Absolute short or long addresses must only be used when referring external system routines and fixed position hardware registers, system tables and variables.

## 7.3 Reentrant Code

All 68K/OS procedures must be reentrant because it is possible that a single copy of a procedure might be executed simultaneously by two programs. As a general rule this implies that a procedure must be written in pure (read only) code which guarantees that it will be reentrant and will also ensure that the code can be executed in ROM.

## 7.4 Procedure Header Block

A 32-byte header block must be coded at the start of each procedure, and has the following format:

| | |
|---|---|
| PEENTRY(.L) | PC relative procedure entrypoint |
| PERAM(.W) | Minimum RAM allocation for program |

followed by the procedure name (in standard IOSS pathname format) if it is intended to include the program in the ROM: directory.

PERAM contains the procedure's RAM requirement for PCB, stack and heap (in units of 1Kb) if it were to be invoked as a program by an OSSTART call. If this is set to zero, the procedure cannot be run as a stand-alone program.

## 7.5 Program Memory Requirements

Unless a program is ROM resident the system will load it to an area of RAM whose size is known from the directory entry. A second non-contiguous area of RAM (whose size is the greater of the PERAM entry or a parameter to OSSTART) is allocated for the PCB, stack and heap.

## 7.6 Program Memory Layout

The stack/heap area allocated to a program is laid out as follows:

```
                     ______________________
                    |                      |   High address
                    |       PROGRAM        |
                    |       CONTROL        |
                    |       BLOCK          |
                    |                      |
                    |       (PCB)          |
PCB         --->    |______________________|
                    |                      |
                    |                      |
                    |       STACK          |
                    |                      |
A7          --->    |______________________|
                    |                      |
                    |                      |
                    |        SPARE         |
                    |        HEAP/         |
                    |        STACK         |
                    |                      |
PBHEAP      --->    |______________________|
                    |                      |
                    |        HEAP          |
                    |                      |
PBHEBASE    --->    |______________________|   Low address
```

## 7.7 Data Area Pointers

The pointers to the user program's data area are:

(a) **A7.** This register is the user stack pointer and always addresses the last word that has been allocated on the stack (which grows down from high memory).

(b) **PBHEAP.** This symbol is an offset in the PCB where the heap pointer is kept. This always points to the first free word above the top of the heap (which grows up from low memory).

(c) **PBHEBASE.** This symbol is an offset in the PCB where the pointer to the base of the heap is kept. This always points to the first byte allocated to the program.

These addresses are aligned on word boundaries.

## 7.8 Special Conditions at Start of Program

At the start of a program the stack contains the return address into the operating system's program termination routine, but is otherwise empty. This address is planted on the stack by the program manager to enable a program to terminate using an RTS intruction.

The heap is used by the system during program initialisation to build those system data structures required for a new applications program, thus PBHEAP will be greater than PBHEBASE.

A5 points to the program control block (required by calls to OSMEMALL).

A1 points to a (possibly null) parameter string passed from the parent program. The standard parameter string is defined in section 8.

## 7.9 Program and Procedure Exit

To terminate either a program or a procedure, the final instruction executed should be:

```
RTS
```

Additionally, a program may be terminated by executing:

```
TRAP    #0
```

Note that use of RTS is preferable since it allows a module to be executed either as a program or a procedure, whereas a TRAP #0 executed by a procedure will terminate the program that called the procedure.

## 7.10 Passing Status Parameters

When a program terminates either normally or because of an error, it is possible to pass completion status parameters back to its parent in registers that are transferred to the program list element:

| | |
|---|---|
| D0.W | status code (zero for successful completion) |
| D1.W | return code (applications specific) |
| A0.L | pointer to return text string |

The status code will normally be zero or a system status code returned from a system call.

The return code is, strictly speaking, applications dependent and can be used to pass completion status in a suite of applications programs. Stand-alone programs should clear the return code.

The return text string is free format and is up to 46 bytes long (including the two-byte string length).

# SECTION 8:

# SYSTEM DATA STRUCTURES

# 8 SYSTEM DATA STRUCTURES

## 8.1 Scope

Included in this section are descriptions of those system data structures that may usefully (and safely) be referenced as data from applications software, namely:

* Directory entry buffer
* Directory information buffer
* Menu fixed data structure
* Procedure entry control block
* Program list element
* Standard parameter string
* Standard text string

The remainder of system data structures should only be referenced from systems software and are defined in the Systems Programmer's Reference Manual.

## 8.2 Notation

The following notation is used to define data structures:

(a) Field names are defined as upper case symbols:

FIELDONE

(b) Field lengths are defined as byte, word or longword (which can be accessed directly as .B, .W or .L) or number of bytes:

BYTELEN (.B)
WORDLEN (.W)
LONGLEN (.L)
NUMBLEN (32)

(c) Field containing bit-length values are explicitly defined:

BITFIELD (.B) contains the following significant bits:

FLAGBIT1
FLAGBIT2

All other 'spare' bits are undefined but should be set to zero to allow for future expansion.

In all cases, the numeric offsets from the start of the record are not defined. The applications programmer should code using the symbols defined in this manual and should include the following directive at the start of the source file:

```
INCLUDE    68KOS.IN              68K/OS parameters
```

This is the main system parameter file which will contain current definitions of all data structure symbols.

## 8.3 Directory Entry Buffer

The directory entry buffer is an area of memory returned from an IOGETDIR call which contains information about a single file in a directory. The layout of the buffer is as follows:

| | | |
|---|---|---|
| DEATDIR | (.B) | access type of the directory |
| DEOPTION | (.B) | access type and mode for this file |
| DEEOF | (.L) | file size in bytes |
| DECREATE | (.L) | creation date |
| DEMODIFY | (.L) | date last modified |
| DEPATH | (19) | filename string |
| DECOMM | (28) | user comment string |

The DEATDIR and DEOPTION fields contain bits to indicate the access type and mode for the directory and file. Three bit fields are significant:

| | | |
|---|---|---|
| OPREAD | 0 = read disabled | 1 = read enabled |
| OPWRITE | 0 = write disabled | 1 = write enabled |
| OPRAN | 0 = sequential access | 1 = random access |

The DEATDIR field contains the combined (most restrictive) file access permissions of both the directory and the file.

The DECREATE and DEMODIFY fields hold the relevant binary system time (defined to be the number of seconds elapsed since 00:00 on 1st January 1983).

Strings are in standard string format as defined in 8.9 below.

## 8.4 Directory Information Buffer

The directory information buffer is an area of memory returned from an IODIRINF call that contains information about an entire directory. The layout of the buffer is as follows:

| | | |
|---|---|---|
| DIENTRY | (.L) | number of directory entries |
| DITOTAL | (.L) | total space available in directory |
| DILEFT | (.L) | current space remaining in directory |

The DIENTRY field is a number greater than or equal to the actual number of entries in use, and its value is device dependent. It gives an upper bound on the number of entries for use by, for example, a sort routine.

The DITOTAL field gives the total free space of an empty directory and the DILEFT field gives the current free space in the specified directory, both figures in units of 1Kb.

## 8.5 Menu Fixed Data Structure

The menu fixed data structure is stored in a standard text string which is read into a display file and displayed on the screen by OSMENDIS. This string contains fixed text, menu formatting information and variable input or message field definitions as follows:

(a) **Display character:** a character is within the standard ASCII range and is displayed in the current fount and foreground/background colours at the next character position on the virtual screen.

(b) **Display command:** a command is a two-byte record to define special action by the display file manager or screen driver (such as a fount or colour change).

(c) **Code MU.NL:** this is used to start a new line in the menu display.

(d) **Code MU.CLB:** this is used to indicate a conditional line break position and consists of two bytes:

MU.CLB code
Number of characters to fit on the line

If sufficient character positions remain in the window line then the MU.CLB code is replaced by a single space, otherwise it has the same effect as a MU.NL code.

(e) **Code MU.HT:** this is used to tab conditionally to the next menu column and consists of two bytes:

MU.HT code
Number of characters to fit on the line

If sufficient character positions remain in the window line then the MU.HT code is replaced by the number of spaces required to align the specified number of characters on the right margin, otherwise it has the same effect as MU.NL.

(f) **Code MU.ESC:** this introduces a menu field specification which consists of four bytes:

MU.ESC code
Field number (1-127)
Field attributes:

Bit 7: 0 = protected, 1 = write enabled
Bits 5-3: foreground colour
Bits 2-0: background colour

Field length (0-255)

The field number is used to indicate the order of cursor movement around the menu, at least one field must be present and fields must form a consecutive sequence starting from one. The field length is converted to spaces within the window (in the background colour). Read-only menus must include a single zero-length field.

## 8.6 Procedure Entry Control Block

The procedure header block defined in 7.4 is read by IOSS during IOLOAD and the PC-relative entry point is converted to the absolute load address. This is passed to the caller in modified form in a 6-byte buffer:

| | | |
|---|---|---|
| PEENTRY | (.L) | Procedure entry point (absolute address) |
| PERAM | (.W) | Minimum RAM allocation for program |

PERAM contains the procedure's RAM requirement for PCB, stack and heap (in units of 1Kb) if it were to be invoked as a program by an OSSTART call. (If this is set to zero, the procedure cannot be run as a stand-alone program.)

Note that the address of a user-constructed PECB is required by a call to IODEFPRO.

## 8.7 Program List Element

The program list element is a buffer created on the calling program's heap when a parent program creates a child program with OSSTART. If the parent waits for the child to finish with an OSWAIT call, a pointer to the buffer will be returned when the child program terminates. The following fields are of interest to applications software:

| | | |
|---|---|---|
| PGRETURN | (.W) | return code |
| PGSTATUS | (.W) | status code |
| PGPARMS | (46) | return string |

This mechanism allows a child program to pass a system status code, an applications specific return code and any arbitrary text string back to its parent, to indicate completion status (or whatever).

Note that the child program passes these item in registers (see 7.9) which are placed in the program list element by the system for later examination by the parent. If the parent does not perform OSWAIT the contents of the program list element are undefined.

## 8.8 Standard Parameter String

The 68K/OS command program ADAM is the usual parent program for all stand-alone applications. To start a program, the user supplies the program name followed by any parameters that must be passed to it, delimited by spaces. This entire command line is passed to the child program in a record containing a set of text strings.

The set of text strings is preceded by a set of fields giving details of the overall length of the parameter record and the offsets from the start of the record to the program pathname string and the parameter strings (if any):

| | | |
|---|---|---|
| APLEN | (.W) | Length of parameter record |
| APNAME | (.W) | Offset to program pathname string |
| APPARM1 | (.W) | Offset to first parameter string |
| APPARM2 | (.W) | Offset to second parameter string |
| : | | |
| APPARMn | (.W) | Offset to nth parameter |

Following the nth parameter offset is the string containing the program pathname (as keyed) followed by the n parameter strings, each string being word-aligned.

Each string in the parameter record is in standard text string format, as defined in 8.9 below.

## 8.9 Standard Text String

All string parameters in system calls are standard string records. These consist of a word-aligned length field (.W) followed by the text characters, one per byte. Note that the length field defines the total number of characters only.

# APPENDIX A:

# I/O SUB-SYSTEM CALLS

## A.1 IOSS Register Conventions

The table following gives a quick summary of the use of registers on entry to and exit from the IOSS.

On entry, register D0 always contains the function code, which is the name of the routine. (A set of definitions for the values of these names is supplied in a parameter file). If IOSS is called with an invalid function code the status STINIOSS is returned.

Usage of parameters is discussed in complete detail for each routine in the following section.

All registers which are not shown in the table are preserved on exit and may have any value on entry (except that registers with defined system-wide usages follow the usual rules).

The register usage table uses the following coding scheme:

| Code | Length | Description |
|---|---|---|
| * | .L | preserved - all registers not shown are preserved |
| B | .L | buffer address |
| BL | .W | buffer length |
| C | .W | channel number |
| DB | .L | directory name buffer |
| DE | .L | directory entry buffer pointer |
| DI | .L | directory information buffer pointer |
| FP | .L | file position |
| MN | .L | magic number for directory scanning |
| NC | .W | number of bytes read or written |
| O | .B | option byte |
| P | .L | pathname |
| PE | .L | procedure entry information |
| PI | .W | procedure identifier |
| S | .W | status |
| St | .L | string |
| U | .L | unit number |
| XA | .L | device-dependent information, address |
| XD | .L | device-dependent information, data |
| YN | .B | yes/no answer |

IOSS Register Usage

| Function | On Entry | | | | | On Exit | |
|---|---|---|---|---|---|---|---|
| DO | D1 | D2 | D3 | AO | A1 | DO | D1 |
| IOSETDEF | | | O | St | | S | * |
| IOGETDEV | | | O | St | | S | * |
| IOGETPRE | | | O | St | | S | * |
| IOOPEN | | | O | P | | S | C |
| IOCLOSE | | | C | | | S | * |
| IOLOAD | | | O | P | PE | S | PI |
| IODEFPRO | | | O | P | PE | S | PI |
| IOUNLOAD | PI | | | | | S | * |
| IODELETE | | | O | P | | S | * |
| IORENAME | | | O | P | P | S | * |
| IOGETDIR | MN | | O | P | DE | S | MN |
| IOPUTDIR | | | O | P | DE | S | * |
| IODIRINF | | | O | P | DI | S | * |
| IOGETSEQ | BL | | C | B | | S | NC |
| IOGETRAN | BL | FP | C | B | | S | NC |
| IOGETLIN | BL | | C | B | | S | NC |
| IOPUTSEQ | NC | | C | B | | S | * |
| IOPUTRAN | NC | FP | C | B | | S | * |
| IOPUTLIN | NC | | C | B | | S | * |
| IOSETPOS | | FP | C | | | S | * |
| IOTRUNC | | | C | | | S | * |
| IOGETPOS | | | C | | | S | FP |
| IOEOF | | | C | | | S | YN |
| IOSIZE | | | C | | | S | FP |
| IOREADY | | | C | | | S | YN |
| IOMOUNT | U | | O | P | DB | S | * |
| IODISMOU | U | | O | P | | S | * |
| IOSPECIA | XD | | O | P | XA | S | XD |

| | | |
|---|---|---|
| ROUTINE | IOCLOSE - Close a Channel | |
| FUNCTION | To close an IOSS channel. | |
| INPUTS | D0.W | IOCLOSE |
| | D3.W | Channel number |
| OUTPUTS | D0.W | Status |
| STATUS CODES | STCHAN | Non-existent channel number |
| | STIOERR | I/O error on device |

SIDE EFFECTS

For sequential disk or microdrive files with a write access type component, all blocks currently slaved in memory are written out.

For all disk or microdrive files with a write access type component, the directory is updated and written out.

NOTES

The closing of a disk or microdrive file is a very complex operation as far as the system is concerned and may take a relatively long time, however, as far as the user program is concerned there are no complications of any interest.

| | | |
|---|---|---|
| ROUTINE | IODEFPRO - Define Procedure Entry Point | |
| FUNCTION | To define a procedure entry point without loading the procedure from a file. | |
| INPUTS | D0.W | IODEFPRO |
| | D3.B | Options byte |
| | A0.L | Address of the pathname string |
| | A1.L | Address of procedure entry control block |
| OUTPUTS | D0.W | Status |
| | D1.W | Procedure identifier |
| STATUS CODES | STAM | Access mode not allowed |
| | STAT | Illegal options byte or access type |
| | STBADDIR | Too many or few directory components in pathname |
| | STBADFIL | Missing or unwanted filename component |
| | STDEVICE | Unknown device |
| | STDEVSEQ | Device is sequential only |
| | STDIRECT | Directory operations not allowed |
| | STPMEM | Heap or stack overflow |
| | STPROC | Procedure name already defined |
| | STSMEM | Insufficient memory to perform IODEFPRO |
| | STSTRLEN | Invalid string length |
| | STSYNTAX | Syntax error |
| SIDE EFFECTS | None | |

NOTES

The procedure entry control block must be defined by the user and contains two fields:

| | |
|---|---|
| PEENTRY | (.L) Procedure entry point |
| PERAM | (.W) RAM requirement for procedure |

The pathname is only required to force through IOSS validation (as if an IOLOAD command were being processed) and is subsequently ignored. Any valid directory device pathname is suitable.

The options byte has one significant bit:

OPPROG    0 = data, 1 = program

| | | |
|---|---|---|
| ROUTINE | **IODELETE - Delete a File** | |
| FUNCTION | To delete the file defined by the pathname provided. | |
| INPUTS | DO.W | IODELETE |
| | D3.B | Options byte |
| | AO.L | Address of pathname string |
| OUTPUTS | DO.W | Status |
| STATUS CODES | STAT | Illegal options byte or access type |
| | STBADDIR | Too many or few directory components in pathname |
| | STBADFIL | Missing or unwanted filename component |
| | STDEVICE | Unknown device |
| | STDIRECT | Directory operations not allowed |
| | STEXIST | File does not exist |
| | STIOERR | I/O error on device |
| | STPMEM | Heap or stack overflow |
| | STSMEM | Insufficient memory to execute IODELETE |
| | STSTRLEN | Invalid string length |
| | STSYNTAX | Syntax error |
| | STUSE | File in use |
| SIDE EFFECTS | The directory will be read (if required), updated and flushed to disk or microdrive. | |
| NOTES | The options byte has one significant bit: | |
| | OPPROG | 0 = data, 1 = program |

| | | |
|---|---|---|
| ROUTINE | IODIRINF - **Fetch Directory Information** | |
| FUNCTION | To fetch information about an entire directory. | |
| INPUTS | DO.W | IODIRINF |
| | D3.B | Options byte |
| | AO.L | Address of pathname string |
| | A1.L | Address of directory data buffer (16 bytes) |
| OUTPUTS | DO.W | Status |
| STATUS CODES | STBADDIR | Too many or few directory components in pathname |
| | STBADFIL | Missing or unwanted filename component |
| | STDEVICE | Unknown device |
| | STDIRECT | Directory operations not allowed |
| | STIOERR | I/O error on device |
| | STPMEM | Heap or stack overflow |
| | STSMEM | Insufficient memory to perform IODIRINF |
| | STSTRLEN | Invalid string length |
| | STSYNTAX | Syntax error |
| SIDE EFFECTS | If the directory is not in memory, it is read in. | |

NOTES

The directory data buffer contains the following fields:

| | |
|---|---|
| DIENTRY | (.L) Size of directory in entries |
| DITOTAL | (.L) Total space of directory in Kb |
| DILEFT | (.L) Current space remaining in Kb |

The options byte has one significant bit:

OPPROG 0 = data, 1 = program

| | | |
|---|---|---|
| ROUTINE | **IODISMOUNT - Dismount a Directory** | |
| FUNCTION | To dismount a specified directory from its current unit or to dismount the directory mounted on a specified unit. | |
| INPUTS | DO.W | IODISMOU |
| | D1.L | Unit number |
| | D3.B | Options byte |
| | AO.L | Address of pathname string |
| CUTPUTS | DO.W | Status |
| STATUS CODES | STEXIST | Directory not found or not mounted |
| | STOPEN | Directory has open files |
| | STSYNTAX | Syntax error |
| | STUNIT | Unit number in use or invalid |
| SIDE EFFECTS | None | |

NOTES

If a directory component is included in the pathname the unit number is ignored and the named directory is dismounted, otherwise the directory mounted on the supplied unit number is dismounted.

Some device drivers will be capable of automatically dismounting the current directory when an IOMOUNT is requested for the same unit number (particularly useful for devices that have a maximum of one directory per unit).

The unit number is device dependent, being typically a small integer for a disk driver (though this must not be assumed by the user), and potentially some complex routing code for a network.

The options byte has one significant bit:

OPPROG    0 = data, 1 = program

| | | | |
|---|---|---|---|
| ROUTINE | IOEOF - **End-of-File Position Test** | | |
| FUNCTION | To determine whether the current file position is equal to the end-of-file position. | | |
| INPUTS | DO.W | IOEOF | |
| | D3.W | Channel number | |
| OUTPUTS | DO.W | Status | |
| | D1.B | Yes (non-zero) or no (zero) | |
| STATUS CODES | STCHAN | Invalid channel number | |
| SIDE EFFECTS | None | | |
| NOTES | For a sequential output file the answer is always yes. | | |

| | | |
|---|---|---|
| ROUTINE | **IOGETDEV - Fetch Default Device String** | |
| FUNCTION | To read either the default data or program device string into a user buffer. | |
| INPUTS | D0.W | IOGETDEV |
| | D3.B | Options byte |
| | A0.L | Address of default string buffer (minimum 11 bytes) |
| OUTPUTS | D0.W | Status |
| STATUS CODES | STPMEM | Heap or stack overflow |
| SIDE EFFECTS | None | |

NOTES

The string buffer must start on an even address. The buffer length is not (and cannot be) checked by the system. It is the user's responsibility to ensure that enough space is available.

The options byte has one significant bit:

OPPROG 0 = data, 1 = program

| | | |
|---|---|---|
| ROUTINE | **IOGETDIR - Read Directory Information** | |
| FUNCTION | Fetch information on a specified file or the next in a range of files to a user bufffer. | |
| INPUTS | D0.W | IOGETDIR |
| | D1.L | Magic number |
| | D3.B | Options byte |
| | A0.L | Address of pathname |
| | A1.L | Address of directory entry buffer (length 64 bytes) |
| OUTPUTS | D0.W | Status |
| | D1.L | Updated magic number |
| STATUS CODES | STBADDIR | Too many or few directory components in pathname |
| | STBADFIL | Missing or unwanted filename component |
| | STDEVICE | Unknown device |
| | STDIRECT | Directory operations not allowed |
| | STEXIST | File does not exist |
| | STIOERR | I/O error on device |
| | STPMEM | Heap or stack overflow |
| | STSMEM | Insufficient memory to perform IOGETDIR |
| | STSTRLEN | Invalid string length |
| | STSYNTAX | Syntax error |
| SIDE EFFECTS | If the directory block is not in memory, it will be read in. | |

NOTES

To search a range of filenames, wild card characters may be contained in the pathname.

The magic number is used by IOSS to determine its position during a range search. It must be set to zero for the first call of IOGETDIR and will then be updated automatically by subsequent calls of IOGETDIR. It must not be modified by the user program.

The options byte has one significant bit:

OPPROG 0 = data, 1 = program

| | | |
|---|---|---|
| ROUTINE | **IOGETLIN - Read a Line** | |
| FUNCTION | To read a line from the given channel into a user buffer of given length. | |
| INPUTS | D0.L | IOGETLIN |
| | D1.W | Buffer length |
| | D3.W | Channel number |
| | A0.L | Buffer address |
| OUTPUTS | D0.W | Status |
| | D1.W | Number of bytes read |
| STATUS CODES | STCHAN | Illegal channel number |
| | STEOF | End of file |
| | STGET | This channel cannot be read |
| | STIOERR | Hard I/O error |
| | STPART | Partial line has filled the buffer |

SIDE EFFECTS

In most cases an STEOF status will indicate that any further attempts to read sequentially from that channel will fail immediately with STEOF status and a zero byte count. However this effect is device specific, and some devices (notably the keyboard driver - KEY:) will permit further input while continuing to return STEOF on each call.

NOTES

A normal status from IOGETLIN indicates that a complete line was read into the user buffer including the terminating newline character. The byte count returned in D1 also includes the newline.

An STPART status from IOGETLIN indicates that the line was too long for the buffer supplied. In this case the newline is not placed in the buffer and the count is returned equal to the buffer length.

An STEOF status from IOGETLIN indicates that an end-of-file condition was encountered by the device driver (the cause is device specific). In this case the newline is not placed in the buffer and the count returned is the number of bytes read prior to the detection of end-of-file.

| | | |
|---|---|---|
| ROUTINE | IOGETPOS - Read the Current File Position | |
| FUNCTION | To read the current position pointer for the given channel. | |
| INPUTS | D0.W | IOGETPOS |
| | D3.W | Channel number |
| OUTPUTS | D0.W | Status |
| | D1.L | File position |
| STATUS CODES | STCHAN | Illegal channel number |
| SIDE EFFECTS | None | |
| NOTES | This call is valid with both sequential and random files, though with sequential output files the result is simply the end-of-file position. | |

| | | |
|---|---|---|
| ROUTINE | **IOGETPRE - Fetch Default Prefix String** | |
| FUNCTION | To read either the default data or program prefix string into a user buffer. | |
| INPUTS | D0.W | IOGETPRE |
| | D3.B | Options byte |
| | A0.L | Address of default string buffer (minimum 46 bytes) |
| OUTPUTS | D0.W | Status |
| STATUS CODES | STPMEM | Heap or stack overflow |
| SIDE EFFECTS | None | |

NOTES

The string buffer must start on an even address. The buffer length is not (and cannot be) checked by the system. It is the user's responsibility to ensure that enough space is available.

The options byte has one significant bit:

OPPROG    0 = data, 1 = program

| | | |
|---|---|---|
| ROUTINE | IOGETRAN - Read Random | |
| FUNCTION | To read a specified number of bytes from the given channel at a defined file position. | |
| INPUTS | D0.W | IOGETRAN |
| | D1.W | Buffer length |
| | D2.L | File position |
| | D3.W | Channel number |
| | A0.L | Buffer address |
| OUTPUTS | D0.W | Status |
| | D1.W | Number of bytes read |
| STATUS CODES | STCHAN | Invalid channel number |
| | STEOF | End-of-file detected |
| | STGET | This channel cannot be read |
| | STIOERR | Hard I/O error |
| | STSETPOS | Invalid file position |
| | STSEQ | Channel is open for sequential access only |
| SIDE EFFECTS | The current file position for the channel is updated by IOGETRAN. | |

NOTES

A normal status from IOGETRAN indicates that the number of bytes requested has been read into the user buffer. In this case D1 is equal to the entry value.

If a negative file position is requested, a status of STSETPOS is returned.

An STEOF status from IOGETRAN indicates that an end-of-file condition was detected during command execution and that a partial transfer of zero or more bytes was carried out, the byte count being held in D1. Note that if a transfer of N bytes is requested and there are N bytes remaining to be read then an STEOF status is not returned. Note also that a start file position greater than or equal to the current end-of-file position will result in an immediate STEOF status and a zero byte count.

| | | |
|---|---|---|
| ROUTINE | **IOGETSEQ - Read Sequential** | |
| FUNCTION | To read the specified number of bytes from the given channel. | |
| INPUTS | D0.W | IOGETSEQ |
| | D1.W | Buffer length |
| | D3.W | Channel number |
| | A0.L | Buffer address |
| OUTPUTS | D0.W | Status |
| | D1.W | Number of bytes read |
| STATUS CODES | STCHAN | Illegal channel number |
| | STEOF | End-of-file detected |
| | STGET | This channel cannot be read |
| | STIOERR | Hard I/O error |

SIDE EFFECTS

The file position pointer is maintained automatically during sequential file access and need not be of concern to the user (unless he is also using random access on the same channel).

NOTES

A normal status from IOGETSEQ indicates that the number of bytes requested has been read into the user buffer. In this case D1 is equal to the entry value.

An STEOF status from IOGETSEQ indicates that an end-of-file condition was detected during command execution and that a partial transfer of zero or more bytes was carried out, the byte count being held in D1. Note that if a transfer of N bytes is requested and there are N bytes remaining to be read then an STEOF status is not returned.

It is permissible to mix sequential and random reads from the same channel provided only that the file was opened for random access. In this case the position of the file pointer used by the IOGETSEQ call can be manipulated directly by calls to IOGETRAN and IOSETPOS.

| | | |
|---|---|---|
| ROUTINE | IOLOAD - Load a Procedure into RAM | |
| FUNCTION | Load a reentrant procedure into RAM if it is not already present. | |
| INPUTS | D0.W | IOLOAD |
| | D3.B | Options byte |
| | A0.L | Address of pathname string |
| | A1.L | Address of procedure entry control block (6 bytes) |
| OUTPUTS | D0.W | Status |
| | D1.W | Procedure identifier |
| STATUS CODES | STAM | Access mode not allowed |
| | STAT | Illegal options byte or access type |
| | STBADDIR | Too many or few directory components in pathname |
| | STBADFIL | Missing or unwanted filename component |
| | STDEVICE | Unknown device |
| | STDEVSEQ | Device is sequential only |
| | STDIRECT | Directory operations not allowed |
| | STEXIST | File does not exist |
| | STIOERR | I/O error on device |
| | STNOFILE | No room left in the Open Files List |
| | STNOSHAR | Device cannot be shared and is in use |
| | STPMEM | Heap or stack overflow |
| | STSMEM | Insufficient memory to perform IOLOAD |
| | STSTRLEN | Invalid string length |
| | STSYNTAX | Syntax error |
| | STUSE | File in use |

SIDE EFFECTS

A procedure list element is created using space grabbed from the user program's heap.

NOTES

68K/OS procedures must be reentrant and position independent, thus if a copy of the procedure is already loaded, it need not be fetched from the device.

If the procedure cannot be loaded, the PEENTRY field will be set up to point to an abort routine within IOSS. If this is called, the calling program will be aborted via TRFINISH with an STNOLOAD status.

The options byte has one significant bit:

OPPROG 0 = data, 1 = program

| | | |
|---|---|---|
| ROUTINE | **IOMOUNT - Mount a Directory** | |
| FUNCTION | To mount an unspecified directory on a given device unit or to enable the use of the specified directory on the given device unit. | |
| INPUTS | D0.W | IOMOUNT |
| | D1.L | Unit number |
| | D3.B | Options byte |
| | A0.L | Address of pathname string |
| | A1.L | Address of directory name buffer (46 bytes) |
| OUTPUTS | D0.W | Status |
| STATUS CODES | STDIRECT | Directory operations not allowed |
| | STEXIST | Specified directory not found |
| | STIOERR | Hard I/O error |
| | STMOUNT | Directory already mounted |
| | STOPEN | Current directory contains open files |
| | STSYNTAX | Syntax error |
| | STUNIT | Unit number in use or invalid |

SIDE EFFECTS

When a directory is mounted, the first directory block is read into memory and will remain slaved in until flushed by some other I/O or memory management operation.

Some device drivers will be capable of automatically dismounting the current directory when an IOMOUNT is requested for the same unit number (particularly useful for devices that have a maximum of one directory per unit). In this case an STOPEN status can be returned if there are open files on the directory to be dismounted.

NOTES

The unit number is device dependent, being typically a small integer for a disk driver (though this must not be assumed by the user), and potentially some complex routing code for a network.

If a directory name is specified in the pathname then IOMOUNT will check that the directory found matches the one supplied, otherwise this check is omitted and any directory found is mounted, its name being returned in the user buffer supplied.

The options byte has one significant bit:

OPPROG    0 = data, 1 = program

| | | |
|---|---|---|
| ROUTINE | IOOPEN - Open a Channel | |
| FUNCTION | To create a channel for the transfer of data between the calling program and the supplied device or file pathname. | |
| INPUTS | D0.W | IOOPEN |
| | D3.B | Options byte |
| | A0.L | Address of pathname string |
| OUTPUTS | D0.W | Status |
| | D1.W | Channel number |
| STATUS CODES | STAM | Access mode not allowed |
| | STAT | Illegal options byte or access type |
| | STATSEQ | Cannot read and write sequential simultaneously |
| | STBADDIR | Too many or few directory components in pathname |
| | STBADFIL | Missing or unwanted filename component |
| | STDEVICE | Unknown device |
| | STDEVSEQ | Device is sequential only |
| | STEXIST | File does not exist |
| | STIOERR | I/O error on device |
| | STNOFILE | No room left in the Open Files List |
| | STNOSHAR | Device cannot be shared and is in use |
| | STPMEM | Heap or stack overflow |
| | STSMEM | Insufficient memory to perform IOOPEN |
| | STSTRLEN | Invalid string length |
| | STSYNTAX | Syntax error |
| | STUSE | File in use |

SIDE EFFECTS

If a non-existent file is opened with a write access type component, then a file of zero length is created.

NOTES

The options byte contains 5 significant bits:

| | |
|---|---|
| OPREAD | 0 = read disabled, 1 = read enabled |
| OPWRITE | 0 = write disabled, 1 = write enabled |
| OPRAN | 0 = sequential access, 1 = random access |
| OPPROG | 0 = data, 1 = program |
| OPRDAHED | 0 = unbuffered, 1 = read ahead/write behind |

The OPRDAHED option will provide system generated double buffering on random or sequential file I/O to certain file structured devices. Those device drivers that support this facility will automatically perform read ahead and write behind operations on memory block sized units of the file.

ROUTINE — **IOPUTDIR - Update Directory Information**

FUNCTION — To update certain fields in the directory information for the file defined by the given pathname.

INPUTS

| | | |
|---|---|---|
| D0.W | IOPUTDIR | |
| D3.B | Options byte | |
| A0.L | Address of pathname string | |
| A1.L | Address of directory entry buffer | |

OUTPUTS

| | |
|---|---|
| D0.W | Status |

STATUS CODES

| | |
|---|---|
| STAT | Illegal options byte or directory write protected |
| STBADDIR | Too many or few directory components in pathname |
| STBADFIL | Missing or unwanted filename component |
| STDEVICE | Unknown device |
| STDIRECT | Directory operations not allowed |
| STEXIST | File does not exist |
| STIOERR | I/O error on device |
| STPMEM | Heap or stack overflow |
| STSMEM | Insufficient memory to perform IOPUTDIR |
| STSTRLEN | Invalid string length |
| STSYNTAX | Syntax error |

SIDE EFFECTS — The directory will be read (if required), updated and flushed to disk or microdrive.

NOTES — Only the following fields within the directory entry buffer can be updated:

| | |
|---|---|
| DEATFILE | (.B) File access type |
| DECOMM | (25) String holding user comment |

changes to other fields are ignored.

It is safe to call IOPUTDIR to change the directory entry of a file that is currently open.

IOPUTDIR will normally be called to update the directory entry buffer fetched by a call to IOGETDIR (the register usage is defined to make this simple), however, the user can construct his own directory entry buffer if required.

The options byte has one significant bit:

OPPROG 0 = data, 1 = program

ROUTINE IOPUTLIN - Write Line

FUNCTION To write a line (length not exceeding the specified number of bytes) to the given channel number.

| | | |
|---|---|---|
| INPUTS | DO.W | IOPUTLIN |
| | D1.W | Byte count |
| | D3.W | Channel number |
| | AO.W | Buffer address |
| CUTPUTS | DO.W | Status |
| STATUS CODES | STCHAN | Invalid channel number |
| | STDIRFUL | Directory full |
| | STIOERR | Hard I/O error |
| | STPART | Partial line written |
| | STPUT | Cannot write to this channel |

SIDE EFFECTS When using IOPUTLIN with sequential access or writing past the end-of-file with random access, the end-of-file pointer is automatically updated.

NOTES Bytes are output to the channel until either a newline character is sent (normal status) or the count in D1 has been exhausted (STPART status).

Calls to IOPUTSEQ, IOPUTRAN and IOPUTLIN can be mixed on a single channel provided that the file has been opened for random access. The data is written starting from the current file position pointer and this is advanced as usual. The end-of-file pointer is updated only when the file is extended (it is a high water mark).

| | | |
|---|---|---|
| ROUTINE | **IOPUTRAN - Write Random** | |
| FUNCTION | To write the specified number of bytes to the given file at the defined position. | |
| INPUTS | D0.W | IOPUTRAN |
| | D1.W | Byte count |
| | D2.W | File position |
| | D3.W | Channel number |
| | A0.W | Buffer address |
| OUTPUTS | D0.W | Status |
| STATUS CODES | STCHAN | Invalid channel number |
| | STDIRFUL | Directory full |
| | STIOERR | Hard I/O error |
| | STPUT | Cannot write to this channel |
| | STSETPOS | Invalid position |
| | STSEQ | Sequential access only |

SIDE EFFECTS

If the file position is greater than the current end-of-file position, the file will be extended prior to writing, nulls being written between the old end-of-file and the starting file position.

NOTES

Calls to IOPUTSEQ, IOPUTRAN and IOPUTLIN can be mixed on a single channel provided that the file has been opened for random access. The data is written starting from the current file position pointer and this is advanced as usual. The end-of-file pointer is updated only when the file is extended (it is a high water mark).

| | | |
|---|---|---|
| ROUTINE | IOPUTSEQ - **Write Sequential** | |
| FUNCTION | To write the specified number of bytes to the given channel number. | |
| INPUTS | D0.W | IOPUTSEQ |
| | D1.W | Byte count |
| | D3.W | Channel number |
| | A0.W | Buffer address |
| OUTPUTS | D0.W | Status |
| STATUS CODES | STCHAN | Invalid channel number |
| | STDIRFUL | Directory full |
| | STIOERR | Hard I/O error |
| | STPUT | Cannot write to this channel |

SIDE EFFECTS

When using IOPUTSEQ with sequential access or writing past the end-of-file with random access, the end-of-file pointer is automatically updated.

NOTES

Calls to IOPUTSEQ, IOPUTRAN and IOPUTLIN can be mixed on a single channel provided that the file has been opened for random access. The data is written starting from the current file position pointer and this is advanced as usual. The end-of-file pointer is updated only when the file is extended (it is a high water mark).

| | | |
|---|---|---|
| ROUTINE | IOREADY - Poll an Input Channel | |
| FUNCTION | To determine whether there is input pending on the specified channel number. | |
| INPUTS | D0.W | IOREADY |
| | D3.W | Channel number |
| OUTPUTS | D0.W | Status |
| | D1.B | Yes (non-zero) or no (zero) |
| STATUS CODES | STCHAN | Invalid channel number |
| | STGET | Not an input channel |
| SIDE EFFECTS | None | |

NOTES

If the input is received from an asynchronous device with input arriving outside the control of the operating system, a yes answer is returned if a call to read a single byte would be satisfied immediately, otherwise a no answer is returned. Typical devices are the keyboard, RS232 input and pipes.

If the input is received from a synchronous device such as a disk or microdrive file, the answer is no if the channel is positioned at end-of-file, otherwise yes.

| | | |
|---|---|---|
| ROUTINE | **IORENAME - Rename a File** | |
| FUNCTION | To rename the file defined by the old pathname to that given by the new pathname. | |
| INPUTS | D0.W | IORENAME |
| | D3.B | Options byte |
| | A0.L | Old pathname |
| | A1.L | New pathnamme |
| OUTPUTS | D0.W | Status |
| STATUS CODES | STAT | Illegal options byte or access type |
| | STBADDIR | Too many or few directory components in pathname |
| | STBADFIL | Missing or unwanted filename component |
| | STDIRECT | Directory operations not allowed |
| | STDEVICE | Unknown device |
| | STEXIST | File does not exist |
| | STIOERR | I/O error on device |
| | STPMEM | Heap or stack overflow |
| | STRENAME | Incompatible pathnames |
| | STSMEM | Insufficient memory to perform IORENAME |
| | STSTRLEN | Invalid string length |
| | STSYNTAX | Syntax error |
| | STUSE | File in use |
| SIDE EFFECTS | The directory will be read (if required), updated and flushed to disk or microdrive. | |
| NOTES | Except for the filenames, the old and new pathnames must be identical. | |

The options byte has one significant bit:

OPPROG    0 = data, 1 = program

| | | |
|---|---|---|
| ROUTINE | **IOSETDEF - Set Default Pathname String** | |
| FUNCTION | To set either the default data or program pathname string for the calling program. | |
| INPUTS | D0.W | IOSETDEF |
| | D3.B | Options byte |
| | A0.L | Address of default string (maximum 44 characters) |
| OUTPUTS | D0.W | Status |
| STATUS CODES | STPMEM | Heap or stack overflow |
| | STSTRLEN | String length invalid |
| | STSYNTAX | Syntax error |

SIDE EFFECTS

The device and/or directory components of the pathname specified will be used to replace any respective null components in any subsequent IOSS calls (by the calling program only).

NOTES

If the pathname consists of a null string, no action is taken and the current defaults are retained.

If the pathname consists of a device name, then this becomes the new default device and the default prefix is cleared.

If the pathname consists of one or more directory components, then these become the new default prefix and the current default device remains unchanged.

If the pathname consists of a device name followed by one or more directory components, then both the default device and the default prefix strings are updated as specified.

The supplied pathname string must start on an even address.

The options byte has one significant bit:

OPPROG    0 = data, 1 = program

| | | |
|---|---|---|
| ROUTINE | IOSETPOS - Set the Current Position Pointer | |
| FUNCTION | To define the position in a random file at which the next write operation will start. | |
| INPUTS | D0.W | IOSETPOS |
| | D2.W | File position |
| | D3.W | Channel number |
| OUTPUTS | D0.W | Status |
| STATUS CODES | STCHAN | Invalid channel number |
| | STDIRFUL | Directory full |
| | STIOERR | Hard I/O error |
| | STSEQ | Cannot perform IOSETPOS on sequential file |
| | STSETPOS | Invalid file position |
| SIDE EFFECTS | If the file is open for random writing and the new position is greater than the current end-of-file position, then the file is extended with null bytes to the new position. | |
| NOTES | If the file pointer is negative, an STSETPOS status is returned. | |

| | |
|---|---|
| ROUTINE | **IOSIZE - Determine File Size** |
| FUNCTION | To determine the size of the file accessed by the given channel number. |
| INPUTS | D0.W IOSIZE<br>D3.W Channel number |
| OUTPUTS | D0.W Status<br>D1.L File size (bytes) |
| STATUS CODES | STCHAN Invalid channel number<br>STNOSIZE No size information available |
| SIDE EFFECTS | None |
| NOTES | For sequential output channels the size is equal to the current position.<br><br>Status code STNOSIZE is returned for devices that do not maintain end-of-file position, such as the keyboard. |

| | | |
|---|---|---|
| ROUTINE | IOSPECIA - Device Specific Operation | |
| FUNCTION | To perform one or more device specific operations as specified by the device dependent parameters. | |
| INPUTS | D0.W | IOSPECIA |
| | D1.L | Device dependent parameter |
| | D3.B | Options byte |
| | A0.L | Address of pathname string |
| | A1.L | Device dependent parameter |
| OUTPUTS | D0.W | Status |
| | D1.L | Device dependent result |
| STATUS CODES | STSPECIA | Not allowed on this device |
| SIDE EFFECTS | Device specific. | |
| NOTES | Status codes will be device specific. | |
| | The options byte has one significant bit: | |
| | OPPROG | 0 = data, 1 = program |

| | | |
|---|---|---|
| ROUTINE | **IOTRUNC - Set End-of-File Pointer to Current Position** | |
| FUNCTION | To truncate a file by setting the end-of-file pointer equal to the current position pointer. | |
| INPUTS | D0.W | IOTRUNC |
| | D3.W | Channel number |
| OUTPUTS | D0.W | Status |
| STATUS CODES | STCHAN | Invalid channel number |
| | STIOERR | Hard I/O error |
| | STPUT | Cannot write to this channel |
| | STSEQ | Sequential access only |
| SIDE EFFECTS | Any disk or microdrive blocks released by file truncation are marked as free in some device dependent manner. | |
| NOTES | IOTRUNC is not available in sequential access mode since, by definition, the current position pointer is always at the end-of-file. | |

| | |
|---|---|
| ROUTINE | **IOUNLOAD - Release a Procedure** |
| FUNCTION | To decrement the use count for the given procedure and release its memory if the use count drops to zero. |
| INPUTS | D0.W IOUNLOAD<br>D1.W Procedure identifier |
| OUTPUTS | D0.W Status |
| STATUS CODES | STINPROC Invalid procedure identifier |
| SIDE EFFECTS | The procedure list element is returned to the user's heap.<br><br>If the procedure's use count drops to zero, the memory it uses is released to the system and the procedure table entry is deleted. |
| NOTES | None |

# APPENDIX B:

# OPERATING SYSTEM CALLS

| | | |
|---|---|---|
| ROUTINE | **OSAVAIL - Determine the Free Stack/Heap Space** | |
| FUNCTION | To determine how much stack/heap space is still free for the calling program. | |
| INPUTS | D0.W | OSAVAIL |
| OUTPUTS | D0.W | Status |
| | D1.L | Size of largest heap record |
| | D2.L | Total heap space available |
| | D3.L | Space between heap and stack |
| STATUS CODES | 0 | Always returns success status |
| SIDE EFFECTS | None | |
| NOTES | The maximum sized heap record that can be allocated is the greater of D1 and D3. | |

| | | |
|---|---|---|
| ROUTINE | **OSBINCLK - Read Date and Time in Binary** | |
| FUNCTION | To read the date and time in binary from the internal system clock. | |
| INPUTS | DO.W | OSBINCLK |
| OUTPUTS | DO.W<br>D1.L | Status<br>Binary internal clock value |
| STATUS CODES | 0 | Always returns success status |
| SIDE EFFECTS | None | |
| NOTES | The value returned is defined to be the number of seconds that have elapsed since 00:00:00 am on 1st January 1983. Whether the returned value is sensible depends on a correct call to OSSETCLK to initialise the clock.<br><br>If the hardware does not support a clock, a value of zero is returned. | |

ROUTINE — OSDELAY - Delay for a Number of Clock Ticks

FUNCTION — To suspend the calling program for a specified number of system clock ticks.

INPUTS

| | | |
|---|---|---|
| DO.W | OSDELAY |
| D1.W | Number of clock ticks |

OUTPUTS

| | | |
|---|---|---|
| DO.W | Status |

STATUS CODES

| | | |
|---|---|---|
| 0 | Always returns success status |

SIDE EFFECTS — All programs with priority lower than the caller will tend to speed up for the duration of the delay.

NOTES — The program performs a passive wait for the duration of the delay period and consumes no system time resources (except the minimal overhead of handling a clock queue entry).

The clock frequency is 50 or 60Hz, depending on the hardware clock rate.

If the delay requested is zero, one or a negative number of clock ticks, the program will be suspended until the next clock tick.

The timing should not be relied upon for great accuracy, particularly when the system is heavily loaded. The precise timing will depend on when (in the clock cycle) the call was made, how many clock interrupts were ignored because of heavy system loading and how long it takes before the scheduler is able to restart the program. Even in an 'idle' system, a request for an N clock tick delay will produce a delay of between N-1 and N clock ticks. To ensure a delay of at least N clock ticks, N+1 should be requested.

If delays are required for periods shorter than one tick or must be accurate to within tens of microseconds the user should perform an active wait with routine SPACTIVE in supervisor mode with interrupts disabled.

| | | |
|---|---|---|
| ROUTINE | OSHEAPAL - Allocate a Heap Record | |
| FUNCTION | To grab a record of specified size from the program's heap. | |
| INPUTS | DO.W | OSHEAPAL |
| | D1.L | Size of record required |
| OUTPUTS | DO.W | Status |
| | AO.L | Address of heap record allocated |
| STATUS CODES | STPMEM | Insufficient heap space |
| SIDE EFFECTS | Too many requests for small heap records which are released in random order are liable to cause heap fragmentation. It is usually better practice to grab heap space in a more structured manner. | |
| NOTES | Heap is grabbed with a first fit algorithm, which is fairly fast, but can lead to fragmentation with undisciplined use. | |

| | | |
|---|---|---|
| ROUTINE | **OSHEAPDE - Release a Heap Record** | |
| FUNCTION | To release a heap record to the calling program's heap. | |
| INPUTS | DO.W | OSHEAPDE |
| | AO.L | Address of heap record |
| OUTPUTS | DO.W | Status |
| STATUS CODES | 0 | Always returns success status |
| SIDE EFFECTS | None | |
| NOTES | The record being returned to the heap is coalesced with adjacent free records if possible. | |
| | If the address of the heap record supplied lies outside of the heap boundaries, the command is ignored. | |

| | |
|---|---|
| ROUTINE | **OSKILL - Force Termination of a Child Program** |
| FUNCTION | To cause the specified child program to be terminated. |
| INPUTS | DO.W OSKILL<br>D1.L Program identifier |
| OUTPUTS | DO.W Status |
| STATUS CODES | STSTOP Child was already stopped<br>STINVAL Invalid program identifier |
| SIDE EFFECTS | The child program is terminated by forcing it to execute a TRAP #0 instructon with the resulting side effects. |
| NOTES | When the child program terminates it returns the following data to the parent in the program list element:<br><br>PGRETURN (.W) -1<br>PGSTATUS (.W) STKILLED<br>PGPARMS (46) null<br><br>Prior to termination the child program is allowed to finish any critical system code that it is executing, its PC is then modified to divert it to a TRAP 0 instruction which performs the STFINISH. It will then terminate in the same way as if it had called STFINISH voluntarily. |

| | |
|---|---|
| ROUTINE | **OSMEMALL - Allocate RAM Memory to a Program** |
| FUNCTION | To allocate a specified number of contiguous 1Kb RAM memory blocks to the calling program. |
| INPUTS | D0.W OSMEMALL<br>D1.L Ownerhip information<br>D2.W Number of 1Kb blocks wanted |
| OUTPUTS | D0.W Status code<br>A0.L Base address of allocated memory |
| STATUS CODES | STSMEM Insufficient memory available |
| SIDE EFFECTS | The memory manager will attempt to allocate a contiguous area of memory using a cyclic first fit method. If there are insufficient contiguous free blocks the memory manager first attempts to release slaved blocks that are up-to-date on disk or microdrive (avoiding data transfers). Failing this it will force slaved blocks to disk or microdrive until there is sufficient contiguous memory or all slaved blocks have been released. |
| NOTES | The value of D1 on entry should normally be the address of the calling program's Program Control Block since this ensures that the memory will be deallocated automatically when the program is terminated.<br><br>If the program wishes to use memory in a non-standard way, D1 may contain any value that the user program requires, but this value must be remembered by the user program or passed to any child program that is inheriting the memory, to enable its subsequent release. |

| | | |
|---|---|---|
| ROUTINE | **OSMEMDA - Deallocate Memory by Ownership Information** | |
| FUNCTION | To deallocate all memory blocks that are allocated to the specified owner. | |
| INPUTS | DO.W | OSMEMDA |
| | D1.L | Ownership information |
| OUTPUTS | DO.L | Status |
| STATUS CODES | 0 | Always returns success status |
| SIDE EFFECTS | None | |
| NOTES | This is the recommended function for the release of memory that has a non-standard ownership value. All memory blocks allocated to the specified owner are marked as free. | |

| | | |
|---|---|---|
| ROUTINE | **OSMEMDS - Deallocate Memory by Address Range** | |
| FUNCTION | To deallocate the specified number of 1Kb blocks starting at the address provided. | |
| INPUTS | D0.W | OSMEMDS |
| | D2.W | Number of 1Kb blocks to deallocate |
| | A0.L | Base address of memory to be deallocated |
| OUTPUTS | D0.W | Status |
| STATUS CODES | 0 | Always returns success status |
| SIDE EFFECTS | If the parameters are incorrect it is possible to deallocate another program's memory, with drastic side effects. | |
| NOTES | This is an alternative function to OSMEMDA to deallocate memory allocated with OSMEMALL or OSMEMOWN. | |

| | | |
|---|---|---|
| ROUTINE | **OSMEMOWN - Change Memory Ownership Information** | |
| FUNCTION | To change the owner information associated with a specified range of contiguous blocks. | |
| INPUTS | D0.W | OSMEMOWN |
| | D1.L | New ownership information |
| | D2.W | Number of blocks to be affected |
| | A0.L | Base address of allocated memory |
| OUTPUTS | D0.L | Status code |
| STATUS CODES | 0 | Always returns success status |
| SIDE EFFECTS | This command is capable of changing the memory allocation of the entire system and thus should be used with great care, to avoid some of the more drastic side effects, the most likely being memory remaining in use after the calling program is terminated. | |
| NOTES | This call can be used to pass memory blocks to a child program from its parent, in which case it is recommended that the address of the child program's Program Control Block is held in D1, to ensure automatic deallocation when the child program is terminated. | |

| | |
|---|---|
| ROUTINE | **OSMENDIS - Display Fixed Menu Data and Initialise Fields** |

FUNCTION — To clear the display file, copy the fixed menu data into it and initialise the variable data fields.

INPUTS

| | |
|---|---|
| DO.W | OSMENDIS |
| AO.L | Menu display file address |
| A1.L | Menu variable data buffer address |
| A2.L | Menu fixed data buffer address |

OUTPUTS

| | |
|---|---|
| DO.W | Status |

STATUS CODES

| | |
|---|---|
| STMURAM | Variable data space insufficient |
| STSYNTAX | Syntax error in fixed menu data |

SIDE EFFECTS — This command displays a menu on the screen using standard DFM calls with their associated side effects.

The variable data buffer is split up into fields (as defined in the fixed menu data) each of which contains a string whose length word is set to zero.

NOTES — The variable data buffer must be large enough to hold all the variable fields. This entire buffer is a string whose length word must be intitialised by the user and must lie on a word boundary. No other initialisation is required.

The effect of this command is to provide all the intial-isation required to enable successive calls to OSMENRD to fetch the user's input to the variable fields. However, to call OSMENRD, at least one variable field must be present.

| | | |
|---|---|---|
| ROUTINE | **OSMENGET - Read Menu Field from Variable Data Structure** | |
| FUNCTION | To read a specified field from the menu variable data structure into the supplied buffer. | |
| INPUTS | D0.W | OSMENGET |
| | A1.L | Menu variable data structure address |
| | A2.B | Field number |
| | A3.L | Buffer address |
| OUTPUTS | D0.W | Status |
| | D1.B | Field attributes |
| STATUS CODES | STMUFLD | Invalid menu field number |
| SIDE EFFECTS | None | |
| NOTES | The screen menu need not be displayed when calling OSMENGET, the menu variable data structure being sufficient. This enables the variable data structure to be used as a parameter passing mechanism between programs. | |
| | The buffer must be large enough to accept the entire contents of the field in string format. | |

| | | |
|---|---|---|
| ROUTINE | OSMENPUT - Redisplay a Variable Field | |
| FUNCTION | To update a menu variable field on the screen and in the variable data structure. | |
| INPUTS | D0.W | OSMENPUT |
| | D1.B | Field attributes |
| | D2.B | Field number |
| | A0.L | Menu display file address |
| | A1.L | Menu variable data structure address |
| | A3.L | Address of string containing new field contents |
| OUTPUTS | D0.W | Status |
| STATUS CODES | STMUFLD | Invalid menu field number |
| | STSTRLEN | String too long for menu field |
| SIDE EFFECTS | The screen is updated with standard DFM calls with their associated side effects. | |
| NOTES | This call can be used to display a prompt or error message in a particular field of the menu and to change the field attributes associated with a given field. | |

ROUTINE **OSMENRD - Read User Input to Menu**

FUNCTION To handle all aspects of keyboard input, screen output and data capture associated with user interaction with a screen menu.

INPUTS

| | | |
|---|---|---|
| | D0.W | OSMENRD |
| | D1.W | Keyboard channel number |
| | D2.B | Field number at which to position cursor |
| | D3.W | Allowable function codes |
| | A0.L | Menu display file address |
| | A1.L | Menu variable data buffer address |
| | A2.L | Option list display file address |

OUTPUTS

| | | |
|---|---|---|
| | D0.W | Status |
| | D1.B | Terminating function code |
| | D2.B | Field number at which cursor is positioned |

STATUS CODES STMUFLD Invalid menu field number

SIDE EFFECTS This command updates the screen using standard DFM calls and reads the keyboard using standard IOSS calls, with their associated side effects.

NOTES

OSMENRD accepts keyboard input and will output characters to the screen in the current field, handling the specific line imaging functions available on the keyboard (usually delete character and delete field). Forward and backward tab functions are used to move the cursor between fields. (Note that any scrolling required is performed automatically by DFM.)

The option list display file may be omitted by setting A2 to zero. This display file contains a list of options (one per line) which may be selected by using the vertical cursor movement keys and the ESCAPE key to copy it into the current menu field. (Scrolling is again handled by DFM.)

User input to the menu is terminated by one of up to sixteen function codes as defined by set bits in positions 0-15 of D3 representing the RETURN (or NEWLINE or ENTER) key and function keys F1 to F15 respectively. Where there are less than fifteen function keys, these codes may be generated by some implementation specific combination of SHIFT and CONTROL functions.

The fields in the menu variable data structure are updated by OSMENRD and are read and modified by OSMENGET and OSMENPUT.

| | | |
|---|---|---|
| ROUTINE | OSSETCLK - Set Date and Time in Binary | |
| FUNCTION | To set the hardware date and time clock to the specified (binary) value. | |
| INPUTS | DO.W | OSSETCLK |
| | D1.L | Date and time |
| OUTPUTS | DO.W | Status |
| STATUS CODES | 0 | Always returns success status |
| SIDE EFFECTS | None | |
| NOTES | The value is defined to be the number of seconds that have elapsed since 00:00:00 am on 1st January 1983. If the hardware does not support a clock, the command is ignored. | |

| | | |
|---|---|---|
| ROUTINE | **OSSTART - Load and Start a Program** | |
| FUNCTION | To create the program data structures, obtain the specified RAM for the program's stack and heap, load the named procedure if it is not already present and start the program in the defined state. | |
| INPUTS | D0.W | OSSTART |
| | D1.W | RAM size required |
| | D2.B | Priority relative to the calling program |
| | D3.B | Data or program default indicator (see IOSETDEF) |
| | D4.B | Program state (0 = ready, otherwise suspended) |
| | A0.L | Address of pathname string |
| | A1.L | Address of parameters string |
| OUTPUTS | D0.W | Status |
| | D1.L | Program identifier |
| STATUS CODES | STAM | Access mode not allowed |
| | STAT | Access type not allowed |
| | STBADDIR | Too many or few directory components in pathname |
| | STBADFIL | Missing or unwanted filename component |
| | STDEVICE | Unknown device |
| | STDEVSEQ | Device is sequential only |
| | STDIRECT | Directory operations not allowed |
| | STEXIST | File does not exist |
| | STIOERR | I/O error on device |
| | STNOFILE | No room left in the Open Files List |
| | STNOSHAR | Device cannot be shared and is in use |
| | STPMEM | Heap or stack overflow |
| | STSMEM | Insufficient memory to perform OSSTART |
| | STSTRLEN | Invalid string length |
| | STSYNTAX | Syntax error |
| | STUSE | File in use |

SIDE EFFECTS

OSSTART calls IOGETDIR, OSMEMALL and IOLOAD and exhibits their side effects. Also a program list element is created on the user's heap.

NOTES

RAM is grabbed by IOLOAD for the procedure code if it is not already loaded. RAM grabbed by OSMEMALL for the program's stack and heap is defined in Kb in D1 and/or in the procedure entry control block, the greater of these values being used.

The bottom three bits of D2 are subtracted from the calling program's priority to give the child's priority.

A1 points to a string in which free format parameters to the child program may be passed.

| | | |
|---|---|---|
| ROUTINE | **OSSTATUS - Determine Program Status** | |
| FUNCTION | Do determine whether a child program is still running. | |
| INPUTS | DO.W | OSSTATUS |
| | D1.L | Program identifier |
| OUTPUTS | DO.W | Status |
| STATUS CODES | STSTOP | Child program has stopped |
| | STINPROG | Invalid program identifier |
| SIDE EFFECTS | None | |
| NOTES | A zero status indicates that the child program is still running. | |

This call enables a program to poll a child program to determine completion. If a program must wait for a child to finish, it is more efficient to use OSWAIT.

| | | |
|---|---|---|
| ROUTINE | **OSTRAP - Define User Trap Routine** | |
| FUNCTION | To define an exception trap vector to the specified address for the given trap number. | |
| INPUTS | DO.W | OSTRAP |
| | D1.W | Name of exception trap |
| | AO.L | Address of user trap routine |
| OUTPUTS | DO.W | Status |
| STATUS CODES | STEXCEPT | Invalid exception routine |
| | STINTRAP | Invalid trap routine address |
| SIDE EFFECTS | None | |

NOTES

The following exceptions can be vectored to user routines:

| | |
|---|---|
| EAADDRES | Odd address |
| EAILLEGA | Illegal instruction |
| EADIVIDE | Divide by zero |
| EACHKINS | Array bound violation |
| EATRAPV | Arithmetic overflow |
| EAPRIV | Privileged instruction |
| EATRACE | Trace mode exception |
| EAALINE | A-line exception |
| EAFLINE | F-line exception |
| EATRAP4 | User trap 4 |
| EATRAP5 | User trap 5 |
| EATRAP6 | User trap 6 |
| EATRAP7 | User trap 7 |
| EATRAP8 | User trap 8 |
| EATRAP9 | User trap 9 |
| EATRAP10 | User trap 10 |
| EATRAP11 | User trap 11 |
| EATRAP12 | User trap 12 |
| EATRAP13 | User trap 13 |
| EATRAP14 | User trap 14 |
| EATRAP15 | User trap 15 |

The user program may call OSTRAP as often as it likes for the same exception, subsequent calls overwriting the previous exception vector with the new one.

To disable a user trap routine, AO should contain zero, but use of this facility may make it impossible to run the program with a debugger (which may wish to handle exceptions in a special way).

| | | |
|---|---|---|
| ROUTINE | **OSWAIT - Wait for a Child Program to Finish** | |
| FUNCTION | To suspend the calling program until the specified child program has finished. | |
| INPUTS | DO.W | OSWAIT |
| | D1.L | Program identifier |
| OUTPUTS | DO.W | Status code |
| | AO.L | Address of program list element |
| STATUS CODES | STINPROG | Invalid program identifier |
| SIDE EFFECTS | The calling program is suspended. | |

NOTES

The program list element contains the following fields of interest to the user:

```
PGRETURN  (.W) Return code
PGSTATUS  (.W) Status code
PGPARMS   (46) Return string
```

These provide the calling program with information concerning the termination of the child program.

When the parent program has finished with its child's program list element it should dispose of it using OSHEAPDE.

# APPENDIX C:

# DISPLAY FILE MANAGER CALLS

| | |
|---|---|
| ROUTINE | **DMALLOC - Allocate Space for a Display File Record** |
| FUNCTION | To allocate sufficient space from the display file's own heap for a new display file record. |
| INPUTS | D0.W DMALLOC<br>D1.W Size of record required<br>A0.L Display file base address |
| OUTPUTS | D0.W Status<br>D1.W Size of record allocated<br>D2.W Record address (offset from DFBA) |
| STATUS CODES | STDFFULL Display file full |
| SIDE EFFECTS | If a user hook routine has been installed by a call to DMHOOK and the display file is full, the user hook routine will be called to scroll sufficient lines to the top file until space for the new record has been made available. Any status codes returned by the user hook routine are passed to the calling program by DMALLOC. |
| NOTES | DMALLOC will not normally be called by a user program, being a by-product of other DFM commands. It is provided as a user callable function for use in a user hook routine when reading data into a display file. |

The value returned in D2 is an offset on the display file base address and must be added to the value in A0, or used in the displacement addressing mode (A0,D2.W), if a pointer to the space record is to be formed.

The contents of the new record are undefined. If a preformed string is to be copied into the record then the first word of the record must be set to the length of the string.

The size of record allocated will be greater than or equal to the size requested and will allow a variable amount of data insertion before a larger record is required to accommodate the data. Control of record size is invisible to the user.

| | |
|---|---|
| ROUTINE | **DMCURDIS - Disable Cursor in Display File Window** |
| FUNCTION | To disable and hide the cursor in the specified display file window. |
| INPUTS | D0.W DMCURDIS<br>A0.L Display file base address |
| OUTPUTS | D0.W Status |
| STATUS CODES | 0 Always returns success status |
| SIDE EFFECTS | If the (disabled) cursor is moved horizontally out of the window then that window will not scroll horizontally unless the cursor is reenabled. |
| NOTES | DMCURDIS is provided to disable the display of a cursor in any window that requires no user interaction (such as a help menu or heading window). It can also be used to hide the cursor temporarily to avoid excessive cursor movement on the screen in a complex update.<br><br>Note that although horizontal scrolling will be suspended while the cursor is disabled, vertical scrolling will be carried out as if the cursor were visible.<br><br>A count of DMCURDIS calls is maintained and an equal number of DMCURENA calls must be made to reenable the cursor. |

| | | |
|---|---|---|
| ROUTINE | **DMCURENA - Reenable Cursor in Display File Window** | |
| FUNCTION | To reenable the display of the cursor in the specified display file window. | |
| INPUTS | DO.W | DMCURENA |
| | AO.L | Display file base address |
| OUTPUTS | DO.W | Status |
| STATUS CODES | 0 | Always returns success status |
| SIDE EFFECTS | If the cursor has been moved horizontally out of the window while disabled, the window will scroll horizontally to display the cursor. | |
| NOTES | DMCURENA is provided to reenable the cursor after having been disabled by DMCURDIS. | |
| | A count of DMCURDIS calls is maintained and an equal number of DMCURENA calls must be made to reenable the cursor. | |

| | | |
|---|---|---|
| ROUTINE | **DMDELCHR - Delete Character** | |
| FUNCTION | To delete a character from a display file at the current cursor position. | |
| INPUTS | DO.W | DMDELCHR |
| | AO.L | Display file base address |
| OUTPUTS | DO.W | Status |
| STATUS CODES | STDFINV | Cursor is at end of line |
| SIDE EFFECTS | None | |
| NOTES | The character at the cursor position is deleted and the remainder of the line (if any) scrolled left.<br><br>If the cursor is at end of line an STDFINV status code is returned. | |

| | | |
|---|---|---|
| ROUTINE | **DMDELCMD - Delete Screen Driver Command** | |
| FUNCTION | To delete a two-byte screen driver command from a display file at the current action pointer position. | |
| INPUTS | DO.W | DMDELCMD |
| OUTPUTS | DO.W | Status |
| STATUS CODES | STDFINV | AP not pointing at first command byte |
| SIDE EFFECTS | None | |
| NOTES | None | |

| | |
|---|---|
| ROUTINE | **DMDELLIN - Delete Line from Display File** |
| FUNCTION | To delete the line containing the cursor from the display file. |
| INPUTS | DO.W DMDELLIN<br>AO.L Display file base address |
| OUTPUTS | DO.W Status |
| STATUS CODES | As returned from the user hook routine |
| SIDE EFFECTS | If the line is displayed on the screen, the lines below it are scrolled up automatically. Scrolling may invoke the user hook routine (if one has been installed) and as a result may generate the associated system or user defined status codes. |
| NOTES | If the line to be deleted is not the last line of the file, the cursor is left at the start of the line below.<br><br>When deleting is the last line of an extended display file, the cursor is left at the start of the line before and previous lines may be scrolled down.<br><br>When deleting the only line in a display file, the line is blanked and the cursor moved to the start of line. |

| | | |
|---|---|---|
| ROUTINE | **DMDISABL - Suspend Display File Window Update** | |
| FUNCTION | To suspend screen updating for the specified display file window. | |
| INPUTS | DO.W | DMDISABL |
| | AO.L | Display file base address |
| OUTPUTS | DO.W | Status |
| STATUS CODES | 0 | Always returns success status |
| SIDE EFFECTS | None | |
| NOTES | DMDISABL is provided to temporarily switch off screen window repainting during a long and complicated series of display file updates (such as the rejustification of a paragraph in a word processor). This will save time (avoiding multiple repaints of the same line) and will make the screen appear less busy. | |

| | | |
|---|---|---|
| ROUTINE | **DMENABLE - Resume Display File Window Updates** | |
| FUNCTION | To resume screen updating for the specified display file window. | |
| INPUTS | D0.W | DMENABLE |
| | A0.L | Display file base address |
| OUTPUTS | D0.W | Status |
| STATUS CODES | 0 | Always returns success status |
| SIDE EFFECTS | DMENABLE will only update those screen lines which have changed or moved since screen updates were switched off by a call to DMDISABL. | |
| NOTES | DMENABLE is provided to resume screen output after it has been suspended by a previous call to DMDISABL for the same display file. | |

| | | |
|---|---|---|
| ROUTINE | **DMFIXDF - Ensure Display File Window is Visible** | |
| FUNCTION | To ensure that part of the window associated with the specified display file remains displayed within the calling program's screen partition. | |
| INPUTS | DO.W | DMFIXDF |
| | AO.L | Display file base address |
| OUTPUTS | DO.W | Status |
| STATUS CODES | STDFINV | The display file is not displayed in a window |
| SIDE EFFECTS | The cursor associated with the specified display file window flashes and is kept visible within the screen partition at all times unless the virtual screen is metascrolled through the partition by direct user commend. Even then, the next display file update will cause an automatic metascroll to reveal the cursor again. | |

NOTES

This function is provided to enable the programmer to specify which window in a multi-window virtual screen remains visible within the partition despite any scrolling that may occur.

If DMFIXDF is not called or is called with AO = O then all the display file windows are deselected and subsequent DFM commands will scroll the virtual screen through the partition as DFM thinks fit.

If a new display file window is created as a result of an IOOPEN call to KEY: or SCREEN: then this will automatically be fixed within the partition by IOSS.

ROUTINE **DMFLUSH - Flush Display File Text with User Hook Routine**

FUNCTION To copy the contents of the display file and the bottom file to the top file using the user hook routine.

INPUTS

| | |
|---|---|
| DO.W | DMFLUSH |
| AO.L | Display file base address |

OUTPUTS

| | |
|---|---|
| DO.W | Status |

STATUS CODES As returned by the user hook routine

SIDE EFFECTS As generated by the user hook routine

NOTES

This routine is automatically invoked by both DMKILWIN and DMKILLDF, but is provided to enable the calling program to have greater control of error conditions (such as a full disk or microdrive) that may occur when calling the user hook routine.

If an error is detected by DMFLUSH via the user hook routine, the calling program should deal with the error (if possible) prior to calling DMFLUSH again.

Because the bottom file is copied to the top file via the display file, it is recommended that DMDISABL is called prior to the call of DMFLUSH. This will disable the window update and allow the process to execute much faster.

Once DMFLUSH has returned a success status, the display file will be unusable until DMRESET has been called (and DMENABLE if DMDISABL was called).

| | | |
|---|---|---|
| ROUTINE | **DMGETCUR - Get Cursor Position** | |
| FUNCTION | To fetch the current position of the cursor within a display file. | |
| INPUTS | D0.W | DMGETCUR |
| | A0.L | Display file base address |
| OUTPUTS | D0.W | Status |
| | D1.W | Line Number |
| | D2.W | Character position |
| STATUS CODES | 0 | Always returns success status |
| SIDE EFFECTS | None | |
| NOTES | The cursor position is returned in terms of line number and character position. Line 0 is the top of the display file and position 0 is the first character in a line. | |

| | | |
|---|---|---|
| ROUTINE | **DMHOOK - Install User Hook Routine** | |
| FUNCTION | To define the address of a user written routine that will handle extended scrolling from the display file to and from a backing medium such as floppy disk, and is invoked by DFM. | |
| INPUTS | D0.W | DMHOOK |
| | A0.L | Display file base address |
| | A1.L | Address of user hook routine |
| | A2.L | Address of user data block |
| OUTPUTS | D0.W | Status |
| STATUS CODES | 0 | Always returns success status |

SIDE EFFECTS

Once the user hook routine has been defined, any calls to DFM that would cause lines to scroll on or off the top or bottom of a full display file will invoke the user hook routine.

NOTES

DMHOOK can be called more than once for the same display file to install a different hook routine, to handle closedown for example. Setting A1 = 0 will disable the user hook routine.

When reading into the display file the user hook routine must call DMALLOC to grab space for the data string. Note that DMALLOC itself might invoke the user hook routine which must cope with one level of recursion.

When writing from the display file, the user hook routine must simply process the data line it is given, the display file space being reclaimed automatically by DFM.

When 'closing' a display file, use DMFLUSH to flush all data from the display file and the bottom file to the top file.

Entry parameters to the user hook routine from DFM are:

| | |
|---|---|
| D0.B | 0 = read line, otherwise write line |
| D1.B | 0 = top file, otherwise bottom file |
| D2.L | Display file line number |
| A0.L | Display file base address |
| A1.L | Data string address (write only) |
| A2.L | User data block address |

These registers are returned, the others must be preserved:

| | |
|---|---|
| D0.W | 0 = success, otherwise status code |
| A1.L | Data string address (read only) |

| | |
|---|---|
| ROUTINE | **DMINITDF - Initialise Display File** |

FUNCTION

To allocate memory (if requested) for a display file and to initialise the data structures associated with it.

INPUTS

| | |
|---|---|
| D0.W | DMINITDF |
| D1.W | Number of bytes required |
| D2.B | Window foreground and background colours |
| D3.B | Window parameters |
| A0.L | Zero or address of preallocated space |

OUTPUTS

| | |
|---|---|
| D0.W | Status |
| A0.L | Display file base address |

STATUS CODES

| | |
|---|---|
| STPMEM | Insufficient heap memory |

SIDE EFFECTS

If A0 contains zero then the number of bytes specified is grabbed from the calling program's heap, otherwise the memory is assumed to have been preallocated with OSMEMALL.

The display file is initialised as an empty structure preceded by a display file control block (which is itself initialised with the parameters supplied).

NOTES

Colours are specified as numbers in the range 0-7 as follows:

| | |
|---|---|
| 0 | Black |
| 1 | Blue |
| 2 | Red |
| 3 | Magenta |
| 4 | Green |
| 5 | Cyan |
| 6 | Yellow |
| 7 | White |

Bits 6-4: background colour
Bits 2-0: foreground colour

The window parameters are bit numbers as follows:

| | |
|---|---|
| DFLNOSCR | Scroll (0) or metascroll (1) to keep cursor visible |
| DFLWRAP | Scroll (0) or wrap (1) at horizontal edge of window |
| DFLDISPO | Scroll (0) or dispose of (1) lines from top of DF |

The empty display file is not displayed in the window at this stage but by the DMNEWWIN command.

| | | |
|---|---|---|
| ROUTINE | **DMINITVS - Initialise Virtual Screen** | |
| FUNCTION | Create an empty virtual screen and a corresponding screen partition for the calling program. | |
| INPUTS | DO.W | DMINITVS |
| | D1.W | Number of lines required |
| OUTPUTS | DO.W | Status |
| STATUS CODES | STPMEM | Insufficient heap memory |
| | STDFINV | Invalid request |

SIDE EFFECTS

A virtual screen control block is created on the calling program's heap and a corresponding blank screen partition is created at the bottom of the screen. Pointers to the VSCB and the partition are stored in the program's PCB.

DMINITVS will attempt to grab for the new partition the same number of lines as requested for the virtual screen by taking lines from the previous partition. If this shrinks to one line then lines are taken from the partition previous to that, and so on until either the requested number of lines has been obtained or all previous screen partitions have been reduced to one line.

NOTES

A line in this context refers to a line of characters, the precise number of raster lines for this being implementation dependent.

A program is allowed one virtual screen. If an attempt is made to create a second virtual screen or there is no screen space available for the partition (there are already as many partitions as screen lines), an STDFINV status is returned.

| | | |
|---|---|---|
| ROUTINE | **DMINSBLK - Insert a Block** | |
| FUNCTION | To insert a block in a display file at the current cursor position. | |
| INPUTS | D0.W | DMINSBLK |
| | D1.W | Byte count to insert |
| | A0.L | Display file base address |
| | A1.L | Address of block buffer |
| OUTPUTS | D0.W | Status |
| | D1.W | Byte count inserted |
| STATUS CODES | STDFFULL | Display file full |

SIDE EFFECTS

The user hook routine may be invoked by this routine and may return system or user defined status codes.

NOTES

The block may contain a mixture of displayable characters, newline codes or two-byte screen driver commands.

If the block contains newline codes then it will be inserted in sections over the required number of lines.

After insertion the cursor is positioned on the first character position to the right of the inserted block.

If the inserted block would cause the cursor to move off the right hand edge of the window, then if the WRAP condition is set (see DMINITDF) the line will be split at the edge of the window and a new line started, otherwise the window is scrolled left.

The number of bytes actually inserted is returned in D1 to allow for retries in the event of a recoverable status condition. The state of the display may be undefined until a success status is achieved.

| | | |
|---|---|---|
| ROUTINE | **DMINSCHR - Insert Character** | |
| FUNCTION | To insert a character in a display file line at the current cursor position. | |
| INPUTS | D0.W | DMINSCHR |
| | D1.B | Character code |
| | A0.L | Display file base address |
| OUTPUTS | D0.W | Status |
| STATUS CODES | STDFFULL | Display file full |
| SIDE EFFECTS | The user hook routine may be invoked and may return system or user defined status codes. | |

NOTES

Only displayable characters in the range $20-$7F should be inserted. The results of inserting characters outside this range are undefined.

Characters are inserted at the cursor position, the remainder of the line being scrolled right. If the cursor is at the right-hand edge of a window prior to insertion, then if the WRAP condition is set (see DMINITDF) the line will be split after the inserted character, otherwise the window will be scrolled left.

| | | |
|---|---|---|
| ROUTINE | **DMINSLIN - Insert Line in Display File** | |
| FUNCTION | To insert a new line into the specified display file. | |
| INPUTS | DO.W | DMINSLIN |
| | AO.L | Display file base address |
| | Al.L | Address of text string to be inserted |
| OUTPUTS | DO.W | Status |
| STATUS CODES | STDFFULL | Display file full |

SIDE EFFECTS

If the new line is displayed on the screen, lines below it will be scrolled down automatically. Scrolling may invoke the user hook routine (if one has been installed) and as a result may generate the associated system or user defined status codes.

NOTES

The line is inserted immediately above the line containing the cursor. The position of the cursor after line insertion is unchanged.

The text string to be inserted must not contain a newline code.

| | | |
|---|---|---|
| ROUTINE | **DMINSSTR - Insert a String** | |
| FUNCTION | To insert a string in a display file at the current cursor position. | |
| INPUTS | DO.W | DMINSSTR |
| | AO.L | Display file base address |
| | Al.L | Address of string |
| OUTPUTS | DO.W | Status |
| STATUS CODES | STDFFULL | Display file full |
| SIDE EFFECTS | The user hook routine may be invoked by this routine and may return system or user defined status codes. | |

NOTES

The string may contain a mixture of displayable characters, newline codes or two-byte screen driver commands. The string bytecount is not inserted into the display file.

If the string contains newline codes then the string will be inserted in sections over the required number of lines.

After insertion the cursor is positioned on the first character position to the right of the inserted string.

If the inserted string would cause the cursor to move off the right hand edge of the window, then if the WRAP condition is set (see DMINITDF) the line will be split at the edge of the window and a new line started, otherwise the window is scrolled left.

Failure of DMINSSTR may leave the display file in an undefined state.

| | |
|---|---|
| ROUTINE | **DMJOIN - Join Two Lines Together** |
| FUNCTION | The line which contains the cursor is joined with the line immediately beneath it (if any). |
| INPUTS | D0.W DMJOIN<br>A0.L Display file base address |
| OUTPUTS | D0.W Status |
| STATUS CODES | STDFFULL Display file full |
| SIDE EFFECTS | Lines below the pair that are joined are automatically scrolled up. Scrolling may invoke the user hook routine (if one has been installed) and as a result may generate the associated system or user defined status codes. |
| NOTES | Before the DMJOIN call, the cursor can be in any position in the line. After the DMJOIN call, the cursor is placed at the join position (on what had been the first character of the second line). |

| | | |
|---|---|---|
| ROUTINE | **DMKILLDF - Release Display File and Associated Window** | |

FUNCTION

To flush display file text to its top output file (if any), to realease any space grabbed from the calling program's heap when the display file was created, and to coalesce its window with the parent window.

INPUTS

| | |
|---|---|
| D0.W | DMKILLDF |
| A0.L | Display file base address |

OUTPUTS

| | |
|---|---|
| D0.W | Status |

STATUS CODES

| | |
|---|---|
| STDFINV | Window cannot be deleted |

SIDE EFFECTS

If the display file has a top output file the routine DMFLUSH will be used to flush the text. This calls the user hook routine defined by DMHOOK which may return system or user defined status codes.

If the display file was created from the calling program's heap (see DMINITDF) then space is returned to the heap.

If the display file was being displayed in a window of the calling program's virtual screen, DMKILWIN is called to coalesce this window with its parent window (from which it was created by DMNEWWIN). If this fails, DMKILWIN may return STDFINV or user hook status codes.

NOTES

DMKILLDF is the recommended method of releasing display files and tidying up the associated files, data structures and screen areas.

The use of DMNEWWIN to create multiple windows will generate a tree structure of parent to child window relationships. In order to unwind this nesting correctly windows must be killed by DMKILLDF in reverse order of creation.

| | |
|---|---|
| ROUTINE | **DMKILWIN - Delete Screen Window** |
| FUNCTION | To remove the specified window from the calling program's virtual screen. |
| INPUTS | D0.W DMKILWIN<br>A0.L Display file base address |
| OUTPUTS | D0.W Status |
| STATUS CODES | STDFINV Window cannot be deleted |
| SIDE EFFECTS | If the display file was being displayed in a window of the calling program's virtual screen, DMKILWIN will attempt to coalesce this window with its parent window (from which it was created by DMNEWWIN). If this fails, DMKILWIN may return STDFINV or user hook status codes. |
| NOTES | The use of DMNEWWIN to create multiple windows will generate a tree structure of parent to child window relationships. In order to unwind this nesting correctly windows must be killed by DMKILWIN in reverse order of creation. |

| | | |
|---|---|---|
| ROUTINE | **DMMARK - To Set a Marker** | |
| FUNCTION | To define a marker point at the current cursor position. | |
| INPUTS | D0.W | DMMARK |
| | D1.B | Marker number (0-7) |
| | A0.L | Display file base address |
| OUTPUTS | D0.W | Status |
| STATUS CODES | 0 | Always returns success status |
| SIDE EFFECTS | None | |
| NOTES | If the marker number is outside the range 0-7, the result is undefined. | |

| | | |
|---|---|---|
| ROUTINE | **DMMKPOS - Fetch Marker Position** | |
| FUNCTION | To fetch the position of the specified marker position. | |
| INPUTS | D0.W | DMMKPOS |
| | D1.B | Marker number (0-7) |
| | A0.L | Display file base address |
| OUTPUTS | D0.W | Status |
| | D1.W | Line number |
| | D2.W | Character position |
| STATUS CODES | 0 | Always returns success status |
| SIDE EFFECTS | None | |
| NOTES | A marker number outside the range 0-7 will return undefined results. | |
| | Line 0 is the first line of the display file, position 0 is the first character in the line. Undefined markers are returned as position 0,0. | |

| | | |
|---|---|---|
| ROUTINE | **DMMOVECU - Move Cursor** | |
| FUNCTION | To move the cursor in the given direction or to the specified marker position. | |
| INPUTS | DO.W | DMMOVECU |
| | D1.B | Movement specifier |
| | AO.L | Display file base address |
| OUTPUTS | DO.W | Status |
| STATUS CODES | STDFINV | Invalid movement specifier |
| SIDE EFFECTS | The user hook routine may be invoked and may return system or user defined status codes. | |

NOTES

If the cursor is moved out of the window, then the window is scrolled in the required direction until the cursor is visible again.

The movement specifier can have the following values:

| | |
|---|---|
| 0-7 | Move to marker 0-7 |
| CH.CURU | Move up one line |
| CH.CURD | Move down one line |
| CH.CURL | Move left one character |
| CH.CURR | Move right one character |

When moving the cursor up or down onto a line shorter than the current line and to a position that would be past the end of line, the cursor will be positioned at the end of line.

If a move left or right is requested and the cursor is at the start or end of line, then the command is ignored.

When the cursor is moved, the action pointer is moved to the same position.

ROUTINE **DMNEWWIN - Add a Window to the Virtual Screen**

FUNCTION To create the initial rectangular window in a virtual screen or to split the rectangular window associated with the parent display file into two smaller rectangles as specified and to associate the new display file with one of these windows.

INPUTS

| | | |
|---|---|---|
| | D0.W | DMNEWWIN |
| | D1.W | Orientation |
| | D2.W | Window size |
| | A0.L | Display file base address |
| | A1.L | Parent display file base address |

OUTPUTS

| | | |
|---|---|---|
| | D0.W | Status |
| | D1.W | Window width in characters |
| | D2.W | Window depth in lines |

STATUS CODES STDFINV New window too large

SIDE EFFECTS If there are insufficient lines in the new display file to fill the new window and a user hook routine has been defined for the new window by DMHOOK, then it will be called to read lines from the bottom file into the display file until the window is full or no more lines are available.

NOTES If this is the first window to be created in the virtual screen then the parameters in D1, D2 and A1 are ignored and the dimensions of the virtual screen are returned in D1 and D2.

If the orientation is vertical (D1=0), the parent window is split vertically with the absolute value of D2 specifying the width (in characters) of the new window. If D2 is positive, the new window is created on the left, otherwise it is created on the right.

If the orientation is horizontal (D1≠0), the parent window is split horizontally with the absolute value of D2 specifying the depth (in lines) of the new window. If D2 is positive, the new window is created at the top, otherwise it is created at the bottom.

The use of DMNEWWIN to create multiple windows will generate a tree structure of parent to child window relationships. In order to unwind this nesting correctly windows must be killed by DMKILWIN in reverse order of creation.

| | |
|---|---|
| ROUTINE | **DMPUTCUR - Position the Cursor** |
| FUNCTION | To move the cursor to the defined position within a display file. |
| INPUTS | D0.W DMPUTCUR<br>D1.W Line number<br>D2.W Character position<br>A0.L Display file base address |
| OUTPUTS | D0.W Status<br>D1.W Line number<br>D2.W Character position |
| STATUS CODES | STEOF Attempt to move cursor beyond file boundary |
| SIDE EFFECTS | The user hook routine may be invoked and may return system or user defined status codes. |
| NOTES | If the specified position cannot be reached, then the actual position reached is returned in D1 and D2.<br><br>Line 0 is the top line of a display file, position 0 is the first character in a line. |

| | | |
|---|---|---|
| ROUTINE | **DMRDBYT - Move Action Pointer and Read Byte** | |
| FUNCTION | To move the display file action pointer within a line without moving the cursor and to read the byte addressed. | |
| INPUTS | D0.W | DMRDBYT |
| | D1.W | Amount to move action pointer |
| | A0.L | Display file base address |
| OUTPUTS | D0.W | Status |
| | D1.B | Byte addressed by action pointer |
| | D2.W | Horizontal character position of action pointer |
| STATUS CODES | STEOF | Attempt to move action pointer beyond line limits |
| SIDE EFFECTS | None | |

NOTES

The action pointer is moved according to the value contained in D1, as follows:

D1 = -N move N bytes to the left
D1 = 0 retain current position
D1 = +N move N bytes to the right

The action pointer must remain in the same line as the cursor and is therefore not allowed to move past the start or end of the current line. If the value in D1 would cause the action pointer to move beyond the line limits, it is moved to the start or end of line (as appropriate) and an STEOF error code is returned.

If the action pointer is moved to the end of line, a newline character code is returned (CH.NL).

Unlike the cursor, which can only be placed on a displayable character, newline or space command, the action pointer can be positioned on and read any byte from a display file line, enabling the calling program to read the non-displayable data within the line.

In order to maintain track of the (displayable) character position within the line after action pointer movement, the value returned in D2 is the closest potential cursor position equal to or to the right of the action pointer.

| | | |
|---|---|---|
| ROUTINE | **DMRELEAS - Deallocate Display File Record Space** | |
| FUNCTION | To release a display file record to the display file heap. | |
| INPUTS | D0.W | DMRELEAS |
| | A0.L | Display file base address |
| | A1.W | Offset to record from DFBA |
| OUTPUTS | D0.W | Status |
| STATUS CODES | 0 | Always returns success status |
| SIDE EFFECTS | None | |

NOTES

DMRELEAS will not normally be called by a user program, being a by-product of other DFM commands. It is provided as a user callable function for use in a user hook routine when writing data from a display file.

Note that the user hook routine provides a pointer to the record in A1.L and DMRELEAS requires an offset on the display file base address in A1.W, thus some maths is necessary.

| | |
|---|---|
| ROUTINE | **DMRESET - Delete and Refresh Display File Text** |
| FUNCTION | To delete all the text in the specified display file, home the cursor and refresh the window from the user hook routine. |
| INPUTS | D0.W DMRESET<br>A0.L Display file base address |
| OUTPUTS | D0.W Status |
| STATUS CODES | As returned from the user hook routine. |
| SIDE EFFECTS | If DMHOOK has been called previously and there is data in bottom input file then this will be read into the display file and displayed in the window. System or user defined status codes may be returned in the process. |
| NOTES | This routine may be called whether or not the display file is displayed in a window.<br><br>If the display file is displayed in a window then sufficient text is read from the bottom input file to fill the window, otherwise only a single line is read.<br><br>If DMHOOK has not been called or the bottom file is empty then the display file will be left with a single blank line. |

| | | |
|---|---|---|
| ROUTINE | **DMSPLIT - Split a Line Into Two** | |
| FUNCTION | To split the current line (at the cursor position) into two lines. | |
| INPUTS | DO.W | DMSPLIT |
| | AO.L | Display file base address |
| OUTPUTS | DO.W | Status |
| STATUS CODES | STDFFULL | Display file full |
| SIDE EFFECTS | Lines following the split are automatically scrolled down. Scrolling may invoke the user hook routine (if one has been installed) and as a result may generate the associated system or user defined status codes. | |
| NOTES | The line is split immediately before the cursor position and the cursor remains at the start of the second line. | |

| | | |
|---|---|---|
| ROUTINE | **DMTTYSEL - Select Window for Console Output** | |
| FUNCTION | To select the window associated with the specified display file to be used for keyboard line reflection. | |
| INPUTS | D0.W | DMTTYSEL |
| | A0.L | Display file base address |
| OUTPUTS | D0.W | Status |
| STATUS CODES | 0 | Always returns success status |
| SIDE EFFECTS | All subsequent IOGETLIN calls from KEY: made by the calling program will echo keystrokes to the specified window. | |

NOTES

This command is used to select which of the calling program's windows will be used for keyboard line reflection. When used to reflect keyboard input, IOGETLIN will accept and reflect the following:

(a) Standard ASCII characters

(b) Backspace or backspace-delete

(c) Delete line

(d) Enter

All other characters are ignored and are not reflected in the window.

If DMTTYSEL has not been called or was called with A0 = 0, then an IOGETLIN call to KEY: or an IOOPEN call to SCREEN: will cause an attempt to create a default window. This will be the entire virtual screen if no windows exist, otherwise the top eight lines of the last window created.

An IOOPEN call to SCREEN: will open a channel to the window currently selected by DMTTYSEL. Subsequent calls to DMTTYSEL will have no effect on any channel already opened. Thus it is possible to have many SCREEN: channels open to different windows, but only one window for KEY: screen reflection.

| | | |
|---|---|---|
| ROUTINE | **DMUMENU - Update Single Line Menu Field** | |
| FUNCTION | To update the single line menu at the bottom of the screen display with the contents of the specified string. | |
| INPUTS | DO.W | DMUMENU |
| | AO.L | Address of menu string |
| OUTPUTS | DO.W | Status |
| STATUS CODES | 0 | Always returns success status |
| SIDE EFFECTS | The address of the menu string is copied to the contents of PCB field PBMENU and becomes the new default menu string for the calling program. | |

NOTES

The single line menu is displayed at the bottom of the screen beneath the space allocated for partitions and will be truncated to the screen line length if necessary. It is normally used for the display of program identification and function key actions associated with the current program.

Each program in the system may have a default string for display in the single line menu, addressed by field PBMENU in the PCB. This is displayed automatically when the program is selected (usually by user keyboard command) as the current program, and is updated by a call to DMUMENU. If PBMENU is zero the system default string is output.

| | | |
|---|---|---|
| ROUTINE | **DMWRBYT - Update Byte in Display File** | |
| FUNCTION | To replace the byte addressed by the action pointer by the byte specified. | |
| INPUTS | DO.W | DMWRBYT |
| | D1.B | Byte to be stored |
| | AO.L | Display file base address |
| OUTPUTS | DO.W | Status |
| STATUS CODES | STDFINV | Invalid update |

SIDE EFFECTS

If the update affects the screen display, the appropriate screen areas are repainted as follows:

(a) An updated displayable character is repainted.

(b) An updated space command causes a line repaint.

(c) An updated colour or fount command causes all affected lines to be repainted.

The display file data structures are updated accordingly.

NOTES

DMWRBYT operates as an overstrike function thus the action pointer is not moved as a result of the call.

Certain operations are not allowed, as follows:

(a) A newline cannot be overwritten.

(b) A command byte cannot be overwritten by a displayable character.

Note that all commands can be modified by DMWRBYT.

# APPENDIX D:

# GRAPHICS ROUTINE CALLS

| | |
|---|---|
| ROUTINE | **SPBLOCK - Draw a Filled Rectangular Block on the Screen** |
| FUNCTION | To draw a rectangular block at the coordinates specified, relative to the display file and graphics window. |
| INPUTS | D0.W SPBLOCK<br>D1.W X coordinate of start (any corner)<br>D2.W Y coordinate of start<br>D3.W X coordinate of finish (corner diagonally opposite)<br>D4.W Y coordinate of finish<br>D7.W Colour identifier<br>A0.L Display file base address<br>A1.L Graphics window block |
| OUTPUTS | D0.W Destroyed |
| STATUS CODES | None |
| SIDE EFFECTS | The block will be clipped if its size and position are such that it extends outside the visible portion of its window. |
| NOTES | The upper byte of the colour identifier is set non-zero for XOR ink, the lower byte has the following fields: |

Bits 7-6 Stipple (0 = Q, 1 = H, 2 = V, 3 = C)
Bits 5-3 XOR of mixer colour and base colour
Bits 2-0 Base colour

Colours are specified as numbers in the range 0-7 as follows:

| | | | |
|---|---|---|---|
| 0 | Black | 4 | Green |
| 1 | Blue | 5 | Cyan |
| 2 | Red | 6 | Yellow |
| 3 | Magenta | 7 | White |

The graphics window is positioned relative to the display file as defined by the graphics window block fields:

| | | | |
|---|---|---|---|
| CLIPXPOS | (.W) | X position offset | (characters) |
| CLIPYPOS | (.W) | Y position offset | (character lines) |
| CLIPWID | (.W) | X width of window | (characters) |
| CLIPSIZ | (.W) | Y height of window | (character lines) |

| | | |
|---|---|---|
| ROUTINE | **SPELLIPS - Draw an Ellipse on the Screen** | |
| FUNCTION | To draw an orthogonal ellipse on the screen. | |
| INPUTS | D0.W | SPELLIPS |
| | D1.W | X coordinate of centre |
| | D2.W | Y coordinate of centre |
| | D3.W | X radius |
| | D4.W | Y radius |
| | D7.W | Colour identifier |
| | A0.L | Display file base address |
| | A1.L | Graphics window block |
| OUTPUTS | D0.W | Destroyed |
| STATUS CODES | None | |
| SIDE EFFECTS | The ellipse will be clipped if its size and position are such that it extends outside the visible portion of its window. | |

NOTES

To produce the effect of a circle, it is recommended that the X radius exceeds the Y radius by around 25% to 35% depending on the particular monitor or TV in use.

The upper byte of the colour identifier is set non-zero for XOR ink, the lower byte has the following fields:

Bits 7-6 Stipple (0 = Q, 1 = H, 2 = V, 3 = C)
Bits 5-3 XOR of mixer colour and base colour
Bits 2-0 Base colour

Colours are specified as numbers in the range 0-7 as follows:

| | | | |
|---|---|---|---|
| 0 | Black | 4 | Green |
| 1 | Blue | 5 | Cyan |
| 2 | Red | 6 | Yellow |
| 3 | Magenta | 7 | White |

The graphics window is positioned relative to the display file as defined by the graphics window block fields:

| | | | |
|---|---|---|---|
| CLIPXPOS | (.W) | X position offset | (characters) |
| CLIPYPOS | (.W) | Y position offset | (character lines) |
| CLIPWID | (.W) | X width of window | (characters) |
| CLIPSIZ | (.W) | Y height of window | (character lines) |

| | |
|---|---|
| ROUTINE | **SPFILL - Area Fill to a Specified Border** |
| FUNCTION | To fill an area in the specified colour up to a defined border colour. |

INPUTS

| | |
|---|---|
| D0.W | SPFILL |
| D1.W | X coordinate of start |
| D2.W | Y coordinate of start |
| D6.W | Border colour identifier |
| D7.W | Fill colour identifier |
| A0.L | Display file base address |
| A1.L | Graphics window block |

OUTPUTS

D0.W Destroyed

STATUS CODES

None

SIDE EFFECTS

The filled area will be clipped if it extends outside the visible portion of its window.

Strange effects may occur if the fill, border or original background colour have plain or stipple colours in common.

NOTES

The start coordinates should be placed well within the fill area to avoid colour boundary effects.

The upper byte of the colour identifier is set zero. The lower byte has the following fields:

Bits 7-6 Stipple (0 = Q, 1 = H, 2 = V, 3 = C)
Bits 5-3 XOR of mixer colour and base colour
Bits 2-0 Base colour

Colours are specified as numbers in the range 0-7 as follows:

| | | | |
|---|---|---|---|
| 0 | Black | 4 | Green |
| 1 | Blue | 5 | Cyan |
| 2 | Red | 6 | Yellow |
| 3 | Magenta | 7 | White |

The graphics window is positioned relative to the display file as defined by the graphics window block fields:

| | | | |
|---|---|---|---|
| CLIPXPOS | (.W) | X position offset | (characters) |
| CLIPYPOS | (.W) | Y position offset | (character lines) |
| CLIPWID | (.W) | X width of window | (characters) |
| CLIPSIZ | (.W) | Y height of window | (character lines) |

| | | |
|---|---|---|
| ROUTINE | **SPLINE - Draw a Line on the Screen** | |
| FUNCTION | To draw a straight line between two defined points. | |
| INPUTS | D0.W | SPLINE |
| | D1.W | X coordinate of start |
| | D2.W | Y coordinate of start |
| | D3.W | X coordinate of end |
| | D4.W | Y coordinate of end |
| | D7.W | Colour identifier |
| | A0.L | Display file base address |
| | A1.L | Graphics window block |
| OUTPUTS | D0.W | Destroyed |
| STATUS CODES | None | |
| SIDE EFFECTS | The line will be clipped if its size and position are such that it extends outside the visible portion of its window. | |

NOTES

The upper byte of the colour identifier is set non-zero for XOR ink, the lower byte has the following fields:

Bits 7-6 Stipple (0 = Q, 1 = H, 2 = V, 3 = C)
Bits 5-3 XOR of mixer colour and base colour
Bits 2-0 Base colour

Colours are specified as numbers in the range 0-7 as follows:

| | | | |
|---|---|---|---|
| 0 | Black | 4 | Green |
| 1 | Blue | 5 | Cyan |
| 2 | Red | 6 | Yellow |
| 3 | Magenta | 7 | White |

The graphics window is positioned relative to the display file as defined by the graphics window block fields:

| | | | |
|---|---|---|---|
| CLIPXPOS | (.W) | X position offset | (characters) |
| CLIPYPOS | (.W) | Y position offset | (character lines) |
| CLIPWID | (.W) | X width of window | (characters) |
| CLIPSIZ | (.W) | Y height of window | (character lines) |

| | |
|---|---|
| ROUTINE | **SPPAINT - Area Fill to an Unspecified Border** |
| FUNCTION | To fill an area in the specified colour over the current background colour until the background colour changes. |

INPUTS

| | | |
|---|---|---|
| | D0.W | SPPAINT |
| | D1.W | X coordinate of start |
| | D2.W | Y coordinate of start |
| | D7.W | Fill colour identifier |
| | A0.L | Display file base address |
| | A1.L | Graphics window block |

OUTPUTS — D0.W Destroyed

STATUS CODES — None

SIDE EFFECTS

The filled area will be clipped if it extends outside the visible portion of its window.

Strange effects may occur if the fill, border or original background colour have plain or stipple colours in common.

NOTES

The start coordinates should be placed well within the fill area to avoid colour boundary effects.

The upper byte of the colour identifier is set zero. The lower byte has the following fields:

Bits 7-6 Stipple (0 = Q, 1 = H, 2 = V, 3 = C)
Bits 5-3 XOR of mixer colour and base colour
Bits 2-0 Base colour

Colours are specified as numbers in the range 0-7 as follows:

| | | | |
|---|---|---|---|
| 0 | Black | 4 | Green |
| 1 | Blue | 5 | Cyan |
| 2 | Red | 6 | Yellow |
| 3 | Magenta | 7 | White |

The graphics window is positioned relative to the display file as defined by the graphics window block fields:

| | | | |
|---|---|---|---|
| CLIPXPOS | (.W) | X position offset | (characters) |
| CLIPYPOS | (.W) | Y position offset | (character lines) |
| CLIPWID | (.W) | X width of window | (characters) |
| CLIPSIZ | (.W) | Y height of window | (character lines) |

| | |
|---|---|
| ROUTINE | **SPPOINT - Draw a (Clipped) Pixel on the Screen** |
| FUNCTION | To draw a pixel on the screen at the defined coordinate position. |

INPUTS

| | |
|---|---|
| D0.W | SPPOINT |
| D1.W | X coordinate |
| D2.W | Y coordinate |
| D7.W | Colour identifier |
| A0.L | Display file base address |
| A1.L | Graphics window block |

OUTPUTS

D0.W Destroyed

STATUS CODES

None

SIDE EFFECTS

The pixel will not be drawn if lies outside the visible portion of its window.

NOTES

The upper byte of the colour identifier is set non-zero for XOR ink, the lower byte has the following fields:

| | |
|---|---|
| Bits 7-6 | Stipple (0 = Q, 1 = H, 2 = V, 3 = C) |
| Bits 5-3 | XOR of mixer colour and base colour |
| Bits 2-0 | Base colour |

Colours are specified as numbers in the range 0-7 as follows:

| | | | |
|---|---|---|---|
| 0 | Black | 4 | Green |
| 1 | Blue | 5 | Cyan |
| 2 | Red | 6 | Yellow |
| 3 | Magenta | 7 | White |

The graphics window is positioned relative to the display file as defined by the graphics window block fields:

| | | | |
|---|---|---|---|
| CLIPXPOS | (.W) | X position offset | (characters) |
| CLIPYPOS | (.W) | Y position offset | (character lines) |
| CLIPWID | (.W) | X width of window | (characters) |
| CLIPSIZ | (.W) | Y height of window | (character lines) |

| | | |
|---|---|---|
| ROUTINE | **SPTEXT - Draw a Text String in Graphics Mode** | |
| FUNCTION | To draw a horizontal text string on the screen. | |
| INPUTS | D0.W | SPTEXT |
| | D1.W | X coordinate of start |
| | D2.W | Y coordinate of start |
| | D5.B | Text attributes |
| | D6.W | Background colour identifier |
| | D7.W | Foreground colour identifier |
| | A0.L | Display file base address |
| | A1.L | Address of graphics window block |
| | A2.L | Address of text string |
| OUTPUTS | D0.W | Destroyed |
| STATUS CODES | None | |
| SIDE EFFECTS | The text string will be clipped (in whole characters) if it extends outside the visible portion of its window. | |

NOTES

The upper byte of the colour identifier is set non-zero for XOR ink, the lower byte has the following fields:

Bits 7-6 Stipple (0 = Q, 1 = H, 2 = V, 3 = C)
Bits 5-3 XOR of mixer colour and base colour
Bits 2-0 Base colour

Colours are specified as numbers in the range 0-7 as follows:

| | | | |
|---|---|---|---|
| 0 | Black | 4 | Green |
| 1 | Blue | 5 | Cyan |
| 2 | Red | 6 | Yellow |
| 3 | Magenta | 7 | White |

The graphics window is positioned relative to the display file as defined by the graphics window block fields:

| | | | |
|---|---|---|---|
| CLIPXPOS | (.W) | X position offset | (characters) |
| CLIPYPOS | (.W) | Y position offset | (character lines) |
| CLIPWID | (.W) | X width of window | (characters) |
| CLIPSIZ | (.W) | Y height of window | (character lines) |

Text attributes are as follows:

| | | | |
|---|---|---|---|
| Bit 0 | Underlined | Bit 4 | Double height |
| Bit 1 | Flashing | Bit 5 | Spaced characters |
| Bit 2 | Transparent | Bit 6 | Double width |
| Bit 3 | XOR ink (+ bit 2) | Bit 7 | (not used) |

# APPENDIX E:

# STATUS CODES

# E STATUS CODES

## E.1 Format

A status code is a two byte positive integer that is returned from most 68K/OS system calls in register D0. The status code defines whether the system call was executed succesfully (zero) or if a particular fail state was detected (non-zero).

## E.2 Alphabetical List of Status Codes

| | |
|---|---|
| STADDRES | Address error trap |
| STALINE | A-line unimplemented instruction |
| STAM | This access mode not allowed for this channel or file |
| STAMSEQ | This access mode illegal on sequential channels |
| STAT | Illegal option byte or access type |
| STATSEQ | Cannot both read and write a sequential channel |
| STBADDIR | Error in directory component |
| STBADFIL | Error in filename component |
| STBUS | Bus error trap |
| STCHAN | Illegal channel number |
| STCHKINS | CHK instruction trap (array bound violation) |
| STDEVICE | Unknown device |
| STDEVSEQ | Device can be accessed sequentially only |
| STDFINV | Invalid call to display file manager |
| STDIRECT | Directory operations not allowed on this device |
| STDIRFUL | No space left on disk or directory |
| STDIVIDE | Divide by zero trap |
| STEOF | End of file |
| STEXIST | File does not exist |
| STFLINE | F-line unimplemented instruction |
| STGET | Reading from this channel is illegal |
| STILLEGA | Illegal instruction trap |
| STINPROC | Invalid procedure identifier |
| STINPROG | Invalid program identifier |
| STINTRAP | Call of OSTRAP failed |
| STIOERR | Read or write error from device |
| STKILLED | Program was killed by TRAP 0 |
| STMUFLD | Invalid menu fields specified |
| STMURAM | Menu variable data space insufficient |
| STNODIR | Directory cannot be found |
| STNOFILE | No room left in open files table |
| STNOLOAD | Unable to load procedure |
| STNOPROC | No room left in procedure table |
| STNOSHAR | Device is not sharable |
| STNOSIZE | No file size information available |
| STOK | Success status (0) |
| STOPEN | Directory in use, cannot dismount |
| STPART | Partial line only read by IOGETLIN |
| STPMEM | Heap or stack overflow |
| STPRIV | Privileged operation trap |
| STPROC | Procedure already exists (from IODEFPRO) |
| STPUT | Writing to this channel is illegal |
| STRENAME | Source and destination path names incompatible |
| STSEQ | Channel is open for sequential access only |
| STSETPOS | Invalid file position |

| | |
|---|---|
| STSMEM | Memory manager cannot satisfy request |
| STSPECIA | Driver does not support IOSPECIAL operations |
| STSPUR | Spurious interrupt |
| STSTOP | Child program has stopped |
| STSTRLEN | String length invalid |
| STSYNTAX | Path name syntax error |
| STTFRLEN | Invalid transfer or buffer length |
| STTRACE | Trace mode trap |
| STTRAP4 | Trap 4 instruction |
| STTRAP5 | Trap 5 instruction |
| STTRAP6 | Trap 6 instruction |
| STTRAP7 | Trap 7 instruction |
| STTRAP8 | Trap 8 instruction |
| STTRAP9 | Trap 9 instruction |
| STTRAP10 | Trap 10 instruction |
| STTRAP11 | Trap 11 instruction |
| STTRAP12 | Trap 12 instruction |
| STTRAP13 | Trap 13 instruction |
| STTRAP14 | Trap 14 instruction |
| STTRAP15 | Trap 15 instruction |
| STTRAPV | TRAPV instruction trap |
| STUNIT | Unit number invalid or in use |
| STUSE | File in use or requested access incompatible |

# APPENDIX F:

# CHARACTER CODES

# F CHARACTER CODES

## F.1 General

This section defines character codes for the QL keyboard, internal ASCII encoding and screen display as follows:

(a) Keyboard ASCII decode tables

(b) Summary of Keyboard Command Keystrokes

(c) Display File Manager Binary Commands

These define all the character translation and encoding in 68K/OS that is available to applications software. Direct access to keyboard matrix codes is detailed in the Systems Programmer's Reference Manual.

## F.2 Changes From Standard US ASCII

Note that the ASCII is non-standard (to reflect the QL keyboard engraving and for compatibility with QDOS in the 7-bit ASCII range), but the differences are minor:

(a) The ASCII grave accent is replaced by '£' ($60).

(b) The copyright sign is added in the rubout position ($7F).

All other codes conform to standard US ASCII.

## F.3 QL ASCII Decode Table ($00-$1F)

| ASCII HEX | ASCII DEC | SHIFT S C A | KEY U S | FUNCTION (USER MODE) | FUNCTION (SYSTEM MODE) |
|---|---|---|---|---|---|
| $00 | 0 | 1 1 0 | 2 @ | null code | |
| $01 | 1 | X 1 0 | A | | |
| $02 | 2 | X 1 0 | B | | |
| $03 | 3 | X 1 0 | C | | |
| $04 | 4 | X 1 0 | D | | |
| $05 | 5 | X 1 0 | E | | |
| $06 | 6 | X 1 0 | F | | |
| $07 | 7 | X 1 0 | G | | |
| $08 | 8 | X 1 0 | H | | |
| $09 | 9 | X 1 0 | TAB | next field | |
| $0A | 10 | X 1 0 | ENTER | new line | |
| $0B | 11 | X 1 0 | K | | |
| $0C | 12 | X 1 0 | L | | |
| $0D | 13 | X 1 0 | M | | |
| $0E | 14 | X 1 0 | N | | |
| $0F | 15 | X 1 0 | O | | |
| $10 | 16 | X 1 0 | P | | |
| $11 | 17 | X 1 0 | Q | | |
| $12 | 18 | X 1 0 | R | | |
| $13 | 19 | X 1 0 | S | | |
| $14 | 20 | X 1 0 | T | | |
| $15 | 21 | X 1 0 | U | | |
| $16 | 22 | X 1 0 | V | | |
| $17 | 23 | X 1 0 | W | | |
| $18 | 24 | X 1 0 | X | | |
| $19 | 25 | X 1 0 | Y | | |
| $1A | 26 | X 1 0 | Z | | |
| $1B | 27 | 0 1 0 | ESCAPE | exit command seq. | exit system mode |
| $1C | 28 | 0 1 0 | \ \| | | |
| $1D | 29 | 0 1 0 | ] } | | |
| $1E | 30 | 1 1 0 | 6 ^ | | |
| $1F | 31 | 1 1 0 | - _ | | |

## QL ASCII Decode Table ($20-$3F)

| ASCII HEX | ASCII DEC | SHIFT S C A | KEY U S | DISPLAY CHAR. | COMMENT |
|---|---|---|---|---|---|
| $20 | 32 | 0 0 0 | SPACE | SPACE | |
| $21 | 33 | 1 0 0 | 1 ! | ! | |
| $22 | 34 | 1 0 0 | ' " | " | |
| $23 | 35 | 1 0 0 | 3 # | # | US ASCII position |
| $24 | 36 | 1 0 0 | 4 $ | $ | |
| $25 | 37 | 1 0 0 | 5 % | % | |
| $26 | 38 | 1 0 0 | 7 & | & | |
| $27 | 39 | 0 0 0 | ' " | ' | |
| $28 | 40 | 1 0 0 | 9 ( | ( | |
| $29 | 41 | 1 0 0 | 0 ) | ) | |
| $2A | 42 | 1 0 0 | 8 * | * | |
| $2B | 43 | 1 0 0 | = + | + | |
| $2C | 44 | 0 0 0 | , < | , | |
| $2D | 45 | 0 0 0 | - _ | - | |
| $2E | 46 | 0 0 0 | . > | . | |
| $2F | 47 | 0 0 0 | / ? | / | |
| $30 | 48 | 0 0 0 | 0 ) | 0 | |
| $31 | 49 | 0 0 0 | 1 ! | 1 | |
| $32 | 50 | 0 0 0 | 2 @ | 2 | |
| $33 | 51 | 0 0 0 | 3 # | 3 | |
| $34 | 52 | 0 0 0 | 4 $ | 4 | |
| $35 | 53 | 0 0 0 | 5 % | 5 | |
| $36 | 54 | 0 0 0 | 6 ^ | 6 | |
| $37 | 55 | 0 0 0 | 7 & | 7 | |
| $38 | 56 | 0 0 0 | 8 * | 8 | |
| $39 | 57 | 0 0 0 | 9 ( | 9 | |
| $3A | 58 | 1 0 0 | ; : | : | |
| $3B | 59 | 0 0 0 | ; : | ; | |
| $3C | 60 | 1 0 0 | , < | < | |
| $3D | 61 | 0 0 0 | = + | = | |
| $3E | 62 | 1 0 0 | . > | > | |
| $3F | 63 | 1 0 0 | / ? | ? | |

## QL ASCII Decode Table ($40-$5F)

| ASCII HEX | ASCII DEC | SHIFT S C A | KEY U S | DISPLAY CHAR. | COMMENT |
|---|---|---|---|---|---|
| $40 | 64 | 1 0 0 | 2 @ | @ | |
| $41 | 65 | 1 0 0 | A | A | |
| $42 | 66 | 1 0 0 | B | B | |
| $43 | 67 | 1 0 0 | C | C | |
| $44 | 68 | 1 0 0 | D | D | |
| $45 | 69 | 1 0 0 | E | E | |
| $46 | 70 | 1 0 0 | F | F | |
| $47 | 71 | 1 0 0 | G | G | |
| $48 | 72 | 1 0 0 | H | H | |
| $49 | 73 | 1 0 0 | I | I | |
| $4A | 74 | 1 0 0 | J | J | |
| $4B | 75 | 1 0 0 | K | K | |
| $4C | 76 | 1 0 0 | L | L | |
| $4D | 77 | 1 0 0 | M | M | |
| $4E | 78 | 1 0 0 | N | N | |
| $4F | 79 | 1 0 0 | O | O | |
| $50 | 80 | 1 0 0 | P | P | |
| $51 | 81 | 1 0 0 | Q | Q | |
| $52 | 82 | 1 0 0 | R | R | |
| $53 | 83 | 1 0 0 | S | S | |
| $54 | 84 | 1 0 0 | T | T | |
| $55 | 85 | 1 0 0 | U | U | |
| $56 | 86 | 1 0 0 | V | V | |
| $57 | 87 | 1 0 0 | W | W | |
| $58 | 88 | 1 0 0 | X | X | |
| $59 | 89 | 1 0 0 | Y | Y | |
| $5A | 90 | 1 0 0 | Z | Z | |
| $5B | 91 | 0 0 0 | [ { | [ | |
| $5C | 92 | 0 0 0 | \ | | \ | |
| $5D | 93 | 0 0 0 | ] } | ] | |
| $5E | 94 | 1 0 0 | 6 ^ | ^ | |
| $5F | 95 | 1 0 0 | - _ | _ | |

QL ASCII Decode Table ($60-$7F)

| ASCII HEX | ASCII DEC | SHIFT S C A | KEY U S | DISPLAY CHAR. | COMMENT |
|---|---|---|---|---|---|
| $60 | 96 | 0 0 0 | £ ~ | £ | replaces grave accent |
| $61 | 97 | 0 0 0 | A | a | |
| $62 | 98 | 0 0 0 | B | b | |
| $63 | 99 | 0 0 0 | C | c | |
| $64 | 100 | 0 0 0 | D | d | |
| $65 | 101 | 0 0 0 | E | e | |
| $66 | 102 | 0 0 0 | F | f | |
| $67 | 103 | 0 0 0 | G | g | |
| $68 | 104 | 0 0 0 | H | h | |
| $69 | 105 | 0 0 0 | I | i | |
| $6A | 106 | 0 0 0 | J | j | |
| $6B | 107 | 0 0 0 | K | k | |
| $6C | 108 | 0 0 0 | L | l | |
| $6D | 109 | 0 0 0 | M | m | |
| $6E | 110 | 0 0 0 | N | n | |
| $6F | 111 | 0 0 0 | O | o | |
| $70 | 112 | 0 0 0 | P | p | |
| $71 | 113 | 0 0 0 | Q | q | |
| $72 | 114 | 0 0 0 | R | r | |
| $73 | 115 | 0 0 0 | S | s | |
| $74 | 116 | 0 0 0 | T | t | |
| $75 | 117 | 0 0 0 | U | u | |
| $76 | 118 | 0 0 0 | V | v | |
| $77 | 119 | 0 0 0 | W | w | |
| $78 | 120 | 0 0 0 | X | x | |
| $79 | 121 | 0 0 0 | Y | y | |
| $7A | 122 | 0 0 0 | Z | z | |
| $7B | 123 | 1 0 0 | [ { | { | |
| $7C | 124 | 1 0 0 | \ \| | \| | |
| $7D | 125 | 1 0 0 | ] } | } | |
| $7E | 126 | 1 0 0 | £ ~ | ~ | |
| $7F | 127 | 1 0 0 | ESC (C) | (C) | copyright in rubout position |

**QL ASCII Decode Table ($80-$9F)**

| ASCII HEX | DEC | SHIFT S | C | A | KEY U | S | FUNCTION (USER MODE) | FUNCTION (SYSTEM MODE) |
|---|---|---|---|---|---|---|---|---|
| $80 | 128 | 1 | X | 1 | 2 | @ | | |
| $81 | 129 | X | X | 1 | A | | | |
| $82 | 130 | X | X | 1 | B | | | |
| $83 | 131 | X | X | 1 | C | | | |
| $84 | 132 | X | X | 1 | D | | | |
| $85 | 133 | X | X | 1 | E | | | |
| $86 | 134 | X | X | 1 | F | | | |
| $87 | 135 | X | X | 1 | G | | | |
| $88 | 136 | X | X | 1 | H | | | |
| $89 | 137 | X | X | 1 | I | | | |
| $8A | 138 | X | X | 1 | J | | | |
| $8B | 139 | X | X | 1 | K | | | |
| $8C | 140 | X | X | 1 | L | | | |
| $8D | 141 | X | X | 1 | M | | | |
| $8E | 142 | X | X | 1 | N | | | |
| $8F | 143 | X | X | 1 | O | | | |
| $90 | 144 | X | X | 1 | P | | | |
| $91 | 145 | X | X | 1 | Q | | | |
| $92 | 146 | X | X | 1 | R | | | |
| $93 | 147 | X | X | 1 | S | | | |
| $94 | 148 | X | X | 1 | T | | | |
| $95 | 149 | X | X | 1 | U | | | |
| $96 | 150 | X | X | 1 | V | | | |
| $97 | 151 | X | X | 1 | W | | | |
| $98 | 152 | X | X | 1 | X | | | |
| $99 | 153 | X | X | 1 | Y | | | |
| $9A | 154 | X | X | 1 | Z | | | |
| $9B | 155 | 0 | X | 1 | ESCAPE | | | |
| $9C | 156 | 0 | X | 1 | \ | \| | | |
| $9D | 157 | 0 | X | 1 | ] | } | | |
| $9E | 158 | 1 | X | 1 | 6 | ^ | | |
| $9F | 159 | 1 | X | 1 | - | _ | | |

QL ASCII Decode Table ($A0-$BF)

| ASCII HEX | DEC | SHIFT S C A | KEY U S | FUNCTION (USER MODE) | FUNCTION (SYSTEM MODE) |
|---|---|---|---|---|---|
| $A0 | 160 | 1 0 0 | SPACE | WP fixed space | |
| $A1 | 161 | 1 0 1 | SPACE | | |
| $A2 | 162 | 1 1 0 | SPACE | | |
| $A3 | 163 | 1 1 1 | SPACE | | |
| $A4 | 164 | 1 0 0 | ENTER | reformat paragraph | |
| $A5 | 165 | 1 0 1 | ENTER | | |
| $A6 | 166 | 1 1 0 | ENTER | | |
| $A7 | 167 | 1 1 1 | ENTER | | |
| $A8 | 168 | 1 0 0 | TAB | previous field | |
| $A9 | 169 | 1 0 1 | TAB | | |
| $AA | 170 | 1 1 0 | TAB | | |
| $AB | 171 | 1 1 1 | TAB | | |
| $AC | 172 | | | | (unused) |
| $AD | 173 | | | | (unused) |
| $AE | 174 | | | | (unused) |
| $AF | 175 | | | | (unused) |
| $B0 | 176 | 0 0 0 | CAPSLOCK | caps lock toggle | |
| $B1 | 177 | 0 0 1 | CAPSLOCK | | |
| $B2 | 178 | 0 1 0 | CAPSLOCK | | |
| $B3 | 179 | 0 1 1 | CAPSLOCK | | |
| $B4 | 180 | 1 0 0 | CAPSLOCK | | |
| $B5 | 181 | 1 0 1 | CAPSLOCK | | |
| $B6 | 182 | 1 1 0 | CAPSLOCK | | |
| $B7 | 183 | 1 1 1 | CAPSLOCK | | |
| $B8 | 184 | 0 0 0 | F1 | help | |
| $B9 | 185 | 0 0 1 | F1 | enter system mode | system menu or |
| $BA | 186 | 0 1 0 | F1 | | |
| $BB | 187 | 0 1 1 | F1 | | |
| $BC | 188 | 1 0 0 | F1 | | boot system |
| $BD | 189 | 1 0 1 | F1 | | |
| $BE | 190 | 1 1 0 | F1 | | |
| $BF | 191 | 1 1 1 | F1 | | |

QL ASCII Decode Table ($C0-$DF)

| ASCII HEX | ASCII DEC | SHIFT S C A | KEY U·S | FUNCTION (USER MODE) | FUNCTION (SYSTEM MODE) |
|---|---|---|---|---|---|
| $C0 | 192 | 0 0 0 | F2 | | next partition |
| $C1 | 193 | 0 0 1 | F2 | | |
| $C2 | 194 | 0 1 0 | F2 | | |
| $C3 | 195 | 0 1 1 | F2 | | |
| $C4 | 196 | 1 0 0 | F2 | | previous partition |
| $C5 | 197 | 1 0 1 | F2 | | |
| $C6 | 198 | 1 1 0 | F2 | | |
| $C7 | 199 | 1 1 1 | F2 | | |
| $C8 | 200 | 0 0 0 | F3 | | grow partition |
| $C9 | 201 | 0 0 1 | F3 | | |
| $CA | 202 | 0 1 0 | F3 | | |
| $CB | 203 | 0 1 1 | F3 | | |
| $CC | 204 | 1 0 0 | F3 | | shrink partition |
| $CD | 205 | 1 0 1 | F3 | | |
| $CE | 206 | 1 1 0 | F3 | | |
| $CF | 207 | 1 1 1 | F3 | | |
| $D0 | 208 | 0 0 0 | F4 | | suspend program |
| $D1 | 209 | 0 0 1 | F4 | | |
| $D2 | 210 | 0 1 0 | F4 | | |
| $D3 | 211 | 0 1 1 | F4 | | |
| $D4 | 212 | 1 0 0 | F4 | | resume program |
| $D5 | 213 | 1 0 1 | F4 | | |
| $D6 | 214 | 1 1 0 | F4 | | |
| $D7 | 215 | 1 1 1 | F4 | | |
| $D8 | 216 | 0 0 0 | F5 | | |
| $D9 | 217 | 0 0 1 | F5 | | |
| $DA | 218 | 0 1 0 | F5 | | |
| $DB | 219 | 0 1 1 | F5 | | |
| $DC | 220 | 1 0 0 | F5 | | program reset |
| $DD | 221 | 1 0 1 | F5 | | |
| $DE | 222 | 1 1 0 | F5 | | |
| $DF | 223 | 1 1 1 | F5 | | |

**QL ASCII Decode Table ($E0-$FF)**

| ASCII HEX | ASCII DEC | SHIFT S C A | KEY U S | FUNCTION (USER MODE) | FUNCTION (SYSTEM MODE) |
|---|---|---|---|---|---|
| $E0 | 224 | 0 0 0 | <-- | cursor left | |
| $E1 | 225 | 0 0 1 | <-- | start of line | |
| $E2 | 226 | 0 1 0 | <-- | delete char. left | |
| $E3 | 227 | 0 1 1 | <-- | delete line | |
| $E4 | 228 | 1 0 0 | <-- | word left | |
| $E5 | 229 | 1 0 1 | <-- | pan window left | |
| $E6 | 230 | 1 1 0 | <-- | delete word left | |
| $E7 | 231 | 1 1 1 | <-- | | |
| $E8 | 232 | 0 0 0 | --> | cursor right | |
| $E9 | 233 | 0 0 1 | --> | end of line | |
| $EA | 234 | 0 1 0 | --> | delete character | |
| $EB | 235 | 0 1 1 | --> | del. to end of line | |
| $EC | 236 | 1 0 0 | --> | word right | |
| $ED | 237 | 1 0 1 | --> | pan window right | |
| $EE | 238 | 1 1 0 | --> | delete word right | |
| $EF | 239 | 1 1 1 | --> | | |
| $F0 | 240 | 0 0 0 | ↑ | cursor up | pan partition up |
| $F1 | 241 | 0 0 1 | ↑ | previous screen | |
| $F2 | 242 | 0 1 0 | ↑ | | |
| $F3 | 243 | 0 1 1 | ↑ | | |
| $F4 | 244 | 1 0 0 | ↑ | top of screen | |
| $F5 | 245 | 1 0 1 | ↑ | pan window up | |
| $F6 | 246 | 1 1 0 | ↑ | | |
| $F7 | 247 | 1 1 1 | ↑ | | |
| $F8 | 248 | 0 0 0 | ↓ | cursor down | pan partition down |
| $F9 | 249 | 0 0 1 | ↓ | next screen | |
| $FA | 250 | 0 1 0 | ↓ | | |
| $FB | 251 | 0 1 1 | ↓ | | |
| $FC | 252 | 1 0 0 | ↓ | bottom of screen | |
| $FD | 253 | 1 0 1 | ↓ | pan window down | |
| $FE | 254 | 1 1 0 | ↓ | | |
| $FF | 255 | 1 1 1 | ↓ | | |

## F.4 Summary of System Mode Commands

The following table shows the system command and shift key combinations for the QL keyboard in system mode. Key combinations marked 'unused' are untranslatable.

| | UNSHIFT | SHIFT | ALTMODE | CONTROL | CONTROL SHIFT | CONTROL ALTMODE | SHIFT ALTMODE | CONTROL SHIFT ALTMODE |
|---|---|---|---|---|---|---|---|---|
| ^<br>\| | PAN PARTN. UP | | | | | | | |
| \|<br>v | PAN PARTN. DOWN | | | | | | | |
| ESC | EXIT SYSTEM MODE | | UNUSED | UNUSED | UNUSED | UNUSED | UNUSED | UNUSED |
| F1 | | BOOT SYSTEM | ENTER SYSTEM MODE | | | | | |
| F2 | NEXT PARTN. | PREV. PARTN. | | | | | | |
| F3 | GROW PARTN. | SHRINK PARTN. | | | | | | |
| F4 | SUSPEND PROGRAM | RESUME PROGRAM | | | | | | |
| F5 | | KILL PROGRAM | | | | | | |

## F.5 Summary of User Mode Commands

The following table shows the recommended command and shift key combinations in user mode. Key combinations marked 'unused' are untranslatable, blank entries are available for applications software.

| | UNSHIFT | SHIFT | ALTMODE | CONTROL | CONTROL SHIFT | CONTROL ALTMODE | SHIFT ALTMODE | CONTROL SHIFT ALTMODE |
|---|---|---|---|---|---|---|---|---|
| <-- | CURSOR LEFT | WORD LEFT | START LINE | DELETE CHAR. LEFT | DELETE WORD LEFT | DELETE LINE | PAN WINDOW LEFT | |
| --> | CURSOR RIGHT | WORD RIGHT | END LINE | DELETE CHAR. | DELETE WORD RIGHT | DELETE TO EOL | PAN WINDOW RIGHT | |
| ↑ | CURSOR UP | TOP OF SCREEN | PREV. SCREEN | | | | PAN WINDOW UP | |
| ↓ | CURSOR DOWN | BOTTOM OF SCREEN | NEXT SCREEN | | | | PAN WINDOW DOWN | |
| TAB | TAB TO NEXT FIELD | TAB TO PREV. FIELD | UNUSED | UNUSED | | UNUSED | | |
| SPACE | SPACE | WP FIXED SPACE | UNUSED | UNUSED | | UNUSED | | |
| ESC | EXIT COMMAND SEQNCE. | COPY-RIGHT | UNUSED | UNUSED | UNUSED | UNUSED | UNUSED | UNUSED |
| CAPS LOCK | CHANGE CASE | | | | | | | |
| ENTER | NEW LINE | REFORM PARA. | UNUSED | UNUSED | | UNUSED | | |
| F1 | HELP | | | | | | | |
| F2-F5 | SELECT MENU OPTION | | | | | | | |

## F.6 Display File Manager Commands

DFM includes two-byte command codes within its display text. These can be used for three purposes:

(a) Within DFM itself as text formatting information.

(b) Within DFM to supply text output control data to screen driver.

(c) By user programs to insert binary data into a display data stream for any purpose whatever.

These commands have the general format:

Byte 1:

| | |
|---|---|
| bit 7 | 1 |
| bit 6 | 0 (system) or 1 (user) |
| bits 5-0 | command code |

Byte 2:

| | |
|---|---|
| bit 7 | 1 |
| bit 6 | 0 (system) or 1 (user) |
| bits 5-0 | command argument(s) |

Note that user commands are guaranteed to be transparent as far as 68K/OS is concerned. System commands are reserved for future 68K/OS expansion.

## F.7 DFM and Screen Driver Commands

The following commands are recognised by DFM and the screen driver and may be inserted in any text string processed by these routines:

(a) **Foreground Colour - ($80)**

Second byte:

| | |
|---|---|
| Bits 7-6 | 10 |
| Bits 5-3 | old foreground colour |
| Bits 2-0 | new foreground colour |

This command changes the foreground text colour as follows:

| | | | |
|---|---|---|---|
| 0 | black | 4 | green |
| 1 | blue | 5 | cyan |
| 2 | red | 6 | yellow |
| 3 | magenta | 7 | white |

The foreground colour of the following text is changed to the new colour up to the next colour command.

(b) **Background Colour - ($81)**

Second byte:

| | |
|---|---|
| Bits 7-6 | 10 |
| Bits 5-3 | old background colour |
| Bits 2-0 | new background colour |

This command changes the background text colour as follows:

| | | | |
|---|---|---|---|
| 0 | black | 4 | green |
| 1 | blue | 5 | cyan |
| 2 | red | 6 | yellow |
| 3 | magenta | 7 | white |

The background colour of the following text is changed to the new colour up to the next colour command.

(c) **Character Fount - ($82)**

Second byte:

| | |
|---|---|
| Bits 7-6 | 10 |
| Bits 4-3 | old fount |
| Bits 1-0 | new fount |

This command changes the character fount of the following text up to the next fount command. The default fount is fount zero.

(d) **Multiple Space - ($83)**

Second byte:

| | |
|---|---|
| Bits 7-6 | 10 |
| Bits 5-0 | number of space characters |

This command will generate the number of spaces requested on the screen. The cursor can only be positioned on the first space.

(e) **Underline On/Off - ($84)**

Second byte:

| | |
|---|---|
| Bits 7-6 | 10 |
| Bit 3 | old underline state (0 = off) |
| Bit 0 | new underline state (0 = off) |

This command switches character underlining on or off until the next underline command.

# APPENDIX G:

# DEVICE DRIVERS

# G DEVICE DRIVERS

## G.1 Overview

68K/OS device drivers are (as far as the user is concerned) called as subroutines via the IOSS. In fact a device driver may also consist of an interrupt routine, polled task, asynchronous program, or all three, communication between the various components being transparent to the user program.

This appendix gives a brief list of the characteristics of standard device drivers which are provided as part of the operating system. Drivers for add-on devices or special user written drivers can be loaded at any time, and accessed via standard IOSS calls (see Systems Programmer's Reference Manual).

## G.2 Keyboard Driver - KEY:

| | |
|---|---|
| Directory operations: | no |
| Reading: | yes |
| Writing: | no |
| Random access: | no |
| Double buffering | no |
| IOSPECIA operation: | no |

Note: CTRL/Z is treated as end of file, ALT/F1 switches to system mode and can therefore never be read from the keyboard by a user program.

## G.3 Screen Driver - SCREEN:

| | |
|---|---|
| Directory operations: | no |
| Reading: | no |
| Writing: | yes |
| Random access: | no |
| Double buffering | no |
| IOSPECIA operation: | no |

The SCREEN: device driver provides an interface to the screen for programs which only wish to use the screen as a simple sequential output device and which do not wish to drive the display file manager directly.

## G.4 Microdrive Filing System - MD:

| | |
|---|---|
| Directory operations: | yes |
| Reading: | yes |
| Writing: | yes |
| Random access: | yes |
| Double buffering | yes |
| IOSPECIA operation: | no |

The microdrive filing system is formally an IOSS device driver.

## G.5 RS232 Output Driver - TX1: and TX2:

There is one RS232 output driver for each line.

| | |
|---|---|
| Directory operations: | no |
| Reading: | no |
| Writing: | yes |
| Random access: | no |
| Double buffering | no |
| IOSPECIA operation: | yes |

The IOSPECIA call sends a break sequence or changes baud rate:

(a) D1.B < 0 Send a break which consists of a start bit of ~1 second

(b) D1.B >=0 Set the baud rate from the value of of D1 as follows:

| | | | |
|---|---|---|---|
| 0 | 19200 | 4 | 1200 |
| 1 | 9600 | 5 | 600 |
| 2 | 4800 | 6 | 300 |
| 3 | 2400 | 7 | 75 |

On power-up the baud rates for both lines are initialised to 9600 as this value suits most modern equipment.

## G.6 RS232 Input Driver - RX1: and RX2:

There is one RS232 input driver for each line.

| | |
|---|---|
| Directory operations: | no |
| Reading: | yes |
| Writing: | no |
| Random access: | no |
| Double buffering | no |
| IOSPECIA operation: | no |

A break condition on the line will give a status code of STIOERR, and will usually do so without losing data, but this cannot be guaranteed on the QL.

The line speeds for RS232 input are derived from the line speeds for output. One of the following will hold:

(a) only one speed is in use for all output and input on both lines

(b) two different speeds are used for output on the two lines, and no input is being performed

(c) any other situation will result in input being mangled.

These limitations are due to the hardware configuration and there is no sensible way to improve on them in software without catastrophic performance implications.

## G.7 Pipe Driver - PIPE:

| | |
|---|---|
| Directory operations: | no |
| Reading: | yes |
| Writing: | yes |
| Random access: | no |
| Double buffering | no |
| IOSPECIA operation: | no |

Pipes are the mechanism provided for programs to communicate and synchronise with each other using IOSS by providing an I/O channel from one applications program to another. A pipe is identified by its filename and many pipes may exist in the system at once.

## G.8 ROM Driver - ROM:

| | |
|---|---|
| Directory operations: | yes |
| Reading: | only IOLOAD |
| Writing: | no |
| Random access: | no |
| Double buffering | no |
| IOSPECIA operation: | no |

It is possible to store a number of procedures in ROM and execute these, either as procedures or as programs. The ROM: device driver exists permits IODIRINF, IOGETDIR and IOLOAD calls only.