GRiD OPERATING SYSTEM (GRiD-OS) REFERENCE

JUNE 1984

2535 Garcia Avenue
Mountain View, CA 94043
(415) 961-4800

Manual Name : GRiD Operating System (GRiD-OS) Reference
Order Number: 29200-44
Issue date: June 1984

TABLE OF CONTENTS

ABOUT THIS BOOK

This book is a programmer's guide to the GRiD Operating System (GRiD-OS). It defines all of the calls that can be made to the operating system from application programs and provides overviews of how the calls can be used. This book is not intended to be user's guide to the operating system. GRiD application programs all utilize the facilities of GRiD-OS and the Common Code routines to provide an easy-to-use, menu-driven interface so that the operating system is transparent to the user.

Chapter 1 introduces the operating system and lists all of the calls available to programmers and should be read to obtain on overview of the facilities that are available.

Chapters 2 through 5 discuss the various categories of system calls and are intended to give you some background information and general theory on how the calls interact with one another. These chapters should be read first before proceeding to the detailed descriptions of the OS calls if you are a new user of GRiD-OS.

Chapter 6 provides a detailed description of each system procedure and function. The calls are listed in alphabetic order to make this chapter easy to use as a reference document once you are generally familiar with the operating system.

CHAPTER 1. AN INTRODUCTION TO THE GRiD OPERATING SYSTEM (GRiD-OS)

The GRiD Operating System (GRiD-OS) has been designed to support GRiD's application programs, such as GRiDPLAN, GRiDPLOT, GRiDWRITE, and GRiDFILE, and to simplify the development of other application programs using any of the available languages (Pascal, PL/M, Assembly, FORTRAN, and C).

FEATURES OF GRiD-OS

GRiD-OS is essentially a resource management tool that simplifies memory management, and device and file management. Additionally, GRiD-OS supports multitasking and provides a variety of system utility services.

Table 1-1 summarizes the system calls provided by GRiD-OS and indicates the chapter where an overview of the calls is provided. Detailed descriptions of all system calls are provided in the alphabetically-ordered Chapter 6.

Table 1-1. A Summary of GRiD-OS Calls

PROCESSOR AND MEMORY MANAGEMENT (Chapter 2)	
OsCreateProcess	Creates a process by loading a program from mass storage.
OsForkProcess	Creates a process from a parameterless procedure.
OsDelay	Suspends execution of a process for a specified delay interval.
OsSetPriority	Changes the priority of a process.
OsWhoAmI	Returns the identification number assigned to a process.
OsDeleteProcess	Deletes another process from the system.
OsExit	Deletes the current process from the system and frees its resources.
OsReceive	Suspends execution of a process while it awaits a message.
OsSend	Sends a message to another process.
OsCreateSemaphore	Creates a semaphore for use with OsSignal and OsWait.
OsWait	Suspends execution of a process while it waits at a semaphore for an OsSignal.
OsSignal	Signals a semaphore to allow a waiting process to proceed.
OsDeleteSemaphore	Deletes a semaphore.
OsAllocate	Allocates from 1 to 64k bytes of memory to the requesting process.
OsFree	Deallocates a previously assigned block of memory.
OsGetSize	Returns the size of a particular block of memory.
OsGetMemStatus	Provides information about memory allocation.
DEVICE AND FILE MANAGEMENT (Chapter 3)	
OsAttach	Makes a device or file available to a program.
OsOpen	Allocates buffers for an attached file.
OsRead	Reads data from an open file.
OsWrite	Writes data to an open file.
OsSeek	Changes the point at which a subsequent access of an open file will begin.
OsTruncate	Deletes data from the end of an open file.
OsFlushAllBuffers	Writes the contents of memory buffers to all open files.
OsRename	Changes the name of an attached file.
OsChangeExtension	Changes only the extension portion of a file's name.
OsClose	Closes and deallocates memory assigned to a file.
OsDelete	Deletes an open file from the system.
OsDetach	Makes a file unavailable to programs.
OsGetStatus	Provides summary information about an attached file.
OsSetStatus	Establishes device specific status information associated with an attached file.
WINDOW GRAPHICS CALLS (Chapter 4)	
BaseLine	Returns the baseline position of the current font.

CharHeight	Returns the height of characters in the current font.
CharWidth	Returns the width of characters in the current font.
LineHeight	Returns the height of a character line in the current font.
WinInitDefaultWindow	Clears the window, resets the clipping rectangle, and (if specified) draws a one-pixel frame surrounding the screen.
WinSetWindow	Sets the window size to the rectangle it receives as an argument.
WinFrameWindow	Draws a one-pixel frame outside the current window bounds.
WinEraseWindow	Erases the contents of the current window.
WinScrollWindow	Scrolls the entire window in the given direction by the distance given in pixels.
WinGetWindowExtent	Returns the dimensions of the application's current window.
WinSetClip	Sets a clipping rectangle within the window boundaries.
WinResetClip	Resets the clipping rectangle to the entire window.
WinEraseRectangle	Erases a rectangle in the window.
WinInvertRectangle	Inverts the bit-map area inside a rectangle in the window.
WinCopyRectangle	Copies one rectangle into another rectangular area of the window.
WinScrollRectangle	Scrolls a rectangle in the given direction by the distance given in pixels.
WinSetAlternateWindow	Forces all subsequent window calls to be performed on the alternate window specified.
WinCopyRemoteRectangle	Copies a rectangle from one window to another.
WinAllocateWindowMemory	Allocates memory for an alternate window.
WinDrawChar	Draws a character in the window at a specified pixel location.
WinEraseChar	Erases a character position.
WinInvertChar	Performs an inversion of all the pixels of a character position.
WinDrawChars	Outputs a character string of a specified length from a text buffer to the screen.
WinLoadFont	Loads a font file into memory and returns a pointer to the font.
WinSetFont	Sets the designated (previously loaded) font as the current font.
WinDrawLine	Draws a line within the window.
WinEraseLine	Erases a line within the current window.
WinInvertLine	Inverts all the pixels on the given line.
WinDrawPixel	Draws a single pixel at the given window coordinate.
WinErasePixel	Erases a single pixel at the given window coordinate.
WinInvertPixel	Inverts a single pixel at the given window coordinate.

CONSOLE CALLS (Chapter 5)	
ConKeyPressed	Tells you if any key on the keyboard has been pressed.
ConCharIn	Waits for one character to be typed on the keyboard

	and then returns that character.
ConDefCsr	Turns the small cursor character "_" on or off.
ConResetDisplay	Clears the screen and displays the cursor character at the top, left hand position of the window.
ConMoveCsr	Moves the cursor to the specified x,y character position on the screen.
ConCharOut	Outputs the supplied character to the screen at the current cursor position.
ConLineOut	Outputs the number of characters specified by length to the screen.
ConLineIn	Inputs the number of characters specified by length from the keyboard.
ConPeekChar	Returns the first character in the keyboard queue.
ConHexOut	Outputs the supplied hexadecimal value to the screen at the current cursor position.
GENERAL UTILITY CALLS	
OsGetArgument	Scans and parses the contents of the command line or other designated buffer.
OsSwitchBuffer	Changes the buffer that OsGetArgument operates on.
OsOverlay	Brings a subprogram into memory.
OsGetSystemID	Returns system identification information.
OsGetTime	Returns all information from the system clock.
OsGetWork	Returns a pointer to the current "work" device used by compilers and the link program.
OsRegisterName	Registers a name and a small amount of associated information.
OsLookUpName	Looks for a name that has been previously registered and returns some information associated with that name.
OsDecodeException	Translates an exception number into an exception message.
OsHandleCancel	Specifies whether the system or application program will handle CODE-ESC.
OsMatchWildcard	Compares a target string, for example a file name, against a string containing wildcard characters.
OsCallDriver	Used by device drivers to pass device-specific requests.
OsAddDevice	Adds a device to the system's table of active devices.
OsRemoveDevice	Removes a device from the system's table of active devices.
OsPutProperty	Sets a system property such as screen frame or time in the User^Profile^ file.
OsGetProperty	Examines a system property such as screen frame or time in the User^Profile^ file.

DATA TYPES

The descriptions of the system calls in this book use the data types defined by Pascal-86 (which include some extensions beyond those of standard Pascal). Your calls must supply parameters meeting the specifications of these data types as defined in Table 1-2.

Table 1-2. Data Types for GRiD-OS Calls

Type	Description
Boolean	Simple ordinal with predefined values of False (0) and True (1).
Byte	An enumerated type defined (0..255).
Integer	Simple ordinal of two bytes in the range -32767 through +32767.
Char	A simple ordinal defined on the ASCII character set.
*Word	Simple ordinal of two bytes. Integers in the range 0 through 65535. Unsigned.
*LongInt	Simple ordinal of four bytes in the range -2,147,483,647 through +2,147,483,647.
Pointer	In Pascal, must be declared by the programmer as a pointer to some defined type.

* Indicates a Pascal-86 extension of standard Pascal data types.

ShortStrings

The ShortString type is simply a collection of bytes, the first of which tells you how many data bytes follow. It is used to describe any sequence of bytes where the first byte (also known as the "length" byte) represents the number of bytes (0 - 255) in the sequence (excluding the length byte). Thus, this type is often used to interface to operating system routines that require ASCII names (such as filenames) which don't have a standard length. ShortStrings are different from (and should not be confused with) strings found in the Common Code routines and in some versions of Pascal or C. ShortStrings can be defined as:

```
ShortString = RECORD
   length : Byte;
   Chars : [0..255] OF Char;
   END
```

When you call a GRiD-OS routine that expects a parameter formatted as a ShortString, you must pass the parameter by reference rather than by value. That is, you pass a pointer to the shortstring rather than passing the structure itself. Since many routines accept ShortStrings shorter than the maximum length of 255 bytes, you can often declare shorter versions of this type to save memory space.

Bytes

Pascal-86 defines one special data type to override standard Pascal's rigorous type-checking. It is the Bytes type. Note that this is not the Byte (singular) type. The Bytes type is not part of standard Pascal.

A parameter of a procedure or function outside of any module (such as a system call) can be defined to be of type Bytes. This lets you pass any type of variable as a parameter and bypass Pascal's normal type-checking. Regardless of the parameter type actually passed, it will always be passed by reference, not by value.

Some of the operating system routines require the Bytes type because it is not known ahead of time exactly which type will be used. Use great caution when passing a pointer variable as a parameter which is of the Bytes type. The Bytes parameter is thus acting as an untyped pointer. For example, OsFre and OsSend use Bytes parameters so that they can accept a pointer of any type. The pointer must be dereferenced (use ^ at the end) to ensure that the correct value is passed.

NOTE: The Bytes identifier can appear only in an external module's PUBLIC section: see the Pascal-86 manual for a detailed discussion of the Bytes type.

CHAPTER 2: PROCESSOR AND MEMORY MANAGEMENT FACILITIES

Two critical resources of the GRiD Compass are the central processing unit and system memory (RAM). Ultimately, the efficiency of the entire system depends on how efficiently you utilize the power of the central processor and how you manage the available memory.

PROCESSOR MANAGEMENT

GRiD-OS maximizes the power, or throughput, of the central processor by using a multi-tasking technique -- the various activities that the system must accomplish are broken into individual tasks or processes. You assign each process a priority which determines when it will be given the use of the central processor. This technique ensures that the central processor is never idle and is always working on the "ready" process with the highest priority. (We'll describe the precise meaning of "ready" in a few paragraphs.)

Typically, many of the processes in the system must interact with one another. They may have a need to share data, pass information back and forth, check on the availability of information from another process, wait for the occurrence of some external event, and so on. Therefore, in addition to scheduling process access to the central processor, GRiD-OS provides the synchronization mechanisms of message passing and semaphore signalling.

In this chapter, we will describe how processes are scheduled, how they are created, executed, and deleted, and how information is passed between processes.

WHAT IS A PROCESS?

A process is an executable entity consisting of some data and some executable code, and requiring its own set of registers and its own stack area. Since the GRiD Compass is a single processor system, only one process can be executing at any instant in time. However, many processes can be created and exist simultaneously within the system; GRiD-OS controls the scheduling and execution sequencing of multiple processes.

Process Scheduling -- An Overview

GRiD-OS performs process scheduling whenever any process issues any system call or when an event which a process has been 'waiting' for occurs. The scheduling technique used by GRiD-OS can be defined as priority-based, preemptive scheduling. It is called priority based since each process has a priority rating assigned to it when it is created. GRiD-OS examines this priority whenever it does process scheduling to determine which process should be the current, running process. The scheduling algorithm is called preemptive because GRiD-OS can preempt the current process whenever rescheduling occurs.

There are many system calls that affect process scheduling. We will give a brief overview here of the technique used by GRiD-OS to handle multiple processes. Details of the calls that affect scheduling are provided with the description of each call in Chapter 6.

The following illustration is a simplified state diagram showing the three possible states in which processes can exist. It also shows all possible transition paths that a process can traverse as it goes from one state to another:

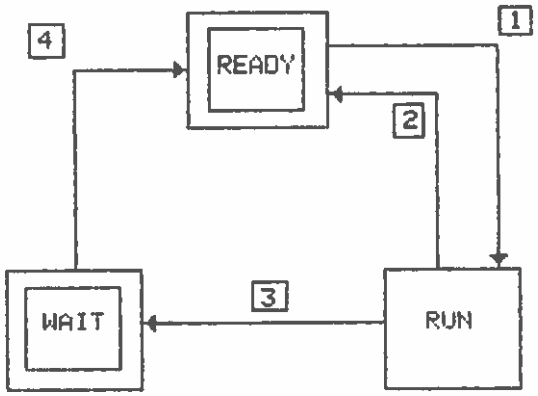


FIGURE 2-1. Process State Diagram

Since there is but one central processor in the GRID Compass system, only one process can actually be executing at any given moment. This process is sometimes referred to as the "current" process and its state is referred to as the "run" state. All other processes existing in the system are either in the "ready" state or the "wait" state.

Note in the preceding illustration that the run state is indicated by a single circle, while the ready and wait states are indicated by several concentric

circles. This convention indicates that there can be but one process in the run state, while there can be any number of processes in the ready and wait states.

All processes begin their existence in the ready state. Transition #1 to the run state occurs when the process becomes the ready process with the highest priority. The process remains in the run state until (transition #2) a process with higher priority enters the ready state, or (transition #3) the process must wait for some event to occur.

Note that GRiD-OS does its process scheduling whenever an event occurs that causes a waiting process to enter the ready state. Examples of these events are hardware interrupts, reception by waiting processes of messages or signals, or completion of timed waits by processes. Whenever any of these events occur, GRiD-OS examines the priorities of the ready processes and, if any of them is of a higher priority than the running process, the ready process will preempt the current process.

When the current process leaves the run state, via either transition path #2 or #3, the next ready process with the highest priority makes transition #1 and becomes the current process in the run state. If there are multiple ready processes with the same priority, they will be served (that is, become the current process) in a first-in, first-out fashion. Each time a system call is issued, the current process returns to the ready queue and the next ready process with an equally high priority becomes the current process.

Processes in the wait state remain there until the required event (for example, reception of a message) occurs. When the required event occurs, the process makes transition #4 to the ready state. If it happens to be of higher priority than the current process, the process that just returned to the ready state from the wait state would proceed to the run state immediately. Otherwise, it would just take its appropriate prioritized position among the other ready processes.

Creating, Deleting, and Executing Processes

The GRiD-OS calls listed below are the basic ones needed to bring a process into existence (Create, Fork), terminate a process (Delete, Exit), and directly affect the execution or running of a process (Delay, Set Priority).

- o OsCreateProcess - creates a new process by loading a program from a mass storage device.
- o OsForkProcess - creates a process who's code is already in memory
- o OsExit - terminates the current process
- o OsDeleteProcess - terminates a "forked" process
- o OsDelay - suspends execution of the current process
- o OsSetPriority - assigns new priority level to the current process

Each of these calls is described in detail in the alphabetically-ordered reference chapter (Chapter 6) of this manual.

MESSAGES -- SENDING AND RECEIVING

GRiD-OS provides two calls that let processes transfer messages between one another. The OsReceive call suspends a process while it awaits a message. The OsSend call delivers a message to a waiting process.

GRiD-OS delivers messages on a first-come, first-served basis. However, each message is addressed or sent to a specific receiving process. If the specified process is not currently waiting to receive a message, GRiD-OS holds all messages sent to that process and delivers them (one by one in the order received) when the process is receiving. Note that only one message can be received per each OsReceive call.

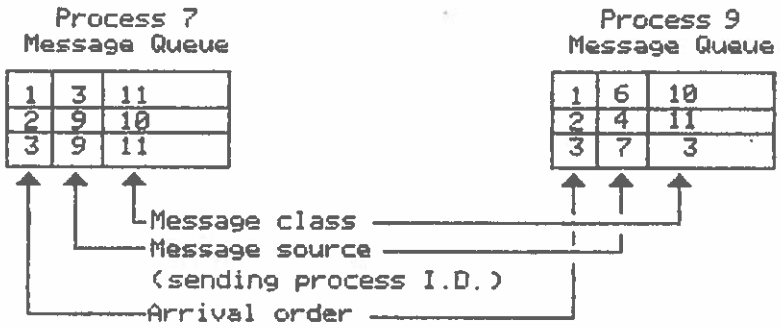
You can specify that a receiving process accept a message sent only by one specified sending process or that it accept a message sent to it by any other process.

Message Classes

Each message sent in the system also is of a user-specified class. The class parameter is a Word; therefore, you can have up to 65,536 different message classes. You can specify that a receiving process accept a message of only one specified class or that it accept a message regardless of class.

Message Transfer Example

The following figure illustrates the various options you have when transferring messages between processes:



This figure is a conceptualization of message queues maintained by GRiD-OS. The system maintains a separate message queue for each process that has a message sent to it. In this figure, GRiD-OS is holding seven messages that have been sent but which have not yet been received by the addressed processes: four messages for process 7 and three messages for process 9. Let's assume that process 7 issues an OsReceive call specifying that it will receive a message of class 1 from any process. When process 7 issues an OsReceive, GRiD-OS would immediately deliver message 4 from its queue that

was sent by process 5, and move process 7 to the ready state. Messages 1, 2, and 3 would not be delivered at this point because they are not of class 1.

After process 7 resumes execution, it issues an OsReceive specifying that it will accept a message of any class but only if sent by process 9. GRiD-OS will deliver message 2.

The send and receive calls can be issued in any sequence. That is, a process can issue OsReceive first and then go and wait for another process to send a message, or a process can issue OsSend to leave a message for a subsequent process to pick up using OsReceive.

Passing Notes

The message sent by the OsSend call is passed by reference rather than directly. The receiving process is given a pointer to the buffer where the actual message is contained. You can, however, deliver a short message more simply by directly sending a "note" via the OsSend call. One of parameters for OsSend is "note", a Word that is passed by value rather than by reference. If you have information to pass between processes that is 16 bits or less in length, you can use this note passing mechanism to send the information.

Message Format

GRiD-OS reserves the first 16 bytes for system use and requires that these 16 bytes of a message contain all zeros. The system, however, makes no other assumptions about the format of messages, nor does it place any restrictions on the message format. Since messages are passed by reference, their format and interpretation (after the 16-byte header of zeros) are left entirely up to the application.

CREATING AND USING SEMAPHORES

Semaphores let you synchronize the activities of multiple processes. They can be used to sequence the execution of a number of processes, to implement rapid responses to asynchronous events, and to provide a mechanism for mutual exclusion of processes.

In railroad terminology, a semaphore is a traffic signal that determines whether a train can enter a particular section of track. The semaphores provided by GRiD-OS perform an analogous function: they can cause processes to wait for a signal before proceeding to execute a section of code.

There are four system calls related to semaphores.

- OsCreateSemaphore - creates a semaphore
- OsDeleteSemaphore - deletes a semaphore
- OsWait - causes a process to wait for a signal
- OsSignal - lets a waiting process proceed

There can be as many semaphores in the system as you want: you're limited only by memory availability. Each semaphore you create is assigned a number (called the semaphore I.D, or "sid") by GRiD-OS.

You cause a process to stop at a semaphore by issuing an OsWait specifying the semaphore number to wait at. The process will wait at that semaphore until the semaphore is "not busy" (or until a specified period of time has elapsed). The OsSignal call is used to set a semaphore to the "not busy" condition. As soon as the semaphore is not busy, a process waiting at the semaphore can proceed.

When a waiting process is given the signal to proceed past a semaphore, its passage sets the semaphore to the busy condition. This prevents any other process that might be waiting at the semaphore from proceeding.

Any number of processes can be queued up waiting for the same semaphore. Each time the semaphore becomes not busy, another process is allowed to proceed. They are granted passage in order of their process priority. Alternatively, you can simultaneously signal all process that are waiting at a semaphore and allow them all to proceed to the ready state.

The functions that semaphores are used to accomplish could also be performed using the message passing facililties of GRiD-OS. Semaphores, however, execute much faster than message passing and are therefore a more efficient way of accomplishing process synchronization.

Semaphore Note Passing

In addition to signalling or waiting at a semaphore, the GRiD-OS semaphore calls let you pass a short message or "note" between signalling processes and waiting processes. One of the parameters for OsSignal is "note", a Word that is passed by value to the semaphore. The note will be given to the next process waiting at the semaphore. Interpretation of the contents of the note is application dependent.

MEMORY MANAGEMENT FACILITIES

The memory management facilities provided by GRiD-OS provide rapid allocation of memory while minimizing fragmentation which can produce small, essentially useless, blocks of memory. The technique used by GRiD-OS to accomplish these goals is a "first fit" approach.

Whenever GRiD-OS receives a request to allocate a block of memory, it starts at the beginning of its unallocated memory list and searches until it finds a free block of sufficient size to meet the request. The first block that it comes to that will fit the request is the one allocated to the process.

When a block of memory is freed, it is removed from the allocated list and added to the free list, which is stored in order of increasing addresses. A freed block of memory is automatically combined, or coalesced, with any adjacent free memory to form the largest contiguous block possible.

The GRiD-OS calls listed below are the ones related to memory management.

- o OsAllocate - allocates from memory to a process
- o OsFree - deallocates a block of memory
- o OsGetSize - returns the size of a block of memory
- o OsGetMemStatus - provides information about memory usage

Each of these calls is described in detail in the alphabetically-ordered reference chapter (Chapter 6) of this manual.

CHAPTER 3. DEVICE AND FILE MANAGEMENT FACILITIES

The GRID-OS file management system provides a uniform and straightforward interface to all system files regardless of the type of device that a file is associated with. Thus, you can access files throughout the system without concerning yourself with the characteristics and idiosyncracies of devices.

Additionally, GRiD-OS provides a "virtual" file system. The system can not only access local devices such as bubble memory and hard disks, it can also access remote devices such as GRiD Server (via GRiDLink or PhoneLink) and GRiD Central (via PhoneLink). The application programmer doesn't have to write any special code to use these devices; GRiD-OS handles them transparently.

The organization of the file system is illustrated in Figure 3-1. A hierarchical, three-level structure is utilized. System devices comprise the uppermost level. Within each system device are any number of directories or "subjects", and within each subject are any number of "titles".

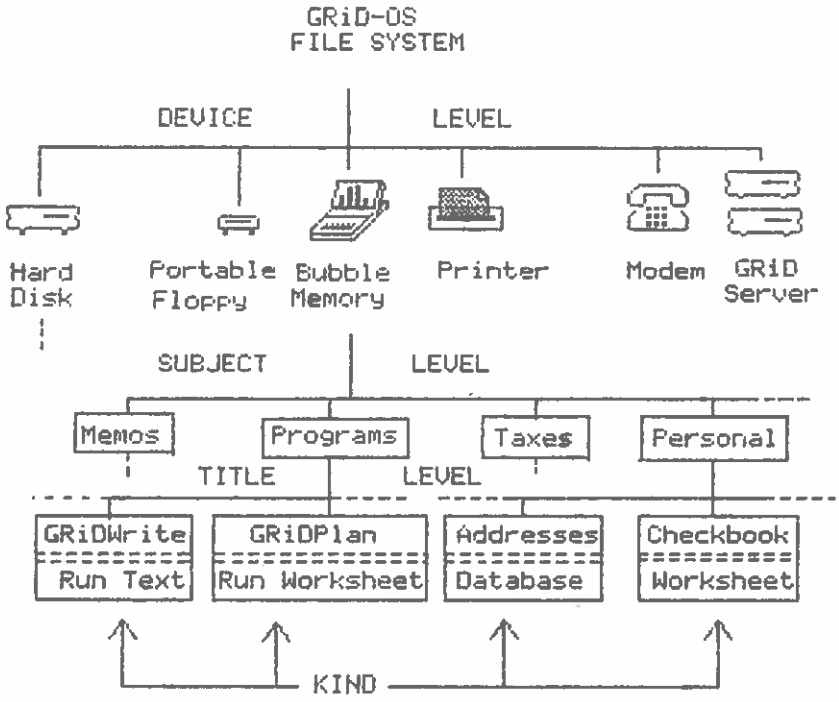


Figure 3-1. GRID-OS File System Organization

PATHNAMES

A file is fully identified by specifiying its "pathname". A pathname defines the route to take when accessing a file; that is, it specifies the device and subject where a title is located. To a programmer, the complete pathname schema is as follows:

```
'device'subject'title~kind~!password
```

Three delimiter characters are used in pathname specifications. The left single quote or "tick" (') -- ASCII code 60 hex -- must precede device, subject, and title names. This character is generated on the Compass keyboard by pressing CODE-'. The tilde (~) -- ASCII code 7E hex -- must precede and follow the kind. This character is generated on the keyboard by pressing CODE-;. The vertical bar (!) -- ASCII code 7C hex -- must precede the password. This character is generated on the keyboard by pressing CODE-SHIFT-;.

NOTE: These delimiter characters were purposely chosen for their obscurity so that end-users could have most commonly used characters available when naming their files.

If a kind is not supplied as part of the pathname, the system uses a default kind of "Untyped". The password portion of the pathname is optional. If a file was created with a password, then the password must be included as part of the pathname.

If you specify a pathname that does not begin with the tick, the system assumes that the first name it encounters is the title and that you have left off the device and subject names. The search for the title will then be limited to the current directory, that is, to the current 'device'subject'.

If you provide the complete pathname including device, subject, and title, the search for the file will begin at the top of the virtual device tree (see Figure 3-1) -- if the title is anywhere in the system, it will be located.

The maximum length of subject and title names is 80 characters each. Subject and title names can consist of any printing characters (including spaces) except the following:

- ' left single quotation mark ("tick")
- ~ tilde
- | vertical bar

The maximum length of a title includes its two optional extensions of kind and password.

DEVICES

The devices currently included in the file system are identified as follows:

Bubble Memory	The nonvolatile, mass storage built into the GRiD Compass.
bb	Bit-Bucket (or "byte-bucket") is a null device used primarily as a dummy device for testing. Data written to the bit-bucket is accepted and then simply disappears. Read operations directed to the bit-bucket return an end-of-file.
ci	Console Input. The keyboard of the GRiD Compass.
co	Console Output. The screen of the GRiD Compass.
Extra Floppy Disk	The floppy disk in a system's second 2101 disk drive.
Extra Hard Disk	A system's second 2101 disk drive.
Floppy Disk	The floppy disk drive in a 2101 disk unit.
GPIB	General Purpose Interface Bus. The IEEE-488 connector

	on the GRiD Compass. This device provides access to all devices that attach to the GPIB connector.
Hard Disk	2101 hard disk (Winchester) mass storage device.
Modem	The 212/103 modem built into the GRiD Compass.
Plotter	The plotter currently attached to the system.
Portable Floppy	2102 portable floppy disk drive.
Printer	The printer currently being used with the system.
Serial	The serial input/output port of the GRiD Compass.
Work	A temporary file used by many programs (for example language translators, and the linker) to store intermediate results.

Not all of the devices participate fully in the hierarchical structure of the file system. Only mass storage devices (Bubble Memory, Hard Disks, Floppy Disks) can hold subjects and titles. Devices such as the printer or the GRiD Compass screen, can have files sent to them, but the files can obviously not be retrieved from such devices. Devices can also be remote, such as disks and printers provided by GRiD Server.

SUBJECTS AND TITLES

A title (or file name, as it is sometimes called) is a name given to a file which might consist of a program, pure data, or some combination of code and data. Subjects (or directories, as they are sometimes called) are also considered to be files -- but a subject is a file whose contents always consist of a collection of titles.

All subject names on a particular device are unique and all title names within a particular subject are unique. That is, you can have multiple occurrences of the same title (identical title name) so long as each occurrence is associated with a different subject and you can use the same subject on different devices. Thus, for example, a file titled DataFileA could exist on both Bubble Memory and Hard Disk. Or DataFileA could reside on the Hard Disk under Subject1 and Subject2.

FILE KINDS

The file kind (sometimes referred to as file "type") extension to titles lets you give several related files the same title while assigning them different "kind" characteristics. Interpretation of the kind extension is left up to the application.

You can directly examine and change the kind extensions of files using the OsChangeExtension call described in Chapter 6.

AN OVERVIEW OF FILE MANAGEMENT CALLS

All of the GRID-OS calls that are related to the file management system are described in detail in Chapter 6. In this chapter we will provide an overview of the interactions and usage of these system calls.

Figure 3-2 illustrates the relationship of the various activities performed in the file management system.

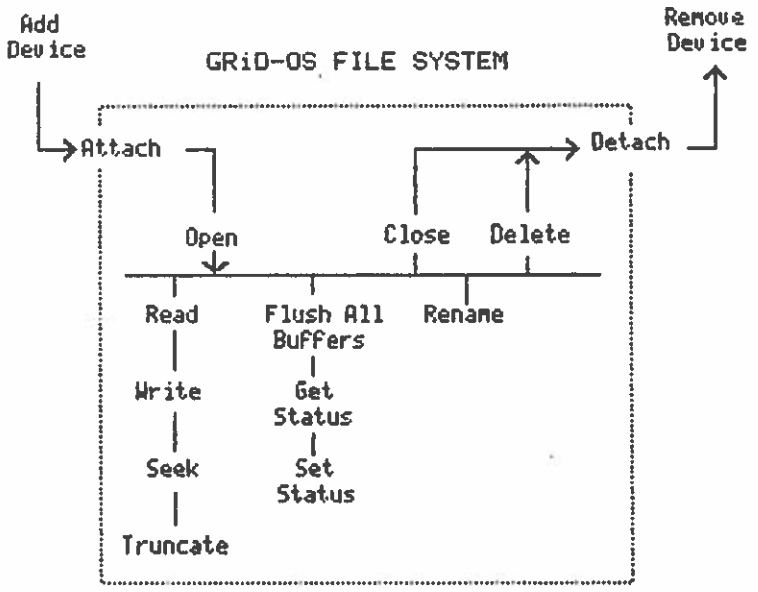


Figure 3-2. An Overview of File Management Calls

Operating on Files

In order for a file to exist in the file system, it must be associated with a device. Most files are associated with, and reside on disk or bubble. However, you can also send files to printers, plotters, or the screen. Before performing a transaction such as attach, open, read or write on a file, the operating system must have the approprate device included in its table of active devices. When the system is powered on, GRiD-OS goes around the system and adds all devices currently connected to the active device table. If a device is subsequently added, you inform GRiD-OS of this event using the OsAddDevice call. You can remove a device from the active device table, in order to free the memory used by its driver, using the OsRemoveDevice call.

Once the appropriate device is in the system, a file is connected to the system with the OsAttach call. If the file does not already exist, it is created by OsAttach. After a file is attached, you must open the file (using OsOpen) before you can perform any activity on the file.

GRiD-OS uses the two-step sequence of attach/open to increase efficiency. The attach locates the file in the system but allocates no buffer space. Allocation of buffer space occurs only when a file is opened. Attaching a file is a relatively time consuming process but uses little memory space. Opening a file is quite fast but consumes more memory. Thus, a program can keep many files attached but, by having only one file at a time open, wastes no buffer space. When the program needs to access a file, that single file can then be quickly opened.

After a file is open, you can access the contents of the file using the OsRead, OsWrite, OsSeek, and OsTruncate calls. These calls are the ones that actually access the contents of files. They let you read the contents of a file (OsRead), alter the contents of a file (OsWrite), change the point of access within a file (OsSeek), and delete a portion of a file.

You can terminate file access using OsClose and then reopen the file without reattaching it. You can sever the connection to a file with OsDetach or you can detach and delete the file from a device with OsDelete.

You can change the title of a file using the OsRename call and you can examine and alter the system characteristics of a file using the OsGetStatus and OsSetStatus calls. The OsFlushAllBuffers call writes the contents of temporary system buffers in memory out to the device where the file is permanently stored.

Note that the OsChangeExtension call does not actually operate within the file system. It is really just a string function that changes a string which contains the pathname of the file; the extension of the file in the file system remains unaltered.

Current File Position Marker

Each open file has a "current file position" marker associated with it. When a file is first opened, this marker is at the first byte (byte zero) of the file. Whenever you access a file, you specify the number of bytes that are to be accessed. As part of that access, the current file position marker is moved so that it is just beyond the last byte accessed and thus indicates the first byte available to a subsequent access. You can directly move the current file position marker with OsSeek which does not read or write any data in the file. Note, however, that you can not insert data in the middle of a file. If you move the current file position marker to somewhere within a file and then perform a write, data will be written over pre-existing data.

Operating on File Directories

GRiD-OS lets you read the contents of directories just as you do with any other file. The only difference is that instead of reading a byte at a time, one directory entry at a time is read.

To prepare a file to be read in "directory mode" you simply use the OsAttach call specifying a directory access mode. The system automatically attaches

the specified file and also performs an OsOpen preparing the file for subsequent read or seek operations. Obviously, the file being accessed in this mode should be a directory (a file with a Kind of "Subject") for these operations to be meaningful.

NOTE: The password assigned by GRiD-OS to all directory files is "GRiDiRG".

Now, OsRead operations or OsSeek operations performed on the file, instead of treating bytes as their object, treat each directory entry as the object. Thus, a read with a length of three would return three directory entries (either partial or complete entries) instead of three bytes of data.

You can read either a partial directory entry containing just the name of the directory file, or the complete directory entry giving all the information about the characteristics of a file. The format for partial directory entries is as follows:

```
PartialDirEntryType = RECORD
                        dummy : ARRAY [1..8] OF Char;
                        length : Byte;
                        name : ARRAY [1..1] OF Char;
                      END;
```

Note that the size of the name portion of the entry is defined by the length parameter. The name length is variable and can be up to 80 bytes long. If information beyond the name of a directory entry is required, you can read complete directory entries whose format is as follows:

```
CompleteDirEntryType = RECORD
                         dummy : ARRAY [1..8] OF Char;
                         length : Byte;
                         name : ARRAY [1..80] OF Char;
                         creationDate : ARRAY [1..11] OF Char;
                         unused1 : Word
                         lastModDate : ARRAY [1..11] OF Char;
                         expirationDate : ARRAY [1..11] OF Char;
                         unused2 : ARRAY [1..25] OF Char;
                         uses8087 : Boolean;
                         version1 : Byte;
                         version2 : Byte;
                         unused3 : ARRAY [1..15] OF Char;
                         version3 : Byte;
                         propertyLength: LongInt;
                         unused4 : ARRAY [1..28] OF Char;
                       END;
```

When you read a complete directory entry, the information returned will be of a fixed length since all fields in each record are filled out to their maximum length. For example, the name returned will always be 80 characters in length. The significant or used portion of the name field will be indicated by the length parameter; the rest of the name field will be filled out with blanks to occupy the full 80 characters of the field.

CHAPTER 4. WINDOW GRAPHICS

The window graphics routines are a set of procedures that let applications display text and graphics on the screen.

With these routines, you can

- Create an application that runs completely independent of the physical screen size or characteristics.
- Draw, erase, or invert text characters, pixels, lines, and rectangles.
- Clip the display within a clipping rectangle so that data outside the rectangle is not displayed.
- Perform bit-by-bit scrolling within a display window, to change the display rapidly.
- Establish alternate windows and pass data from one window to another.

SETTING UP WINDOWS

A window defines an area of the screen (often the entire screen) that can subsequently be referenced by other graphic calls to display information consisting of rectangles, text, lines, and pixels. GRiD-OS lets you maintain more than one window at a time. You can have more than one window image in memory at a time and switch from one window to another.

Once you define a window, only those portions of information sent to that window that do not extend beyond the window boundaries are displayed on the screen -- the window "clips" information that would be outside of the defined window.

A window is defined using the calls described below.

- o WinInitDefaultWindow -- Clears the window and resets the clipping rectangle.
- o WinSetWindow -- Sets the window to a specified size.
- o WinFrameWindow -- Draws a frame around the window
- o WinEraseWindow -- Erases the contents of the current window.
- o WinScrollWindow -- Scrolls the entire window a specified distance and direction.
- o WinGetWindowExtent -- Returns the size of the current window.

ALTERNATE WINDOWS

There can only be one window at a time displayed on the screen. This window is called the "current window". GRiD-OS, however provides calls that let you maintain multiple windows in memory. There are two reasons why you might want to have alternate windows: to redirect the contents of one window to another window or to convert screen image files stored in GRiD's format to the format needed by a screen other than that of the Compass computer.

The alternate window calls make the window routines compatible with other computer systems regardless of the screen size or formats of screen image data as stored in memory. The calls also let you maintain multiple windows in memory and bring window contents to the screen or dismiss them from the screen to display other windows.

A window is established in memory with the WinAllocateWindowMemory call. This call specifies the characteristics of the window such as format, size, bits/pixel, and so on. Each program has a "current" window where the program's calls are directed to perform such operations as drawing lines and displaying characters.

A program establishes alernate windows with separate WinAllocateWindowMemory calls for each window and specifies which window is the current window with the WinSetAlternateWindow call.

You can copy information from one window in memory to another with the WinCopyRemoteRectangle call. This call can place the entire contents or any portion of one window into another window.

Applications that use screen image files or that need to manipulate the screen directly must use alternate windows in order to work with computers having screen chatacteristics that differ from those of the Compass computer. For example, a screen image stored on disk or in bubble memory must first be read into a window that is specified as being in GRiD format. You can then copy the contents of that window (with WinCopyRemoteRectangle) to one specified as being in host screen format so that the image will be properly displayed.

The calls used with alternate windows are as follows:

- o WinAllocateWindowMemory -- Allocates memory for an alternate window.
- o WinSetAlternateWindow -- Causes all subsequent window calls to operate on an alternate window.

- WinCopyRemoteRectangle -- Copies a rectangle from one window region to another and converts display data (if necessary) to the format required by the destination window.

CLIPPING RECTANGLES

Within each window, you can further define rectangular areas that have specific characteristics. You can have multiple rectangles within a single window and can manipulate all of the pixels in each rectangle separately from other rectangles in the window. A rectangle can also clip information at its boundaries just as a window does. The calls that operate on rectangles are as follows:

- WinSetClip -- Sets a clipping rectangle within a window.
- WinResetClip -- Resets the clipping rectangle to the entire window.
- WinEraseRectangle -- Erases the contents of a rectangle.
- WinInvertRectangle -- Inverts the contents of a rectangle.
- WinCopyRectangle -- Copies one rectangle into another.
- WinScrollRectangle -- Scrolls a rectangle a specified distance and direction.

TEXT GRAPHICS

These routines complete characters within defined window locations. For a further discussion of character formation, see the discussion of character fonts that follows.

- WinDrawChar -- Draws a character at a specified location in the window.
- WinEraseChar -- Erases a character at a specified location in the window.
- WinInvertChar -- Inverts a character at a specified location in the window.
- WinDrawChars -- Outputs a character string to the screen at a specified location in the window.

CHARACTER FONTS

The character spacing used when displaying text on the screen depends on which font is the "current" font. Two calls are used to handle fonts: WinLoadFont loads a specified font into memory and WinSetFont determines which of the currently loaded fonts is to be the current font used to display characters on the screen. You can determine the characteristics (character size, spacing, and so on) of a font by examining the FontInfoRecord associated with each font (see WinSetFont) or by using the functions Baseline, Charheight, Charwidth, and Lineheight.

The standard (built-in) font contained in ROM for the Compass computer has the following characteristics (values listed are in number of pixels):

```
charWidth  (= 6)
charHeight (= 8)
```

lineHeight (= 10)
baseLine (= 7)

Figure 4-1 illustrates how the dimensions of font characters are measured. Note that there are other fonts available besides the built-in font depicted in this figure.

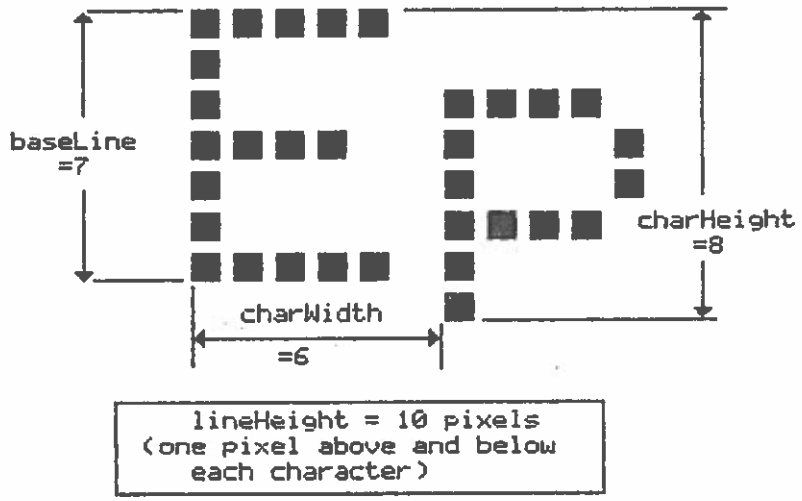


Figure 4-1. Built-In Character Font Dimensions

Even though each character displays as 5 pixels across, the actual font is 6 pixels wide. There is a blank pixel at the right edge of each character thus leaving a single pixel space between characters. While each character is eight pixels high, the line height is 10 pixels altogether, which leaves a two pixel vertical spacing between lines of text.

The value lineHeight represents character height plus the amount of space between lines. Each line has a one-pixel space above it and below it. The value baseLine represents the distance from the top of the character to the line connecting the bottom of the capital letters.

- WinLoadFont -- Loads a font file into memory.
- WinSetFont -- Sets a previously loaded font as the current font.

LINE GRAPHICS

These routines let you manipulate lines within defined windows. Lines are defined by defining their two end points (pixel coordinates).

- o WinDrawLine -- Draws a line within the window.
- o WinEraseLine -- Erases a line within the current window.
- o WinInvertLine -- Inverts a line within the window.

PIXEL GRAPHICS

These calls let you manipulate a single pixel within a window. Note: usually, the Line graphic calls can be used to accomplish detailed graphics. However, the pixel graphic calls are provided to give you completely detailed control of the display.

- o WinDrawPixel -- Draws a single pixel at a specified coordinate.
- o WinErasePixel -- Erases a single pixel at a specified coordinate.
- o WinInvertPixel -- Inverts a single pixel at a specified coordinate.

COORDINATE SYSTEM

The window calls draw pixels, characters, and lines on the screen within a window coordinate system. All coordinates are ultimately designated with absolute pixel locations on the screen by the system. However, almost all window routines use relative pixel coordinates that are based on the current window instead of absolute screen locations. Each window has its own coordinate system with 0,0 at its top left.

Each coordinate refers to one pixel. Thus, drawing a line from (0, 0) to (0, 1) will cause two pixels to be displayed. Drawing a line from one pixel to itself causes a single pixel to be displayed.

DATA STRUCTURES

```
* TYPE Point = RECORD x,y: Integer END;
```

Point is a record of x,y pixel coordinates which are either absolute or relative to the window. Points can represent two-dimensional positioning offsets or window dimensions, as well.

```
* TYPE Rectangle =
        RECORD topLeft, extent: Point END;
```

Rectangle is a record of two Points. The variable topLeft defines the pixel coordinates of the upper left corner of a rectangle. The x coordinate of extent defines the width of the rectangle in pixels; the y coordinate of extent determines the height of the rectangle.

```
*TYPE Direction = (up, down, left, right);
```

Defines a direction for scrolling rectangles and windows on the screen display.

```
* TYPE
     WindowFormat = (screenFormat, GRiDFormat);

     WindowRegion = RECORD
                      format : WindowFormat;
                      width : Word;
                      height : Word;
                      bufLength : Word;
                      buf : Pointer;
                      bitsPerPel : Byte;
                      bytesPerLine : Word;
                    END;
```

format -- GRiD format or host screen format.
width -- the width of the window in pixels.
height -- the height of the window in pixels.
bufLength -- the size, in bytes, of the buffer allocated by the system for this window.
buf -- a pointer to the first byte of the buffer allocated for this window.
bitsPerPel -- the number of bits-per-pixel used for the window. For GRiD format windows, there is one bit per pixel.
bytesPerLine -- the number of bytes used by the system to store one horizontal line of pixels for the allocated window.

CHAPTER 5. CONSOLE ROUTINES

These routines give you direct access to the screen and keyboard of the Compass Computer. All of the routines that output information to the screen operate within the current window (see Chapter 4). They differ from the related window graphic routines such as WinDrawChar because they treat the window as a virtual console. Thus, while characters output by WinDrawChar will be clipped when they reach the window or clipping rectangle boundaries, characters output by the console routines will wrap to the next line within a window when a boundary is reached. All GRiD application programs display text on the screen using the window routines.

The following three console routines are useful for obtaining input from the Compass keyboard:

ConKeyPressed	Tells you if any key on the keyboard has been pressed.
ConCharIn	Waits for one character to be typed on the keyboard and then returns that character.
ConPeekChar	Returns the first character in the keyboard queue.

The remaining console routines are provided primarily to be compatible with the interface requirements of the compilers and other Intel development tools. They are rarely used within GRiD applications but can be handy during debugging.

ConDefCsr	Turns the small cursor character "_" on or off.
ConResetDisplay	Clears the screen and displays the cursor character at the top, left hand position of the window.
ConMoveCsr	Moves the cursor to the specified x,y character position on the screen.

ConCharOut	Outputs the supplied character to the screen at the current cursor position.
ConLineOut	Outputs the number of characters specified by length to the screen.
ConLineIn	Inputs the number of characters specified by length from the keyboard.
ConHexOut	Outputs the supplied hexadecimal value to the screen at the current cursor position.
GetConsoleState	Returns information describing the current state of the console, such as cursor location and last character printed to the screen.

CHAPTER 6. GRiD-OS PROCEDURES AND FUNCTIONS

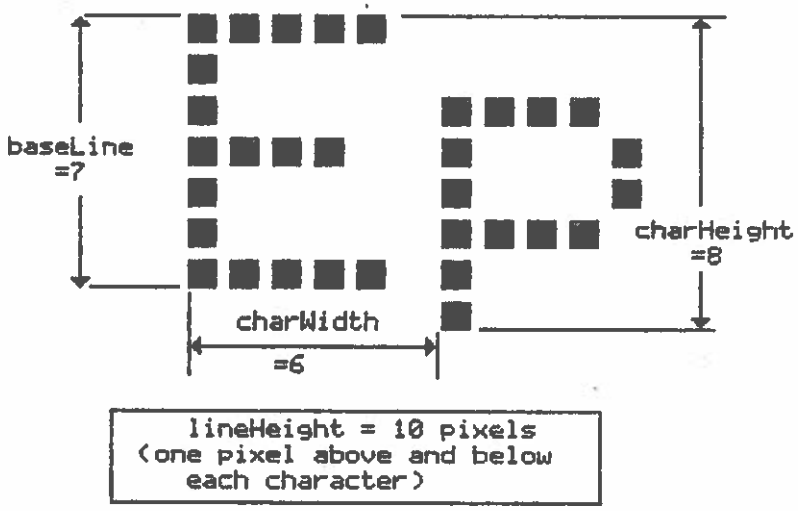
This chapter lists all of the procedures and functions provided by GRiD-OS in alphabetical order. For discussions of concepts and interactions of these calls, refer to the appropriate chapter earlier in this manual. This chapter simply lists the calls in alphabetical order and provides a comprehensive description of each call for maximum ease-of-use for reference purposes.

BaseLine

FUNCTION BaseLine : Integer;

Purpose and Operation

This routine returns an integer that is baseline of the current font. The baseline is the distance from the top of a character to the line where the bottom of capital letters are formed. Baseline is also the line where the tip of the cursor is positioned.



CharHeight

FUNCTION CharHeight : Integer;

Purpose and Operation

This routine returns an integer that is the height (in pixels) of the characters in the current font. See the BaseLine function for a figure illustrating CharHeight.

CharWidth

```
FUNCTION CharWidth : Integer;
```

Purpose and Operation

This routine returns an integer that is the width (in pixels) of the characters in the current font. See the BaseLine function for a figure illustrating CharWidth.

ConCharIn

```
FUNCTION ConCharIn: Char;
```

Purpose and Operation

This routine waits for one character to be typed on the keyboard and then returns that character. (See Appendix A for a table showing the values returned by each keystroke.)

This routine actually returns a 16-bit word. The low order byte contains the 8-bit value representing the key that was pressed. The high-order byte of the word provides the following additional information from the keyboard:

Bit #	Interpretation
12	Set to 1 if a repeated character
13	Set to 1 if SHIFT key also depressed
14	Set to 1 if CODE key also depressed
15	Set to 1 if CTRL key also depressed

If you want to receive this additional information, you must change the ConCharIn function declaration (in the include file ConPas.Inc) to "FUNCTION ConCharIn : Word;" in order to have the full 16-bit value returned.

ConCharOut

```
PROCEDURE ConCharOut (ch: Char);
```

Purpose and Operation

This routine outputs the supplied character to the screen at the current cursor position.

ConDefCsr

```
PROCEDURE ConDefCsr (on: Boolean);
```

Purpose and Operation

This routine turns the small cursor character "_" on or off. If the parameter "on" is True, the cursor will be displayed; if False, the cursor will not be displayed.

ConHexOut

```
PROCEDURE ConHexOut (num: Word);
```

Purpose and Operation

This routine outputs the hexadecimal number specified by num to the current cursor position.

ConKeyPressed

```
FUNCTION ConKeyPressed: Boolean;
```

Purpose and Operation

This routine returns a Boolean True if any key on the keyboard has been pressed. If you want to determine which key is depressed, rather than just the fact that a key was depressed, use the function ConCharIn or ConPeekChar.

ConLineIn

```
FUNCTION ConLineIn (VAR buffer: Bytes;
                    maxLength: Word): Word
```

Purpose and Operation

This routine inputs characters from the keyboard and places them in the designated text buffer. The keyboard entry is terminated either when the number of characters specified by maxLength has been entered or when the RETURN key is pressed. The function returns a word indicating the actual number of characters returned (including the terminating CR/LF if less than maxLength characters read).

Parameters

buffer -- the text buffer where the characters are to be stored.
maxLength -- the maximum number of characters to input.

ConLineOut

```
PROCEDURE ConLineOut (VAR buffer: Bytes;
                      length: Word);
```

Purpose and Operation

This routine outputs the number of characters specified by length to the window. If the window or clipping rectangle boundary is reached, the characters wrap around to the next line.

Parameters

buffer -- the text buffer where the characters are to be output are stored.
length -- the number of characters to output.

ConMoveCsr

```
PROCEDURE ConMoveCsr (x, y: Byte);
```

Purpose and Operation

This routine moves the small cursor character "_" to the specified x,y character position (not pixel position) in the current window: 0,0 is the upper left corner.

Parameters

x,y -- the character row and column position relative to your window (not the screen) where the cursor is to be positioned.

ConPeekChar

```
FUNCTION ConPeekChar: Char;
```

Purpose and Operation

This routine returns the first character in the keyboard queue. If the queue is empty, this routine waits until a key is pressed. (See Appendix A for a table showing the values returned by each keystroke.)

This routine actually returns a 16-bit word. The low order byte contains the 8-bit value representing the key that was pressed. The high-order byte of the word provides the following additional information from the keyboard:

Bit #	Interpretation
12	Set to 1 if a repeated character
13	Set to 1 if SHIFT key also depressed
14	Set to 1 if CODE key also depressed
15	Set to 1 if CTRL key also depressed

If you want to receive this additional information, you must change the ConPeekChar function declaration (in the include file ConPas.Inc) to "FUNCTION ConPeekChar : Word;" in order to have the full 16-bit value returned.

ConResetDisplay

```
PROCEDURE ConResetDisplay;
```

Purpose and Operation

This routine clears the current window, moves the small cursor character "_" to the top, left hand position (0,0) of the window, and turns the small cursor character "_" on.

LineHeight

```
FUNCTION LineHeight : Integer;
```

Purpose and Operation

This routine returns an integer that is the height (in pixels) of the character lines (character height plus the spacing between lines) in the current font. See the BaseLine function for a figure illustrating LineHeight.

GetConsoleState

```
FUNCTION GetConsoleState : ConsoleStatePtr;
```

Purpose and Operation

This routine returns a pointer to a record describing the current state of the console. The organization of the console state record is as follows:

```
ConsoleStateType = RECORD
                     xLoc : Integer;
                     yLoc : Integer;
                     cState : Byte;
                     scroll : Byte;
                     curChar : Byte;
                     upperFlag : Byte;
                     NMIFlag : Byte;
                   END;

   ConsoleStatePtr = ^ConsoleStateType;
```

ConsoleStateType Record Fields

xLoc, yLoc -- the current cursor location (see ConCharOut) indicating where the last character was drawn on the screen.
cState -- cursor state. This is 1 if cursor is on and 0 if cursor is off.
scroll -- an internal variable used by the window routines.
curChar -- current character. The character last printed to the screen.
upperFlag -- internal flag indicating whether the upper case keylock (SHIFT-ESC) is set on.
NMIFlag -- an internal variable used by the window routines.

OsAddDevice

```
PROCEDURE OsAddDevice (VAR pathName : Bytes;
                       VAR name : Bytes;
                       VAR entryPoint : Bytes;
                           intAddr : Byte;
                           mass : Boolean;
                           mode : Byte;
                       VAR error : Word);
```

Purpose and Operation

This call adds a device to the Active Device Table maintained by GRiD-OS. The device can then be accessed by programs just as though it were another file. Thus, this call is the equivalent of "activating" a device from the command line (see the Program Development Guide for a description of the Activate program). This call can add a device that is in a file, or that is linked into a program or can make a second copy of an already activated device. For a detailed discussion of how to use this call, refer to GRiD documentation on device drivers.

Parameters

pathName -- the pathname (formatted as a ShortString) of the file containing the device driver or the name of device already listed in the Active Device Table, depending on the setting of bit 0 of the mode parameter. If the driver is linked into the currently running program, this parameter should be NIL.

name -- the actual name (formatted as a ShortString) of the driver as it will be listed in the Active Device Table. (Note: the specified name should not have a backquote (`) in front of it.) If this parameter is NIL, the title part of the pathName parameter is used as the device name.

entryPoint -- the name of the device driver main procedure if the driver is linked into the currently running program. In this case, the pathName parameter is ignored. If the driver is not linked into the program, the entryPoint parameter should be NIL.

intAddr -- the interface address (usually, the device's GPIB address. If this device is not a GPIB device, set intAddr to NULLBYTE (0FFh).

mass -- a Boolean that, if TRUE, indicates that the device being added is a mass storage device such as a hard disk, floppy disk, or bubble.

mode -- the bits of this Byte determine various attributes of the device being added as follows:

Bit #

0 -- driver location. If set to 0, the pathName parameter specifies the device driver location. If set to 1, the pathName parameter is the name of a device already in the Active Device Table.

1 -- visible/invisible. If set to 0 and the mass parameter is TRUE, the device will appear on active device list and be displayed on the File form. If set to 1 or if the mass parameter is FALSE, the device will be invisible.

2 -- local/remote. If set to 0, the device is local. If set to 1,

the device is remote such as GRiDLink or PhoneLink.

3 -- mass storage. If set to 0, indicates that the device is not a mass storage device. If set to 1, indicates a mass storage device such as hard disk.

4 -- server. If set to 0, indicates that the device is not a network server. If set to 1, indicates that device is a network server such as GRiDLink or PhoneLink.

5 -- reserved for system use: always set to 0.

6 -- reserved for system use: always set to 0.

7 -- search. If set to 0 and if the mass storage bit (bit 3) is set to 1, indicates a searchable device. GRiD-OS may search this device for an appropriate application program, such as GRiDWrite~Run Text~ to use with a file of Kind ~Text~. If set to 1, the device will never be searched.

OsAllocate

```
FUNCTION OsAllocate (length : Word;
                     VAR error : Word) : Pointer;
```

Purpose and Operation

This call assigns or allocates a memory block of a specified number of bytes to the calling program. The memory block can be of any size from one byte to 64k bytes. If more than 64k bytes are needed, additional OsAllocate calls must be issued. The allocated block will be the lowest addressed segment that satisfies the request.

NOTE: Since there is no inherent memory protection, the program must ensure that it does not alter any memory outside of the allocated block. When your program exits, any memory allocated to the program is freed by the system.

Parameters

length -- a Word specifying the number of bytes of memory to be allocated. A length of zero implies a request of 64k bytes.

Function Return

block -- a pointer to the first byte of the allocated block of memory. NOTE: If you call this function from Pascal, you must define Pointer to be a pointer to whatever kind of variable you are trying to allocate space for. For example: Type Pointer = ^Array[1..2000] OF Word;

Possible Errors

Out of memory (error 2).

OsAttach

```
FUNCTION OsAttach (VAR pathName : Bytes;
                       fileMode : Byte;
                   VAR reserved : Bytes;
                       accessMode : Byte;
                   VAR error : Word) : Word;
```

Purpose and Operation

This call establishes a connection to the file specified by pathName and returns a connection number which other calls use to refer to the file.

OsAttach will establish a connection to an existing file and can also create a new file and connect to that new file.

A connection is always dedicated to a specific type of access: read only, write only, update (read and write), partial directory read, or complete directory read. If a connection is for a write or update access, there can be only one active connection to the file. There can be multiple active connections to the file, however, if all of the connections are for read only access.

The maximum number of active connections to all files in the system is limited only by available memory.

If the access mode is for a partial directory read or complete directory read, the system not only attaches the file, it also opens the file in preparation for subsequent read or seek operations.

NOTE: See Chapter 3 for a description of directory entry formats.

Parameters

pathName -- device-subject-title-kind-password of the file to be attached. This parameter should be formatted as a shortString.

fileMode -- specifies whether the connection is to an existing file or new file as follows:

> 1 = old file. The file must already exist. If the file does not exist, a file-not-found error is returned. (The two directory-type accesses work only with old files.)
>
> 2 = update file. If the file already exists, that file is attached. If the file does not exist, it is created and then attached.
>
> 3 = new file. If the file does not exist, it is created and attached. If the file already exists, the contents of the existing

version are deleted and this new empty file is attached.

reserved -- reserved for future system use. Set to zero using a "dummy" variable. NOTE: See "Special Note" at the end of this description.
accessMode -- a byte defining the type of access that will be permitted for this attachment to the file:

1 = read only access.
2 = write only access.
3 = update (read and write) access.
4 = reserved for system - do not use.
5 = partial directory read access.
6 = complete directory read access.

Function Return

conn -- connection number (data type Word) that can be used in subsequent file-related calls to refer to this file.

Possible Errors

Out of memory (error 2).
Password protected (error 27).
File does not exist (error 33).
File cannot be shared (error 40).
Device full (error 41).
Bad parameter (error 225).
Device not active (error 227).
Any disk errors (errors 101 - 108).

Special Note

The "reserved" parameter was formerly (version 3.0.0 and earlier) used to specify the file password. In version 3.1.0 and later, the password must be specified as part of the pathName parameter. Programs using the old format of this call must put the password in the pathname using the OsChangeExtension call. Programs written prior to version 3.1.0 do not have to be modified if they do not explicitly manipulate passwords.

OsCallDriver

```
PROCEDURE OsCallDriver (VAR pathName : Bytes;
                            level : Byte;
                            request : Word;
                        VAR paramList : ParamListType;
                        VAR error : Word;
```

Purpose and Operation

This procedure is used from within device driver shells. Application programs would not normally call this procedure. A descripton of how to write device drivers and how to use this call within drivers is beyond the scope of this document. Refer to GRiD documents describing device drivers for details on the use of this call.

Parameters

pathName -- the pathname (formatted as a ShortString) of the device to which a request is being sent (typically, 'serial or 'gpib).
level -- a value of 1 specifies that this is a low-level driver (for a mass storage device such as bubble memory, hard disk, or floppy disk), a value of 0 specifies that it is a file level driver (for devices such as printers, PhoneLink, or serial port).
request -- a word defining the specific activity (such as open, read, write) that the device driver is to perform on the device. Refer to the "Guide to GRiD Devices and Device Drivers" for details.
paramList -- a list specifying device characteristics in the following format:

```
ParamListType = RECORD
                    conn : Word;
                    buffer : Pointer;
                    position : LongInt;
                    length : Word;
                    mode : Byte;
                    numBuf : Byte;
                    intAddr : Byte;
                    overflow : Pointer;
                END;
```

OsChangeExtension

```
PROCEDURE OsChangeExtension (VAR pathName : Bytes;
                                 extNum : Byte;
                             VAR extension : Bytes;
                             VAR error : Word);
```

Purpose and Operation

This call lets you examine or change the filename extension (kind or password) on a pathname string while leaving the rest of the pathname unchanged. This call is just a string function. You pass it a string representing a pathname and it returns a portion of this string or modifies the string (depending on what you request with the extNum parameter). Note that this call has no effect whatsoever on the file system or the actual titles of files.

If the pathname already has an extension, then it will be changed to the new one that you specify. If there is currently no extension, the new one will be appended to the file name.

The maximum length of a file name is 80 characters, including any extensions. Since a new extension can increase the length of the file name, you must make sure the pathname buffer is large enough to hold the new name and also ensure that the maximum length is not exceeded.

Parameters

pathName -- references the file whose extension is to be changed. This parameter should be formatted as a short string.
extNum -- specifies which extension (kind or password) is to be examined or changed as follows:

- 1 = change kind extension
- 2 = reserved for system use
- 3 = change password extension
- 41 hex = query current kind extension
- 42 hex = reserved for system use
- 43 hex = query current password extension
- 81 hex = change kind only if none currently appended
- 82 hex = reserved for system use
- 83 hex = change password only if none currently appended

extension -- the actual extension to be appended or the current extension returned by a query. This parameter should be formatted as a short string.

Possible Errors

Bad parameter (error 225).

OsClose

```
PROCEDURE OsClose (conn : Word;
                   VAR error : Word);
```

Purpose and Operation

This call closes a file that was previously opened. The contents of all buffers assigned to the file are written to the file and all memory allocated to the file released to the system.

The file remains attached and can be re-opened without doing another OsAttach.

Parameters

conn -- connection number (data type Word) that specifies the file that is to be closed.

Possible Errors

File not open (error 205).
Bad connection (error 221).
All disk errors (101 - 108).

OsCreateProcess

```
FUNCTION OsCreateProcess (VAR commandLine : Bytes;
                              priority : Byte;
                              uses8087 : Boolean;
                          VAR error : Word) : Word;
```

Purpose and Operation

This call creates a new process with the parameters specified. The process being created is loaded into memory from mass storage -- bubble, disk, etc. (as contrasted with a forked process, which must already be in memory) and is created in the ready state. Thus, if the created process happens to be the highest priority ready process, it would begin executing immediately.

NOTE: After a process has been created, it can only be terminated by issuing an OsExit call -- itself. It can not be deleted by another process.

Parameters

commandLine -- the buffer containing the command to run and any parameters required for that command. This parameter is NOT formatted as a shortString. The buffer contents should be just as though you entered a command via a command line. For example, "GRiDWrite pathName". You must end the command line with a carriage return. NOTE: the file that is to be created as a process is expected to have a Kind of "Run" or "Run fileKind" (for example, "Run Database"). If no kind is specified in the commandLine, the system supplies a kind of "Run". If no device or subject is specified, the system will first look in the currently prefixed subject and then in the Programs subject of the currently prefixed device.

priority -- a value in the range of 0 to 255 indicating the priority of this process. The highest priority is 0, the lowest is 255.

uses8087 -- if this Boolean value is true, it indicates that the 8087 numerical data processor is used by this process. This informs GRiD-OS that the contents of the 8087 registers must be saved whenever the process leaves the run state.

Function Return

pid -- process identification number (data type Word) that can be used by other system calls to refer to this process.

Possible Errors

Out of Memory (error 2) if insufficient memory is available to load this process.
All file system errors.
All disk errors (101 - 108).
All GPIB errors.

OsCreateSemaphore

```
FUNCTION OsCreateSemaphore (VAR error : Word) : Word;
```

Purpose and Operation

This call creates a semaphore for use by system processes. This function returns a word that is the semaphore number or semaphore ID. Processes use this ID number (sid) to refer to the semaphore when issuing OsWait and OsSignal calls.

When a semaphore is created, it is initially set to the busy state. If you want the semaphore to initially be in the not busy state (for example, if you're using it for mutual exclusion), you must issue an OsSignal to the semaphore.

Function Return

sid -- the semaphore ID assigned to the semaphore created.

Possible Errors

Out of Memory (error 2) if there is insufficient memory available for storage of the semaphore.

OsDecodeException

```
PROCEDURE OsDecodeException (code : Word;
                             VAR execption : Bytes);
```

Purpose and Operation

This call converts a numerical error code generated by the system to a more meaningful string of up to 80 ASCII characters. (NOTE: the text comprising the character string associated with each error code is in the file named @SystemErrors~text~. This file must be under the programs directory of the currently prefixed device.). The resultant string can provide a more useful error message to users and operators of the Compass.

Parameters

code -- the system error number that is to be decoded. See the @SystemErrors~Text~ file for a numerical listing of the error numbers and the message strings that will be returned.

exception -- a buffer (which should be formatted as a shortstring) where the error message will be returned.

OsDelay

```
PROCEDURE OsDelay (timeLimit : Word);
```

Purpose and Operation

This call suspends execution of the current process by placing it in the wait state for a specified time limit. The process will remain in the wait state until the specified time limit has expired. It will then proceed to the ready state where it assumes its prioritized position among the other ready processes.

If you specify a time limit of zero, the process will leave the run state, go to the wait state, and then proceed immediately to the ready state. Thus, processes of equal priority could use this mechanism to ensure that they all get their turn as the current process.

Note that a process can only delay itself (there is no process ID parameter to let you specify another process).

Parameters

timeLimit -- a word specifying the number of milliseconds (rounded up to a multiple of 10 milliseconds) to suspend the current process. Thus, you can specify delays ranging from zero to 65,540 milliseconds.

OsDelete

```
PROCEDURE OsDelete (conn : Word;
                    VAR error : Word);
```

Purpose and Operation

This call deletes the specified file from the file system. The file must currently be attached for either a write access or update access and it must also be open.

An OsDetach is performed automatically after the file is deleted since the connection is meaningless after the file is deleted.

Parameters

conn -- connection number (data type Word) that specifies the file that is to be deleted.

Possible Errors

File does not exist (error 33).
All disk errors (101 - 108).

OsDeleteProcess

```
PROCEDURE OsDeleteProcess (pid : Word;
                           VAR error : Word);
```

Purpose and Operation

This call deletes the specified forked process from the system. A process can be deleted regardless of which state it is in. (NOTE: OsDeleteProcess cannot be used to terminate a process that was created with OsCreate: OsExit call must be used for that purpose.)

Parameters

pid -- process ID, a word identifying the forked process to be deleted.

Possible Errors

Process does not exist (error 251).

OsDeleteSemaphore

```
PROCEDURE OsDeleteSemaphore (sid : Word;
                             VAR error : Word);
```

This call deletes the specified semaphore from the system. Any process that issues an OsSignal or OsWait to this semaphore will receive a Semaphore does not exist error. Any processes that are actually waiting at this semaphore when it is deleted, will proceed to the ready state and have a Semaphore does not exist (error 252) error returned.

Parameters

sid -- semaphore identification number that was returned by GRiD-OS when the semaphore was created.

Possible Errors

Semaphore does not exist (error 252).

OsDetach

```
PROCEDURE OsDetach (conn : Word;
                    VAR error : Word);
```

Purpose and Operation

This call severs a file connection that was established previously by OsAttach. All system resources being utilized for the connection are released and the relationship between this connection and a pathname is severed.

If the file has not been closed, an OsClose will automatically be performed by the system.

Parameters

conn -- connection number (data type Word) that specifies the file connection that is to be severed. CAUTION: Passing an uninitialized value to conn can result in the OS trying to access memory mapped I/O apce -- this could hangthe system.

Possible Errors

Bad connection (error 221) if the specified connection number does not exist.

All disk errors (101 - 108)

OsExit

```
PROCEDURE OsExit (code : Word);
```

Purpose and Operation

This call is used to exit a program by causing the current process to delete itself. When a process exits or is deleted, all of its resources, such as memory or active file and device connections are returned to the system.

Any processes waiting for a message from this process will receive the contents of the code parameter (as the "note" parameter of OsReceive) and will also get a "Process does not exist" error. Any semaphores created by this process are also deleted. Therefore, any processes that subsequently wait on these semaphores will get a "Semaphore does not exist" error.

Parameters

code -- the contents of this word are put into the "note" parameter (see OsReceive) of any process waiting for a message from the exiting process.

OsFlushAllBuffers

```
PROCEDURE OsFlushAllBuffers (conn : Word;
                             VAR error : Word);
```

Purpose and Operation

This call writes the contents of all buffers in memory (allocated with OsOpen) currently assigned to a file out to the file residing on a device-subject. It can thus be thought of as a precautionary call that lets you save the contents of file buffers without going through an OsClose-OsOpen sequence.

Parameters

conn -- connection number (data type Word) that specifies the file whose buffer(s) is to be saved.

Possible Errors

File not open (error 205).
Bad connection (error 221).
All disk errors (101 -108).

OsForkProcess

```
FUNCTION OsForkProcess (VAR entryPoint : Bytes;
                            priority : Byte;
                            uses8087 : Boolean;
                            stackSize : Word;
                        VAR error : Word) : Word;
```

Purpose and Operation

This call forks a new process with the parameters specified. "Forking" a process is similar to creating a process with the following exceptions and limitations:

- The code for the process being forked must already be present in memory.
- The process being forked must be a parameterless PUBLIC procedure, and must be a 'far' or LARGE procedure as opposed to a 'near' or SMALL procedure. (See the compiler controls section of the appropriate language manual for a discussion of LARGE and SMALL.)
- The forked process cannot be terminated using an OsExit call: it must be terminated using OsDeleteProcess.

When a process is first forked, it will be in the ready state. Thus, if it happens to be the highest priority ready process, it could begin executing immediately.

Parameters

entryPoint -- the address of a LARGE, parameterless, procedure. (In Pascal, you can just specify the name of the procedure.)

priority -- a value in the range of 0 to 255 indicating the priority of this process. The highest priority is 0, the lowest is 255.

uses8087 -- if this Boolean value is true, it indicates that the 8087 numerical data processor is used by this process. This informs GRiD-OS that the contents of the 8087 registers must be saved whenever this process leaves the run state.

stackSize -- specifies the number of words to be reserved as stack for the process (typically in the range of 500-1000 words). Note: an insufficient stack size will cause seemingly random failures.

Function Return

pid -- process identification number (data type Word) that can be used by other system calls to refer to this process.

Possible Errors

Out of Memory (error 2) if insufficient memory is available to fork this process.

OsFree

```
PROCEDURE OsFree (block : Pointer;
                  VAR error : Word);
```

Purpose and Operation

This call frees or deallocates a block of memory that was previously allocated to the calling process.

Parameters

block -- points to the first byte of the block of memory to be freed. This should be the same as the pointer returned from OsAllocate when the block was allocated. See the "block" parameter for the OsAllocate function for a discussion.

Possible Errors

Invalid memory block (error 11).

OsGetArgument

```
FUNCTION OsGetArgument (short : Boolean;
                        VAR argument : Bytes) : Char;
```

Purpose and Operation

This call returns arguments from the command line in the form of a string of characters up to 255 characters in length. Each argument must be separated by a delimiting character (described below) and each call to OsGetArgument returns one argument. Therefore, you would use this call repeatedly until you have obtained all of the arguments contained in the command line.

The argument record returned by this call is a short string that may be up to 255 bytes in length excluding the length byte. Since you cannot know the length of the argument until it is returned, you must ensure that the buffer you provide can accommodate the maximum length of the argument.

Delimiter Characters

In addition to returning an argument, this function also returns the character that was used as the delimiter to mark the end of each argument. The ASCII codes recognized as delimiters are as follows:

character	ASCII hex value
space	20
!	21
#	23
$	24
%	25
&	26
'	27
(	28
)	29
+	2B
-	2D
:	3A
<	3C
=	3D
>	3E
[	5B
\	5C
]	5D
¦	7C
DEL	7F

Additionally, any byte with a value from 00 to 20 hex or with a value greater than 80 hex will be recognized as a delimiter character and returned by OsGetArgument.

NOTE: These delimeter characters can be used within an argument (for example,

within a file name) but they must be enclosed with single quotation marks (').

Parameters

short -- a Boolean indicating whether the argument to be returned is short (true) or long (false). A short argument can be up to 80 characters in length and all alphabetic characters will be shifted to upper case. A long argument can be up to 255 characters in length and no shifting of alphabetic characters is performed.
argument -- the buffer where the returned argument (formatted as a short string) is to be placed.

Function Return

delim -- the character used as the trailing delimiter for the argument.

Possible Errors

None.

Examples

The following example illustrates the short string records and delimiters returned by successive calls to OsGetArgument from the following argument.

```
CAT  'List Directory' <RETURN>
```

	length	argument	delimter
1st call	3	CAT	20 hex (space)
2nd call	14	List Directory	0D hex (RETURN)

Notice that a string enclosed in single quotation marks (') is considered a literal and characters within such a string that would normally be considered to be delimeters are simply returned as part of the argument. Thus, the space separating "List" and "Directory" is not treated as a delimiter. Note also that the enclosing quotes are not returned as part of the argument.

Another example of the OsGetArgument is when a file of Kind "Text" is selected from the File form. The system invokes GRiDWRite and when that application program begins executing, it obtains the pathname of the selected file by calling OsGetArgument to parse the command line passed to it by the Executive program.

OsGetMemStatus

```
PROCEDURE OsGetMemStatus (pid : Word;
                      VAR memStatus : MemStatusType;
                      VAR error :Word);
```

Purpose and Operation

This call returns information concerning the amount of memory that has been allocated and how much is still available. Most of the information returned is a summary of system-wide information but, if you supply a process ID (pid) with the call, you will be given some specific information about memory allocation for that process. The organization of the memory status record returned by this call is as follows:

```
    MemStatusType = RECORD
                        freeBytes : Longint;
                        freeBlocks : Word;
                        largestFree : Word;
                        allocBytes : Longint;
                        allocBlocks : Word;
                        largestAlloc : Word;
                    END;
```

Parameters

pid -- process ID. A word identifying the process whose memory status information is to be returned. If this word is null (OFFFF hex), then the memory status information returned will be a summary of memory usage by all processes.
memStatus -- the location where the memory status record should be returned.

MemStatusType Record Fields

freeBytes -- the number of unallocated bytes remaining in the system.
freeBlocks -- the total number of unallocated blocks, regardless of size, remaining in the system.
largestFree -- the size, in bytes, of the largest unallocated block remaining in the system.
allocBytes -- the number of bytes allocated to the calling process or, if a null (0ffff) pid is specified, the total number of bytes allocated to all processes in the system.
allocBlocks -- the number of blocks, regardless of size, allocated to the calling process or, if a null (0ffff) pid is specified, the total number of blocks allocated to all processes in the system.
largestAlloc -- the size, in bytes, of the largest block of memory allocated to the calling process or, if no pid is specified, the largest block of memory allocated to any process in the system.

Possible Errors

Process does not exist (error 251).

OsGetPrefix

```
FUNCTION  OsGetPrefix : ShortStringPtr;
```

Purpose and Operation

This routine returns a pointer to a short string containing the current device-subject prefix and can be used to read the current prefix. Note: the pointer that is returned points to a string that is in the system data space. This string should not be updated. You can use a Common Code call, if it is necessary to change the prefix.

OsGetProperty

```
PROCEDURE  OsGetProperty (tag : Word;
                          VAR length : Word;
                          VAR buffer : Bytes;
                          VAR error : Word);
```

Purpose and Operation

This routine lets you examine some of the system-wide properties that apply to a specific Compass computer. These properties are normally examined and set using GRiDManager and the settings of the properties are recorded in the file User~Profile~ under the Programs subject.

See OsPutProperty for additional information on the User~Profile~ file.

Parameters

tag -- specifies which system property is to be examined as follows:

Value	Property
1	time offset
2	screen frame on/off
5	system-wide font
9	current printer
10	current plotter
11	start-up file

length -- the length, in bytes, of the buffer parameter to be associated with the designated tag.

buffer -- a sequence of bytes defining the characteristics of the designated tag as follows:

Tag	Data
1	timeOffset record (described below)
2	data = 1, frame is on; 2 = frame is off
5	system-wide font name
9	current printer name
10	current plotter name
11	start-up (boot) file name

The data associated with font, printer, plotter, and start-up file is the name of the device/file as it would appear in the Options form of GRiDManager: a complete pathname is not required.

error -- if the tag specified does not exist, an error 225 (Bad parameter) is returned.

TimeOffsetType Record Fields

```
TimeOffsetType = RECORD
                    year : Word;
                    dayOffset : Word;
                    hour : Byte;
                    minute : Byte;
                    second : Byte;
                    dayOfWeek : Byte;
```

```
END;
```

Each of the fields in this record specify an offset from the time as maintained by the built-in clock in the Compass computer. The built-in clock maintains Greenwich Mean Time (GMT). The offset values in this record provide the information needed to "localize" the time displayed by applications to the time where the Compass is currently located.

OsGetSize

```
FUNCTION OsGetSize (block : Pointer;
                    VAR error : Word) : Word;
```

Purpose and Operation

This call returns the size of a memory block that was previously allocated to the calling process.

Parameters

block -- points to the first byte of the block whose size is to be returned. This should be the same as the pointer returned from OsAllocate when the block was allocated.

Function Return

length -- a word specifying the length, in bytes, of the specified memory block allocated to this process. A length of zero indicates 64k bytes.

Possible Errors

Invalid memory block (error 11).

OsGetStatus

```
PROCEDURE OsGetStatus (conn : Word;
                       VAR status : Bytes;
                           length : Word;
                       VAR error : Word);
```

Purpose and Operation

This call is a rather special purpose call that would normally be used only by system level maintenance or trouble-shooting programs. It lets you examine the status information associated with each file or device. This call returns status information about a file that is currently attached. The status information includes such things as whether the file is open, what type of access it is open for, permissable seek directions, current file position, current size of the file, and total space allocated for the file. The organization of the status record returned for files and mass storage devices by this call is as follows:

```
StatusType = RECORD
                 open : Boolean;
                 access : Byte;
                 seek : Byte;
                 filePosition : Longint;
                 fileLength : Longint;
                 numPages : Word;
                 numPagesAllocated : Word;
             END;
```

Note: The status record returned for non-mass storage devices (such as printers) is uniquely defined and different for each device.

Parameters

conn -- connection number (data type Word) that specifies the file whose status is to be examined.
status -- the location where the status information is to be returned.
length -- the number of status bytes to get from the StatusType record.

StatusType Record Fields (for files or mass storage devices)

open -- a Boolean that, if true, indicates that the file is currently open. If False, then no other values in the record are valid.
access -- a byte indicating the access rights for this file (specified at OsAttach time). If the appropriate bit listed below is on (1), then the indicated access is allowed:

bit #	access
0	delete access (true if either write or update true)
1	read access
2	write access
3	update access

seek -- a byte indicating the types of seeking which can be performed on this connection. The types of seeks permitted are device dependent. If the appropriate bit listed below is on (1), then the indicated access is allowed:

bit #	access
0	seek forward
1	seek backward

filePosition -- a long integer indicating the byte number that is the current file position location.
fileLength -- a long integer indicating the total number of bytes in the file.
numPages -- a word indicating the total number of pages this connection currently occupies on the device. Page size (sector size) is 256 bytes on bubble memory and 512 bytes for all other mass storage devices.
numPagesAllocated -- a word indicating the total number of pages (sectors) allocated for this connection on the device.

Possible Errors

Bad connection (error 221).

OsGetSystemID

```
PROCEDURE OsGetSystemID (VAR systemID : Bytes);
```

Purpose and Operation

This call returns the identification string for the system in the form "Version #.#.# of GRiD-OS". The identifier is in the format of a ShortString.

Parameters

systemID -- the location to store the returned system ID record. This should be formatted as a short string.

OsGetTime

```
PROCEDURE OsGetTime (mode : Byte;
                     VAR time : TimeType);
```

Purpose and Operation

This call returns all available information from the Compass' real time clock. The organization of the time record returned is as follows:

```
    TimeType = RECORD
                   year : Word;
                   month : Byte;
                   day : Byte;
                   hour : Byte;
                   minute : Byte;
                   second : Byte;
                   tenthOfSec : Byte;
                   dayOfWeek : Byte;
                   dayOfYear : Word;
               END;
```

Parameters

mode -- if this byte equals 1, then time is based on Greenwich Mean Time (GMT). If this byte equals 2, then all times are based on the local, or Compass-relative time.
time -- the location where the time imformation is to be returned.

TimeType Record Fields

year -- a Word specifying the current year.
month -- a Byte specifying the number of the current month (1-12).
day -- a Byte specifying the current day (1-31).
hour -- a Byte specifying the current hour (0-23).
minute -- a Byte specifying the current minute (0-59).
second -- a Byte specifying the current second (0-59).
tenthOfSec -- a Byte specifying the current 1/10 second (0-9).
dayOfWeek -- a Byte specifying the current day of week (Sunday = 1, Saturday = 7).
dayOfYear -- a Word specifying the current day of year (1-366).

OsGetWork

```
FUNCTION  OsGetWork : ShortStringPtr;
```

Purpose and Operation

This routine returns a pointer to the short string containing the current device designated as the 'work' device used by compilers and the link program. Note: the pointer that is returned points to a string in the system data space. This string should not be updated.

OsLookUpName

```
FUNCTION OsLookupName (VAR name : Bytes;
                       VAR error : Word) : LongInt;
```

Purpose and Operation

This call looks up a specified name and returns the token that was stored with it by an OsRegisterName call.

Parameters

name -- the location of the name to look up. The name should be formatted as a ShortString up to 255 characters in length.

Function Return

token -- the LongInt that accompanied the specified name.

Possible Errors

File (Name) does not exist (error 33).

OsMatchWildcard

```
PROCEDURE OsMatchWildCard (VAR testStr : Bytes;
                               strLen : Word;
                           VAR matchStr : Bytes;
                               matchLen : Word;
                               idepOfCase : Boolean;
                               fullMatch : Boolean;
                           VAR length : Word);
```

Purpose and Operation

This routine compares a specified string (testStr) to a wildcard string (matchStr) and is typically used for comparing pathnames. The wildcard string can contain the wild card character (0F7 hex) which will match with any character or string of characters. For example, if the matchStr is "G...n" (where "..." represents the wildcard character 0F7 hex), this string would match fully with GRiDPlan and Govern, and would match through the first six characters with the string "Goldenrod".

You can specify that upper case and lower case be ignored and whether the two strings must match completely. Upon completion, the variable length indicates how much of the two strings are the same. If length is zero, then there was no match.

Parameters

testStr -- the sequence of bytes that are to be compared against the wildcard string.
strLen -- the length of the string that is to be compared against the wildcard string.
matchStr -- the wildcard byte string against which the comparison is to be made. This string can contain the wildcard character (7F hex) that will match with any character(s) in the target string.
matchLen -- the length of the wildcard string.
idepOfCase -- ignore case. If true, the comparison is made without regard to upper or lower case. If false, the case of characters in the strings must match exactly.
fullMatch -- If true, the two strings must match in their entirety. If they do not, a length of zero is returned. If false, the length will indicate how many bytes of the two strings matched.
length -- indicates how many bytes of the two strings matched. The call is terminated as soon as non-matching bytes are encountered.

OsOpen

```
PROCEDURE OsOpen (conn : Word;
                  numBuf : Byte;
              VAR error : Word);
```

Purpose and Operation

This call opens a file by allocating memory for the file buffers and file pointers that will be used during subsequent accesses. The file must have previously been attached using OsAttach.

Each opening of a file requires the allocation of at least one buffer in memory. You can specify that more than one buffer be allocated to increase performance. -- you are limited only be the amount of available memory. However, one buffer should usually be sufficient and is the recommended number because of the memory utilization penalty incurred by multiple buffers. NOTE: the buffer for hard disks and floppy disks is 512 bytes, and for bubble memory is 256 bytes.

When a file is first opened, the current file position marker is set to zero. See "Operating on Files" in Chapter 3 for a discussion of the current file position marker.

Parameters

conn -- connection number (data type Word) that specifies the file that is to be opened.
numBuf -- the number of buffers to use for this file.

Possible Errors

Out of memory (error 2).
Bad connection (error 221).
File already open (error 222).
All disk errors (101 - 108).

OsOverlay

```
PROCEDURE OsOverlay (VAR name : Bytes;
                         pid : Word;
                     VAR error : Word);
```

Purpose and Operation

This call loads a specified overlay program into memory. Only one level of overlays is allowed: a program that has been brought into memory as an overlay cannot then issue an OsOverlay call. This routine can be called only from the root (non-overlaid) phase.

IMPORTANT: When an overlay module is loaded into memory, the previous overlay's code and data segments are overwritten. Therefore, you cannot have any static variables in the data segment of an overlay: they must be in the root module. For a thorough discussion of overlays, see the GRiD Program Development Guide.

Parameters

name -- a record, formatted as a ShortString, containing the name of the overlay. The overlay name is defined using the linker overlay control. Refer to the Program Development Guide for details.

pid -- the process ID of the overlay. Usually, this will be the same as the pid returned by OsWhoAmI; that is, the overlay is part of the same process that is issuing the OsOverLay call.

Possible Errors

File not found (error 33).
All disk errors (101 - 108).
All loader errors (300 - 304).

OsPutProperty

```
PROCEDURE  OsPutProperty (tag : Word;
                          length : Word;
                      VAR buffer : Bytes;
                      VAR error : Word);
```

Purpose and Operation

This routine lets you alter some of the system-wide properties that apply to a specific Compass computer. These properties are usually set using GRiDManager and the current settings are recorded in the file User^Profile^ under the Programs subject.

WARNING: Tab values 1 thorugh 1000 (decimal) are reserved for use by GRiD. Never specify a tag value in the range 1 - 1000 other than those listed below. Other tag values in this range are associated with system internal information and altering the data associated with these other tags can have unpredictable results. You can, however, use tags beyond this range to record user-specific information in the file User^Profile^.

Parameters

tag -- specifies which system property is to be altered as follows

Value	Property
1	time offset
2	screen frame on/off
5	system-wide font
9	current printer
10	current plotter
11	start-up file

length -- the length, in bytes, of the buffer parameter to be associated with the designated tag.

buffer -- a sequence of bytes defining the characteristics of the designated tag as follows:

Tag	Data
1	timeOffset record (described below)
2	data = 1, turn frame on; 2 = turn frame off
5	system-wide font name
9	current printer name
10	current plotter name
11	start-up (boot) file name

The data associated with font, printer, plotter, and start-up file is the name of the device/file as it would appear in the Options form of GRiDManager: a complete pathname is not required.

error -- if the tag specified does not exist, an error 225 (Bad parameter) is returned.

TimeOffsetType Record Fields

```
TimeOffsetType = RECORD
                     year : Word;
```

```
    dayOffset : Word;
    hour : Byte;
    minute : Byte;
    second : Byte;
    dayOfWeek : Byte;
  END;
```

Each of the fields in this record specify an offset from the time as maintained by the built-in clock in the Compass computer. The built-in clock maintains Greenwich Mean Time (GMT). The offset values in this record provide the information needed to "localize" the time displayed by applications to the time where the Compass is currently located.

OsRead

```
FUNCTION OsRead (conn : Word;
                 VAR buffer : Bytes;
                     length : Word;
                 VAR error : Word) : Word;
```

Purpose and Operation

This call reads a specified number of bytes from a file and places them in a specified buffer. The read operation begins at the current file position. If the end of the file is reached before the specified number of bytes are read, the read is terminated and the current file position is left at one byte beyond the end of the file. This function returns a word specifying the number of bytes actually read from the file. The number of bytes read can be less than the number specified by length only if the end of file is reached or if an error occurs.

If the file was attached in the partial directory mode (attach access type = 5) or complete directory mode (attach access type = 6), this call treats directory entries, rather than bytes, as the objects that are read. The read operation begins at the current directory entry. If the end of the directory is reached before the specified number of entries are read, the read is terminated and the current file position is left at one entry beyond the end of the directory. In directory mode, this function returns a word specifying the number of entries actually read from the file. The number of entries read will be less than the number specified by length only if the end of the directory is reached or if an error occurs.

Parameters

conn -- connection number (data type Word) that specifies the file to be read.
buffer -- references the buffer where the data read from the file is to be placed. Note: it is the programmer's responsibility to provide a buffer large enough to accommodate the data that is read. The operating system does not check the size of the buffer.
length -- the number of bytes (or directory entries) to be read from the file. The maximum length is 65,535 bytes.

Function Return

amountRead -- a word specifying the number of bytes (or entries) actually read from the file.

Possible Errors

File access denied (error 38).
File not open (error 205).
Bad connection (error 221).
All disk errors (101 - 108).

OsReceive

```
FUNCTION OsReceive (sourcePid : Word;
                    class : Word
                    timeLimit : Word;
                VAR note : Word;
                VAR error : Word) : Pointer;
```

Purpose and Operation

This call places the current process in the wait state where it remains until it receives a message sent by another process, or until a specified time limit has expired, or until it receives an appropriate error message from the system.

If an appropriate message is already available when the process issues an OsReceive, the process immediately proceeds to the ready state. If you specify that the message must be sent by a particular process, and if that process does not exist, GRiD-OS will give the waiting process a Process Does Not Exist error and move the process to the ready state.

If a message of the specified class and from the specified sending process is not available when this process enters the wait state, the process will remain there. The process will stay in the wait state until an appropriate message is received or until the specified time limit expires. You can specify a time limit with a null (0ffffH) value. In this case, the process will wait forever to receive the appropriate message. (NOTE: if the specified sending process is deleted, the waiting process would be given an appropriate error indication and moved to the ready state.)

The sending process does not make a separate copy of the message for the receiving process: there is but a single instance of the message. Therefore, when the receiving process gets back to the run state, it should immediately make its own copy of the message and inform the sending process that it is finished with the message. The receiving process could accomplish this by passing a note back to the sending process.

Parameters

sourcePid -- the process from which the message is to be received. If null (0ffffH), then a message sent by any other process can be received.

class-- the class of message that can be received. If this is null(0ffffH), a message of any class can be received.

timeLimit -- the amount of time, in milliseconds (rounded up to a multiple of 10 milliseconds), that the process will wait for an appropriate message. If the time limit expires before a message is received, the process goes to the ready state and a Time Out error is returned. If you specify a null (0ffffH) timeLimit, the process will wait forever for a message. If you specify a timeLimit of zero, the process will proceed immediately to the

ready state.

note -- the 2-byte note (data type Word) that can be passed by value from the sending process. Interpretation of the note contents is application dependent.

Function Return

This procedure returns a pointer to the buffer holding the actual message sent. If you are issuing this call in Pascal, you must provide an appropriate data type to obtain the returned pointer.

Possible Errors

Process does not exist (error 251) if the specified message-sourcing process does not currently exist in the system.

Timeout (error 253) if a message is not received before the specified time limit expires.

OsRegisterName

```
PROCEDURE OsRegisterName (VAR name : Bytes;
                              token : LongInt;
                              mode : Byte;
                          VAR error : Word);
```

Purpose and Operation

This call records or registers a ShortString containing a name that other processes or programs can examine or look up (using OsLookupName). A token (data type LongInt) is stored along with the name and this token can thus be accessed by any process or program that knows the appropriate name.

This same call can delete or unregister a name so that it is no longer available to other processes or programs in the system.

The OsRegisterName and OsLookupName calls provide a very simple mechanism for exchanging information between processes. This capability is most often used to establish initial contact between processes before they know the process IDs required to use the message passing or semaphore calls to communicate with other processes.

Parameters

name -- the location of the name to be registered or unregistered. The format of the actual name at this location is a short string up to 255 characters in length.
token -- a LongInt that is stored along with the name. Interpretation of the token is entirely up to the user.
mode -- if the value of this byte equals 1, it means that the indicated name is to be registered. If the value of this byte equals 2, it means that the indicated name is to be unregistered or deleted.

Possible Errors

Out of memory (error 2).
File (name) already exists (error 32).

OsRemoveDevice

```
PROCEDURE OsRemoveDevice (VAR name : Bytes;
                          VAR error : Word);
```

Purpose and Operation

This call removes the specified device from the system's Active Device Table. Thus, this call is the equivalent of "deactivating" a device from the command line (see the Program Development Guide for a description of the Deactivate program). For a detailed discussion of how to use this call, refer to GRiD documentation on device drivers.

Parameters

name -- the device name (formatted as a ShortString) assigned during the OsAddDevice call to the device driver.

OsRename

```
PROCEDURE OsRename (conn : Word;
                    VAR newName : Bytes;
                    VAR error : Word);
```

Purpose and Operation

This call changes the name of an existing, attached file. The file must have been attached with a write access or update access specified and the file must also be open.

Parameters

conn -- connection number (data type Word) that specifies the file that is to be renamed.

newName -- the new file name to be given to this file. Note that the device-subject part of the pathname remain unchanged. It is only the fileName (or title) portion that is altered. If you supply a full pathname with this parameter, the device-subject are ignored. If you do not specify a kind, it is given a kind of Untyped.

Possible Errors

File already exists (error 32).
All disk errors (101 - 108).

OsSeek

```
PROCEDURE OsSeek (conn : Word;
                  mode : Byte;
                  length : Longint;
              VAR error : Word);
```

Purpose and Operation

This call alters the current file position by moving the marker a specified number of bytes. The first byte of a file is byte zero. You can move the marker forward or backward in the file, move it to a specific byte location in the file, or move to a position a specific number of bytes in from the end of the file.

A seek does not actually access a file on a device -- it simply changes the current file position.

If a seek is made beyond the end of the file, the current file position is changed but the file is not actually extended until a subsequent write is performed at that position.

Parameters

conn -- connection number (data type Word) that specifies the file on which the seek is to be performed.
mode -- a byte specifying the type of seek to perform as follows:
- 1 = move marker back by length bytes
- 2 = set marker at byte specified by length
- 3 = move marker forward by length bytes
- 4 = move marker to end of file minus length bytes

length -- the number of bytes to seek or the location that the marker should be positioned to.

Possible Errors

File not open (error 205).
Bad connection (error 221).
Bad parameter (error 225).

OsSend

```
PROCEDURE OsSend (destPid : Word;
                  class : Word
                  note : Word;
              VAR message : Bytes;
              VAR error : Word);
```

Purpose and Operation

This call sends a message to another process. The OsSend call does not make a separate copy of the message. Therefore, you must ensure that you do not alter the message until after the intended receiving process is done with the message. There is no automatic mechanism for verifying reception of a message. You can accomplish this verification yourself by, for example, having the receiving process send a note back to the originator when it has finished with the message.

Parameters

destPid -- the process that is to receive the message.

class -- the user specified class that will be associated with this message and examined to determine if it can be delivered to a receiving process. If you specify a null (0ffffH) class, the message can only be received by a process that has specified a null class as part of its OsReceive call.

note -- the 2-byte note (data type Word) that can be passed by value to the receiving process. Interpretation of the note contents is application dependent.

message -- the buffer containing the actual message. IMPORTANT: GRiD-OS requires that the first 16 bytes of the message contain all zeros. The length and format of the rest of the message is application dependent.

Possible Errors

Process does not exist (error 251) if the process that the message is addressed to does not exist in the system.

Out of memory (error 2) if there is insufficient memory to send the message.

OsSetPriority

```
PROCEDURE OsSetPriority (pid : Word;
                         priority : Byte;
                     VAR error : Word);
```

Purpose and Operation

This call assigns a new priority to a specified process. Thus, you can dynamically change process priorities from the initial values assigned when each process is created. A process can change the priority of any other process, and can also change its own priority.

Parameters

pid -- process identification number. A word identifying the process whose priority is to be changed.

priority -- the new priority, in the range of 0 to 255, for the specified process. Zero is the highest priority, 255 the lowest.

Possible Errors

Process does not exist (error 251).

OsSetStatus

```
PROCEDURE OsSetStatus (conn : Word;
                       VAR status : Bytes;
                           length : Word;
                       VAR error : Word);
```

Purpose and Operation

This call sets up status information about a file that is currently attached. It will typically be used only on special devices, such as the modem or serial port, that require very specific operating parameters. For example, you would use the OsSetStatus procedure to set the baud rate, parity and other operating parameters for the modem. The use of OsSetStatus is device dependent and is described in the documentation for each specific device.

Note that the 'status' parameter used here is not the same StatusType record that is used with the OsGetStatus call. Instead, it simply points to a buffer containing application or device-dependent bytes.

Parameters

conn -- connection number (data type Word) that specifies the file whose status is to be set.
status -- the status information to be sent.
length -- the number of status bytes to send.

Possible Errors

Bad connection (error 221).

OsSignal

```
PROCEDURE OsSignal (sid : Word;
                    mode : Byte;
                    note : Word;
                VAR error : Word);
```

Purpose and Operation

A semaphore is always created in the busy state. This call sets the specified semaphore to the not busy state. If another process is waiting at this semaphore when the OsSignal call is issued, that waiting process proceeds to the ready state. If more than one process is waiting at this semaphore, the process with the highest priority proceeds to the ready state (except for mode 3, explained below, which allows all waiting processes to proceed).

Parameters

sid -- semaphore identification number that was returned by GRiD-OS when the semaphore was created.

mode -- a byte specifiying one of three signalling modes:

mode=1 -- This mode always lets one, and only one, process pass. If no process is currently waiting at the semaphore, the signal is retained (the semaphore is held not busy) until an OsWait is issued to this semaphore. The process issuing the OsWait proceeds to the ready state and sets the semaphore busy. If a process is already waiting at the semaphore, it proceeds to the ready state and the semaphore returns to busy. This mode can be used to ensure that only one process can proceed through a critical section of code at a time.

mode=2 -- This mode lets one process pass if there is currently a process waiting, but the signal is not retained. If a process is currently waiting at the semaphore, it is signalled and proceeds to the ready state. Otherwise, the semaphore remains busy and a process arriving subsequently must wait for another OsSignal. This mode is useful for informing any waiting process that a particular event has occurred.

mode=3 -- This mode lets all currently waiting processes pass. All processes waiting at the semaphore are signalled and proceed to the ready state. This mode is useful for synchronizing the initiation of several processes. The signal is not retained; if no processes are currently waiting at the semaphore, the semaphore remains busy and processes arriving subsequently must wait for another OsSignal.

note -- the 2-byte note (data type Word) that can be passed by value from the signalling process. Interpretation of the note contents is application dependent.

Possible Errors

Semaphore does not exist (error 252) if the specified semaphore (sid) does not exist in the system.

OsSwitchBuffer

```
FUNCTION OsSwitchBuffer (VAR buffer : Bytes) : Word;
```

Purpose and Operation

This call lets you specify an alternate buffer to be used for the OsGetArgument call and thus obtain arguments from places other than the command line. You should not use this call until the command line has been completely processed since there is no way to switch back.

Parameters

buffer -- the new buffer that a subsequent OsGetArgument call should scan for arguments.

Function Return

length -- a word indicating how far scanning had proceeded in the previous buffer; that is, the first byte beyond the last delimiter character encountered on the previous OsGetArgument call.

Possible Errors

None.

OsTruncate

```
PROCEDURE OsTruncate (conn : Word;
                      VAR error : Word);
```

Purpose and Operation

This call deletes the contents of a file from the current file position to the end of the file. Upon completion of the truncation, the current file position is one byte beyond the new end of file.

Parameters

conn -- connection number (data type Word) that specifies the file that is to be truncated.

Possible Errors

File not open (error 205).
Bad connection (error 221).
All disk errors (101 - 108).

OsWait

```
FUNCTION OsWait (sid : Word;
                 timeLimit : Word;
                 VAR error : Word) : Word;
```

Purpose and Operation

This call suspends the current process by placing it in the wait state where it remains until the specified semaphore is not busy or until a specified time limit has expired.

If the semaphore is not busy when the process issues this call, the process immediately proceeds to the ready state and the semaphore is set to busy. The semaphore remains busy until an OsSignal call is directed to it (typically, by the process that most recently proceeded past the semaphore).

If the semaphore is busy when the process issues this call, the process stays in the wait state until the semaphore is signalled (set not busy) or until the specified time limit expires. You can specify a time limit with a null (0ffffH) value. In this case, the process will wait forever for the semaphore to become not busy. (NOTE: if the specified semaphore is deleted, the waiting process would be given an appropriate error indication and moved to the ready state.)

If other processes had previously issued OsWait calls to this semaphore and are still waiting for their turn to proceed, this process is placed in a queue according to its process priority. It cannot proceed until all of the waiting processes of a higher priority have passed the semaphore.

Parameters

sid -- semaphore identification number that was returned by GRiD-OS when the semaphore was created.

timeLimit -- the amount of time, in milliseconds (rounded up to a multiple of 10 milliseconds), that the process will wait for a signal. If the time limit expires before a signal is received, the process goes to the ready state and a Time Out error is returned. If you specify a null (0ffffH) timeLimit, the process will wait forever for a signal. If you specify a timeLimit of zero, the process will proceed immediately to the ready state: if there was no signal for the semaphore, a timeout error will be returned.

Function Return

note -- the 2-byte note (data type Word) that can be passed by value from the signalling process. Interpretation of the note contents is application dependent.

Possible Errors

Timeout (error 253) if a signal is not received before the specified time limit expires.

Semaphore does not exist (error 252) if the specified semaphore (sid) does not exist in the system.

OsWhoAmI

```
FUNCTION OsWhoAmI : Word;
```

Purpose and Operation

This call returns the process identification number (pid) assigned to this process when it was created.

Function Return

pid -- a word that is the process identification number assigned to the requesting process.

OsWrite

```
PROCEDURE OsWrite (conn : Word;
                   VAR buffer : Bytes;
                       length : Word;
                   VAR error : Word);
```

Purpose and Operation

This call writes a specified number of bytes to a file. The write operation begins at the current file position. If the end of the file is reached, the additional data is appended to the file and the end of file marker is moved to a position one byte beyond the last byte written. If the current file position where the write begins is already beyond the end of the file, the file is extended to that point and the writing begins there.

If the current file position is not beyond the end of the file, the new data is written over the previously existing data.

Parameters

conn -- connection number (data type Word) that specifies the file to be written to.
buffer -- a pointer to the buffer containing the data to be written to the file.
length -- the number of bytes to be written to the file.

Possible Errors

File access denied (error 38).
Device full (error 41).
File not open (error 205).
Bad connection (error 221).
All disk errors (101 - 108).

WinAllocateWindowMemory

```
FUNCTION WinAllocateWindowMemory (width : Integer;
                                  height : Integer;
                                  format : WindowFormat;
                              VAR error : Word): WindowRegionPtr;
```

Purpose and Operation

This call allocates memory for an alternate window. It frees an application from concerning itself with the number of bits per pixel required by the screen. The application must specify whether the window region is to be used as a GRiD format window or a host (non-GRiD) screen format window. If the alternate window is to be used to load screenimage files, then it should be in GRiD format. If the alternate window is only going to be used to redirect the output so that the user doesn't see it, then it should be in the screen format. In screen format, transfers between windows will be accomplished more quickly.

A pointer is returned to the WindowRegion record for this window. The organization of the WindowRegion record is as follows:

```
TYPE
  WindowFormat = (screenFormat, GRiDFormat);

  WindowRegion = RECORD
                   format : WindowFormat;
                   width : Integer;
                   height : Integer;
                   bufLength : Word;
                   buf : Pointer;
                   bitsPerPel : Byte;
                   bytesPerLine : Word;
                 END;

WindowRegionPtr = ^WindowRegion;
```

Note: To deallocate memory for a window, you must use two OsFree calls -- one to free the WindowRegionPtr and one to free the "buf" pointer.

Parameters

width -- the width of the window in pixels.
height -- the height of the window in pixels.
format -- GRiD format or host screen format.

WindowRegion Record Fields

format -- GRiD format or host screen format.
width -- the width of the window in pixels.
height -- the height of the window in pixels.
bufLength -- the size, in bytes, of the buffer allocated by the system for

this window.
buf -- a pointer to the first byte of the buffer allocated for this window.
bitsPerPel -- the number of bits-per-pixel used for the window. For GRiD format windows, there is one bit per pixel.
bytesPerLine -- the number of bytes used by the system to store one horizontal line of pixels for the allocated window.

Function Return

WindowRegionPtr -- a pointer to the WindowRegion record for this window.

WinClipLine

```
FUNCTION WinClipLine (VAR x1, y1, x2, y2: Integer) : Boolean;
```

Purpose and Operation

This function tells you if any portion of a line (defined by x1,y1 and x2,y2) extends outside of the current clipping rectangle. If clipping would occur, the variables x1, y1, x2, y2 contain the coordinates of the line as it will be clipped and the function returns a True Boolean value. If the line lies completely inside the window, this function returns FALSE and the unchanged coordinates of the line are returned. Note: This function neither draws nor clips the line; use WinDrawLine to draw the line -- it will be clipped as necessary by the clipping rectangle. You can use WinClipLine to determine if a line would be drawn completely outside of a clipping rectangle and thus skip the WinDrawLine if the line would not be displayed within the rectangle.

Parameters

x1,y1, x2,y2 -- the two window relative pixel coordinates defining the line. On entry, they define the line that is to be checked for clipping. On return, they define the line as it would be clipped.

WinClipRectangle

```
PROCEDURE WinClipRectangle (VAR r : Rectangle);
```

Purpose and Operation

This function tells you if any portion of a rectangle (r) extends outside of the current clipping rectangle. If clipping would occur, the variable r contains the coordinates of the rectangle as it will be clipped. If the rectangle lies completely inside the window, the unchanged coordinates of the rectangle are returned. Note: This function neither draws nor clips the rectangle.

Paramters

r -- the rectangle that is to be clipped. On return, contains the clipped dimensions of the rectangle.

WinCopyRectangle

```
PROCEDURE WinCopyRectangle (VAR r: Rectangle;
                                newTopLeft: Point);
```

Purpose and Operation

This procedure copies an area defined by the rectangle r into another rectangular area of the window. The new rectangular area is the same size as the original, but its top left corner is at the pixel position newTopLeft in the window. The new rectangle will be clipped as necessary to be displayed within the clipping rectangle of the window. WinCopyRectangle copies the areas point by point and overwrites all pixels in the copy location.

Parameters

r -- the source rectangle whose contents are to be copied. On return, this variable indicates the size of the resultant destination rectangle (possibly clipped).
newTopLeft -- the upperleft corner position where the rectangle is to be copied.

WinCopyRemoteRectangle

```
PROCEDURE  WinCopyRemoteRectangle (source : WindowRegionPtr;
                                   dest: WindowRegionPtr;
                                VAR r : Rectangle;
                                   newTopLeft : Point;
                                   mode : WORD);
```

Purpose and Operation

This routine lets you copy a rectangle from one window region to another. If either the source or destination window regions are NIL, then the screen is assumed to be the source or destination. If either source or destination window are in GRiD format, then the data is not only copied, but is also translated to the different format required by the destination window. If both window regions are in GRiD format, then this routine will keep the data in GRiD format. The mode parameter is currently reserved for future use and its value must be zero in order for the routine to function properly.

Parameters

source -- a pointer to the window from where the rectangle is being copied.
dest -- a pointer to window to which the rectangle is being copied.
r -- on entry, specifies the size of the source rectangle that is to be copied; on return, the size of the source rectangle as it was clipped to fit in the destination window. Note: the source and destination rectangles are both clipped to the window bounds -- not the clipping rectangle bounds.
newTopLeft -- the pixel coordinates of the top left corner of the destination window.
mode -- reserved for future use. Must be set to zero.

WinDrawChar

```
PROCEDURE WinDrawChar(ch: Char; x,y: Integer);
```

Purpose and Operation

This procedure draws a character in the window, given the window relative pixel coordinates where the top left corner of the character is to appear. Nothing is drawn if any part of the character would be clipped because it lies outside the window. The size of the character drawn is dependent on which font is currently loaded.

Parameters

ch -- the 8-bit ASCII value for the character to be displayed. Note: Because of internal requirements and for historical reasons, two ASCII codes draw characters other than the characters you would expect. If ch is 0CDh the font character represented by 80h is drawn and if ch is 0F7h the font character represented by 86h is drawn.

x,y -- the window-relative pixel location where the upper left corner of the character is to be drawn.

WinDrawChars

```
PROCEDURE WinDrawChars(VAR ch: Bytes;
                           count, x, y: Integer);
```

Purpose and Operation

Beginning with character ch in a text buffer, the procedure outputs a character string that is "count" characters long. It positions the upper left pixel of the first character at the window-relative pixel coordinate (x,y).

Example

The following procedure call draws a character string in the window. The top left pixel of the first character appears at pixel (20, 20) of the window.

```
WinDrawChars(str^.chars[1], str.len, 20, 20);
```

Parameters

ch -- a pointer to the first character in a text buffer that is to be output.
count -- the number of characters to be drawn.
x,y -- the window-relative pixel location where the upper left corner of the first character is to be drawn.

WinDrawLine

```
PROCEDURE WinDrawLine (x1,y1, x2,y2: Integer);
```

Purpose and Operation

This procedure draws a line within the window. Any portions of the line lying outside the window are clipped.

Parameters

x1,y1, x2,y2 -- the window-relative pixel coordinates defining the two end points of the line to be drawn.

WinDrawPixel

```
PROCEDURE WinDrawPixel (x,y: Integer);
```

Purpose and Operation

This procedure draws a single pixel at the given window coordinate. If the pixel lies outside the window bounds, it will be clipped (not drawn).

Parameters

x, y -- the window-relative pixel coordinate where the pixel is to be drawn.

WinEraseChar

```
PROCEDURE WinEraseChar (x,y: Integer);
```

Purpose and Operation

This procedure will erase a character position (of dimensions charHeight by charWidth) even if portions of it extend out of the window bound.

Parameters

x, y -- the window-relative pixel coordinate where the top left pixel of the character to be erased is located.

WinEraseLine

```
PROCEDURE WinEraseLine (x1,y1, x2,y2: Integer);
```

Purpose and Operation

A line within the current window is erased. Any portion of the line lying outside the current window boundaries will not be affected.

Parameters

x1,y1, x2,y2 -- the window-relative pixel coordinates defining the two end points of the line to be erased.

WinErasePixel

```
PROCEDURE WinErasePixel (x,y: Integer);
```

Purpose and Operation

This procedure erases a single pixel at the given window-relative coordinate. If the pixel lies outside the window bounds, no action is taken.

Parameters

x, y -- the window-relative pixel coordinate of the pixel is to be erased.

WinEraseRectangle

```
PROCEDURE WinEraseRectangle (VAR r: Rectangle);
```

Purpose and Operation

This procedure erases a rectangle in the window. If two rectangles overlap and one is erased, the other one will not be restored: the procedure changes the display's bit map directly. Any portion of the rectangle lying outside the current window is not affected.

Parameters

r -- the rectangle that is to be erased. On return, this variable indicates the rectangle that was actually erased since portions outside the window are not affected.

WinEraseWindow

PROCEDURE WinEraseWindow;

Purpose and Operation

This procedure erases the contents of the current window, but not the surrounding frame.

WinFrameWindow

PROCEDURE WinFrameWindow;

Purpose and Operation

This procedure draws a one-pixel (thin) frame outside the current window bounds. The frame will not be drawn if it has been disabled in the user profile via GRiDManager's Option command.

WinGetWindowExtent

```
PROCEDURE WinGetWindowExtent (VAR extent : Point);
```

Purpose and Operation

This procedure tells you the size of your window by returning the variable extent. Extent only tells you how big your current window is; it does not indicate where the window is on the screen. Because windows can be placed anywhere on the screen, only the size of the window (and not its location) is important.

Typically an application will use this call during initialization to determine how big a window it has to work with. Since GRiD-OS reserves the right to change your window size at any time, applications should be designed to run independent of the window size and screen characteristics.

If GRiD-OS does change your window size, it places a special character (0C3 hex) in the keyboard queue. When an application receives this character, it should assume that the dimensions of the window have been altered and must recalculate the window size (using WinGetWindowExtent) and redisplay the window using the new boundaries.

WinInitDefaultWindow

```
PROCEDURE WinInitDefaultWindow;
```

Purpose and Operation

This procedure sets the window to the entire screen, erases it, and draws a one-pixel frame surrounding the screen. The frame will not be drawn if it has been disabled in the user profile via GRiDManager's Option command.

WinInvertChar

```
PROCEDURE WinInvertChar (x,y: Integer);
```

Purpose and Operation

The procedure performs an exclusive OR (XOR) on all the pixels of a character position (charHeight by charWidth). Any portion of the character outside of the window bounds is not affected.

Parameters

x,y -- the window-relative pixel location of the upper left-hand corner of the character.

WinInvertLine

```
PROCEDURE WinInvertLine(x1,y1, x2,y2: Integer);
```

Purpose and Operation

This procedure performs an exclusive OR (XOR) operation on the given line, inverting it within the window. Any portion of the line outside of the window bounds is not affected.

Parameters

x1,y1 x2,y2 -- the window-relative pixel locations defining the two end points of the line to be inverted.

WinInvertPixel

```
PROCEDURE WinInvertPixel (x,y: Integer);
```

Purpose and Operation

The procedure performs an exclusive OR (XOR) operation with the single pixel position specified. If the pixel lies outside the current window, no action is taken.

x,y -- the window-relative coordinate defining the pixel to be inverted.

WinInvertRectangle

```
PROCEDURE WinInvertRectangle (VAR r: Rectangle);
```

Purpose and Operation

This procedure inverts the bit-map area inside a rectangle in the window. Any portion of the rectangle lying outside the current window is not affected.

Parameters

r -- the rectangle that is to be inverted. On return, this variable indicates the rectangle that was actually inverted since portions outside the window are not affected.

WinLoadFont

```
FUNCTION WinLoadFont (conn : Word;
                      VAR error : Word): FontPointer;
```

Purpose and Operation

This routine loads a font file into memory and returns a pointer to the font that can subsequently be used by the WinSetFont function. Before the font can be loaded, you must first attach (OsAttach) and open (OsOpen) the file. WinLoadFont also does not detach the file; you must close and detach the file when finished with it.

NOTE: There are also font handling procedures provided in the common code package. Those are higher level calls and therefore may be easier to use.

Parameters

conn -- conncection number (data type Word) obtained from OsAttach that specifies the font file to be loaded.

WinResetClip

```
PROCEDURE WinResetClip;
```

Purpose and Operation

Resets the clipping rectangle to the entire window. Clipping by the window boundaries and by the clipping rectangle will now be the same.

WinScrollRectangle

```
PROCEDURE WinScrollRectangle (VAR r: Rectangle;
                                  dir: Direction;
                                  distance: Integer);
```

Purpose and Operation

The procedure scrolls a rectangle in the given direction by the distance given in pixels. Portions of the rectangle scrolled out of the display window are clipped. An open area is left when the rectangle scrolls away from its original location. WinScrollRectangle returns the coordinates of the open space as the rectangle r, without modifying the space. The application must update the open area.

Parameters

r -- on entry, defines the rectangular area to be scrolled. On return, defines the rectangular open area freed by the scrolling that can now be updated by the application.
dir -- the direction (up, down, left, right) in which the rectangle is to be scrolled.
distance -- the number of pixels that the rectangle is to be scrolled.

WinScrollWindow

```
PROCEDURE WinScrollWindow (VAR r: Rectangle;
                           dir: Direction;
                           distance: Integer);
```

Purpose and Operation

The procedure scrolls the entire window in the given direction by the distance given in pixels. The window will leave an empty area when it scrolls away from its location. WinScrollWindow returns the coordinates of the empty area as the rectangle r, without modifying the area, so that the application can update the area. Anything scrolled beyond the window bounds will be clipped.

Note: The rectangle r, which you specify in window-relative pixel coordinates, acts as an output parameter only. It returns the rectangular coordinates of an area that the application should update. The procedure needs no input parameter for the window bounds because it obtains them directly.

Parameters

r -- On return, defines the rectangular open area freed by the scrolling that can now be updated by the application.
dir -- the direction (up, down, left, right) in which the window is to be scrolled.
distance -- the number of pixels that the window is to be scrolled.

WinSetAlternateWindow

```
PROCEDURE  WinSetAlternateWindow (alt : WindowRegionPtr);
```

Purpose and Operation

This call forces all subsequent window calls to be performed on the alternate window specified. The alternate window must be in the host screen format. If not, then this routine does nothing. If the WindowRegionPtr is NIL, then the screen is assumed.

Parameters

alt -- specifies the pointer for the alternate window.

WinSetClip

```
PROCEDURE WinSetClip (VAR r : Rectangle);
```

Purpose and Operation

Sets a clipping rectangle within the window boundaries. This clipping is in addition to the clipping performed automatically at the window boundaries. Note that the clipping rectangle is defined in pixel coordinates -- it is independent of the visible and constraint parameters defined by common code routines for tables, menus, and forms.

This procedure makes displaying multiple views quite easy. For example, in displaying two different views from the same application, each view would draw an entire window full, just as if it were the only view. But the clipping window would be set to different parts of the screen for each view. You only modify the clipping to display different views; you need not modify your window display code.

Parameters

r -- on entry, the coordinates defining the boundaries of the clipping rectangle being established. On return, the coordinates of the actual clipping rectangle established; the actual rectangle may differ from the specified rectangle since any portion of the rectangle lying outside the window boundaries is clipped.

WinSetFont

```
FUNCTION WinSetFont (font : FontPointer;
                     VAR info : FontInfoRecord;
                        count : Word): FontPointer;
```

Purpose and Operation

This function sets the designated font as the new font. The variable info can be examined upon return to check the characteristics of the current font. The function returns a pointer to the font that was loaded prior to this call; that is, a pointer to the previous font. The format of the FontInfoRecord is as follows:

```
FontInfoRecord = RECORD
                   charWidth : Byte;
                   charHeight : Byte;
                   lineHeight : Byte;
                   baseLine : Byte;
                 END;

FontPointer = ^Byte;
```

NOTE: The four bytes in this record can be examined directly using the function calls charWidth, charHeight, lineHeight, and baseLine described earlier in this chapter.

Parameters

font -- a pointer (obtained from the function WinLoadFont) to the font that is to be set as the current font. If font has a value of zero, the system font in PROM is set. If a Null pointer is specified, the current font is left in place and info on the current font is returned.
info -- the four byte FontInfoRecord describing the dimensions of the font.
count -- determines how many bytes of the FontInfoRecord will be returned in the variable info. For example, if count has a value of two, only charWidth and charHeight are returned.

Function Return

font -- a pointer to the previous font. You should save this pointer so that you can later restore the initial font before exiting.

WinSetWindow

```
PROCEDURE WinSetWindow(VAR w: Rectangle);
```

Purpose and Operation

This procedure changes the size and location of your current window. It sets the window size to the rectangle it receives as an argument. The additional clipping rectangle within this window is reset to this size too.

WinSetWindow is the only procedure that requires absolute screen coordinates. The rectangle must be defined in absolute screen coordinates because no window-relative coordinates are valid for this window until the procedure has finished. Windows larger than the display screen boundaries (screenHeight by screenWidth) will be clipped. You must leave a single pixel space on all four margins if you want a frame -- this procedure can claim the outermost absolute pixel positions if you request it. However, you must call WinFrameWindow to actually draw the frame.

APPENDIX A. COMPASS KEYBOARD CODES

Table A-1 on the following page lists all the hexadecimal codes that can be generated from the Compass keyboard. Since various combinations of the CODE and SHIFT keys are used in GRiD applications, all of the codes that result from the key combinations are shown in the table.

Table A-1. Compass Keyboard Codes

Key	Unshifted	SHIFT	CODE	CODE-SHIFT	CTRL	SHIFT-CTRL	CODE-CTRL	CODE-SHIFT-CTRL
'	27 (')	22 (")	60 (`)	5C (\)	07 (BEL)	02 (STX)	00 (NUL)	1C (FS)
,	2C (,)	3C (<)	5B ([)	7B ({)	0C (FF)	1C (FS)	1B (ESC)	1B (ESC)
-	2D (-)	5F (_)	AD	7F DEL	0D (CR)	1F (US)	8D	1F (US)
.	2E (.)	3E (>)	5D (])	7D (})	0E (SO)	1E (RS)	1D (GS)	1D (GS)
/	2F (/)	3F (?)	BF	BF	0F (SI)	1F (US)	9F	9F
0	30 (0)	29 ())	B0	A9	10 (DLE)	09 (HT)	90	89
1	31 (1)	21 (!)	B1	A1	11 (DC1)	01 (SOH)	91	81
2	32 (2)	40 (@)	B2	C0	12 (DC2)	00 (NUL)	92	80
3	33 (3)	23 (#)	B3	A3	13 (DC3)	03 (ETX)	93	83
4	34 (4)	24 ($)	B4	A4	14 (DC4)	04 (EOT)	94	84
5	35 (5)	25 (%)	B4	A5	15 (NAK)	05 (ENQ)	95	85
6	36 (6)	5E (^)	B6	DE	16 (SYN)	1E (RS)	96	9E
7	37 (7)	26 (&)	B7	A6	17 (ETB)	06 (ACK)	97	86
8	38 (8)	2A (*)	B8	AA	18 (CAN)	0A (LF)	98	8A
9	39 (9)	28 (()	B9	A8	19 (EM)	08 (BS)	99	88
;	3B (;)	3A (:)	7E (~)	7C (|)	1B (ESC)	1A (SUB)	1E (RS)	1C (FS)
=	3D (=)	2B (+)	BD	AB	1D (GS)	0B (VT)	9D	8B
A	61 (a)	41 (A)	E1	E1	01 (SOH)	01 (SOH)	81	81
B	62 (b)	42 (B)	E2	E2	02 (STX)	02 (STX)	82	82
C	63 (c)	43 (C)	E3	E3	03 (ETX)	03 (ETX)	83	83
D	64 (d)	44 (D)	E4	E4	04 (EOT)	04 (EOT)	84	84
E	65 (e)	45 (E)	E5	E5	05 (ENQ)	05 (ENQ)	85	85
F	66 (f)	46 (F)	E6	E6	06 (ACK)	06 (ACK)	86	86
G	67 (g)	47 (G)	E7	E7	07 (BEL)	07 (BEL)	87	87
H	68 (h)	48 (H)	E8	E8	08 (BS)	08 (BS)	88	88
I	69 (i)	49 (I)	E9	E9	09 (HT)	09 (HT)	89	89
J	6A (j)	4A (J)	EA	EA	0A (LF)	0A (LF)	8A	8A
K	6B (k)	4B (K)	EB	EB	0B (VT)	0B (VT)	8B	8B
L	6C (l)	4C (L)	EC	EC	0C (FF)	0C (FF)	8C	8C
M	6D (m)	4D (M)	ED	ED	0D (CR)	0D (CR)	8D	8D
N	6E (n)	4E (N)	EE	EE	0E (SO)	0E (SO)	8E	8E
O	6F (o)	4F (O)	EF	EF	0F (SI)	0F (SI)	8F	8F
P	70 (p)	50 (P)	F0	F0	10 (DLE)	10 (DLE)	90	90
Q	71 (q)	51 (Q)	F1	F1	11 (DC1)	11 (DC1)	91	91
R	72 (r)	52 (R)	F2	F2	12 (DC2)	12 (DC2)	92	92
S	73 (s)	53 (S)	F3	F3	13 (DC3)	13 (DC3)	93	93
T	74 (t)	54 (T)	F4	F4	14 (DC4)	14 (DC4)	94	94
U	75 (u)	55 (U)	F5	F5	15 (NAK)	15 (NAK)	95	95
V	76 (v)	56 (V)	F6	F6	16 (SYN)	16 (SYN)	96	96
W	77 (w)	57 (W)	F7	F7	17 (ETB)	17 (ETB)	97	97
X	7B (x)	58 (X)	F8	F8	18 (CAN)	18 (CAN)	98	98
Y	79 (y)	59 (Y)	F9	F9	19 (EM)	19 (EM)	99	99
Z	7A (z)	5A (Z)	FA	FA	1A (SUB)	1A (SUB)	9A	9A
BACKSPACE	08	C8	88	8A	08 (BS)	88	88	8A
RETURN	0D	CD	8D	8C	0D (CR)	8D	8D	8C
DownArrow	C4	CE	D2	D6	84	BE	92	96
ESC	1B	1B	9B	9B	1B (ESC)	1B (ESC)	9B	9B
LeftArrow	C6	D0	D4	D8	86	90	94	98
RightArrow	C7	D1	D5	D9	87	91	95	99
Spacebar	20 (SP)	20 (SP)	20 (SP)	20 (SP)	00 (NUL)	00 (NUL)	00 (NUL)	00 (NUL)
TAB	09	C9	89	8B	09	89	89	8B
UpArrow	C5	CF	D3	D7	85	BF	93	97

INDEX

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W