Issue 3, 1990

Personal Systems

IBM Personal Systems Technical Solutions

IBM®

IBM Personal Systems Technical Solutions is published by the U.S. Marketing and Services Group, International Business Machines Corporation, Roanoke, Texas, U.S.A.

Editor	Ed Bamberger
Consulting Editor	Libby Boyd
Technical Consultant	Dave Dill
Automation Consultant	Andy Frankford
Little Solutions	Larry Pollis
Communications	Elisa Davis
Layout, Cover Design	Corporate Graphics
Illustrator	Bill Carr
Manager	Bill Hawkins

To correspond with *IBM Personal Systems Technical Solutions*, please write the editor at:

IBM Corporation
Internal Zip 40-A2-04
One East Kirkwood Blvd.
Roanoke, TX 76299-0015

To subscribe to this publication, use an IBM System Library Subscription Service (SLSS) form, available at IBM branches, and specify form number GBOF-1229.

Contents

Software

1 DOS – A Look Under the Hood to See How It Spins

12 Memory Management in a DOS Environment

20 FASTOPEN – The DOS Performance Enhancer

24 DOS 4.00 Compatibility Issues

28 'Out of Environment Space' Errors

34 A New LAN Requester for DOS Systems

43 Creating a Dialog Box Dynamically Using WinCreateDlg

53 An Alternative for the OS/2 START Command

Random Data

65 CUA: A Consistent Interface

69 Little Solutions

71 New Products

DOS – A Look Under the Hood to See How It Spins

Pylee Lennil
IBM Corporation
Boca Raton, Florida

This article provides an in-depth look into the internal workings of IBM PC DOS. It describes PC DOS major components, which include: COMMAND.COM, IBMDOS, IBMBIO, device drivers, File Allocation Table, disk structure, root directory, master boot record, and BIOS parameter block. Included is a detailed description of how these components are interconnected and how they interface from user level to hardware level.

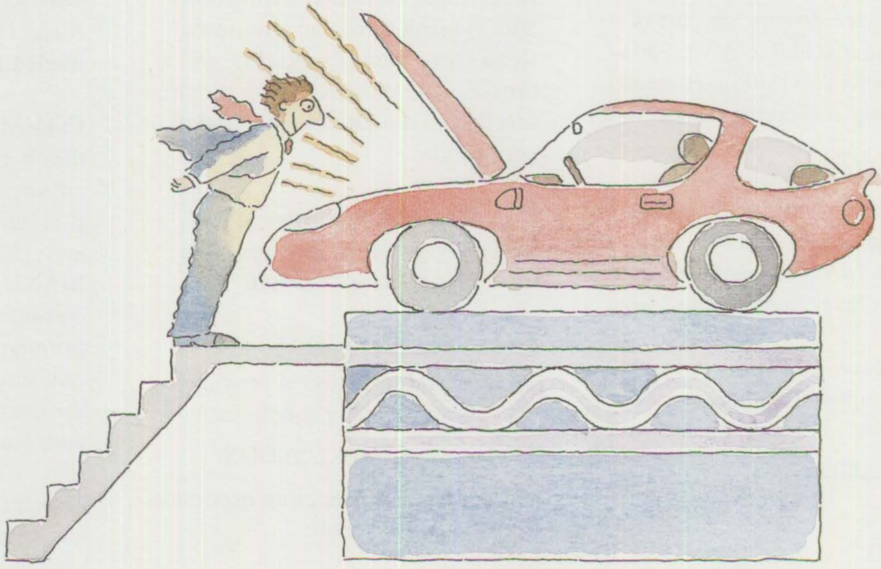
Turn on the PC, and within a few seconds you will notice the DOS C: prompt indicating that DOS is up and running. What happens during the first few seconds? How does DOS load, initialize itself and start running? Type DIR, and DOS displays directory entries. How does DOS search for a file? How does DOS access file sectors on the disk? How does it execute a program?

Before we look into the details of DOS, let's start with a general overview. DOS is the most widely used PC operating system. It is a single-task operating system that executes one program at a time. Basically, an operating system is a program that manages the system resources, such as memory, disk, diskette, console, keyboard, and printer. The operating system makes these resources available to the application through operating system services. These services include allocation and deallocation of memory, manipulation of files and directories, and other miscellaneous operating system functions.

To access the system services, the operating system provides a set of functions to the application. Similarly, DOS provides a set of functions (INT 20H, INT 21H, INT 24H, INT 25H, and INT 26H). Instead of accessing the resources directly, the application calls these functions for DOS services. DOS, in turn, allows the application to access the system resources in a controlled manner.

DOS is basically composed of a user interface program (DOS shell), DOS kernel, and a set of DOS device drivers. These components reside in a bootable disk or diskette device. When the system is turned on, the PC ROM BIOS exercises the system hardware by executing the Power On Self Test (POST). It then initiates the DOS loading. During loading, a user can configure DOS using the system configuration commands in the CONFIG.SYS file. This file contains commands to load the installable device drivers as well as to specify the number of system buffers and tables needed by DOS.

Once DOS is running, users can run applications by either typing the program name or by specifying the name in a batch file. DOS supports two types of executable programs: Transient programs stay in memory only during execution time; Terminate-and-Stay-Resident (TSR) programs stay in memory until the system is boot loaded. TSR programs can be activated by pressing a special combination of keys.



DOS has a tree-structured file system that stores user files and directories. The file system consists of a root directory with subdirectories and files. DOS maintains special areas on the disk and diskette to store the information and locations of these files and directories as well as the disk-specific information. These areas are the file allocation table (FAT), root directory, master boot record, and boot sector. These areas are created when the FORMAT utility is used to format the disk or diskette.

DOS maintains the file system in both hard disk and diskette devices. A diskette device is always maintained as a single partition, while a hard disk can be divided into one or more partitions. Each partition is identified by a drive letter such as C:, D:, E:, and so forth. The disk partitioning is done using the FDISK program. FDISK also creates a master boot record that indicates the location of all partitions on a hard disk.

DOS supports internal device drivers and installable device drivers. A device driver provides device-independent hardware services. The internal device drivers are part of DOS, while installable device drivers are loaded and initialized during DOS loading and configuration time.

To learn how DOS works, let's peek under the hood, look at its major components, and see how they interact with one another from application level to hardware level.

At first glance, you will notice that DOS is composed of three major components:

- COMMAND.COM
- IBMDOS
- IBMBIO

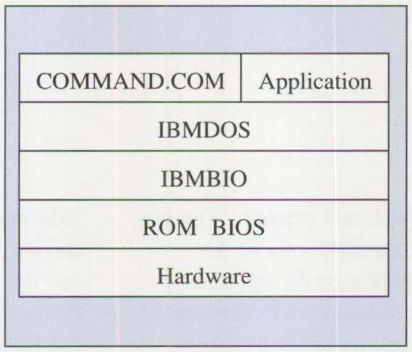


Figure 1. Basic DOS Structure

Figure 1 shows how these components form several layers between the application and the hardware. Let's examine each layer.

Applications access DOS services through DOS function calls. COMMAND.COM provides the user interface. IBMDOS is known as the DOS kernel and provides hardware-independent DOS services to the applications.

IBMBIO loads and initializes IBMDOS and COMMAND.COM, and it contains a set of internal device drivers that are used by the kernel to access the hardware devices.

Basic Input/Output system (ROM BIOS) resides permanently in the Read-Only Memory (ROM), and it provides the most basic input/output services to the DOS kernel and to applications.

Now that we have observed the major components, let's examine these components in detail.

COMMAND.COM

The COMMAND.COM provides the following major functions:

- The user interface to DOS
- Loading and executing user commands or programs
- Redirecting output of one command to input of another command
- Processing batch files

COMMAND.COM is loaded by IBMBIO during the last phase of the DOS initialization. As shown in Figure 2, there are two sections in COMMAND.COM. The first is a resident section, which is loaded just above the DOS system buffers. This section contains setup functions, Ctrl-Break and Critical Error Handling routines. The second section is the transient section that is loaded into the upper memory just below 640 KB. This section contains code for batch file handling and DOS internal commands such as DIR, DEL, and COPY. The transient part may eventually be overwritten by the application. If that happens, it will be reloaded when the application returns control to COMMAND.COM.

COMMAND.COM can execute two types of commands: 1) internal DOS commands such as DIR, CD, and PATH, and 2) external commands, which can be either DOS utilities such as XCOPY and BACKUP, or applications, or batch files. External commands must be loaded from disk.

COMMAND.COM checks the user's command to see whether it is an internal or external command. If it is internal, it is executed immediately, because it is part of COMMAND.COM. If it is an external command (an .EXE, .COM or .BAT program), the current drive and current directory will be searched for the program, unless the program path was specified in the command line. COMMAND.COM will search first for .COM, then .EXE, and finally a .BAT program. Additional drives and directories will be searched if they are specified in the PATH command. If the file is found, it will be loaded into memory and executed. If the file is not found in any of the directories, the "Bad command or file name" message is displayed.

IBMDOS (DOS Kernel)

IBMDOS provides the following hardware independent DOS services to applications:

- Directory Operations – MKDIR, CHDIR, RMDIR, FIND-FIRST, FIND-NEXT
- Dynamic Memory Allocation and Deallocation – Alloc, Dealloc, Setblock
- Block Device and Character Device I/O functions – Keyboard Input, Display
- Critical Error Generation – INT 24H
- File Operations – CREATE, OPEN, READ, WRITE, CLOSE, DELETE
- Miscellaneous functions – Get Date and Time, Get Disk Information
- Disk I/O – Reset Disk, Select Disk

IBMDOS is loaded by IBMBIO during the system initialization and configuration time. It is composed of the following types of routines:

- DOS function call dispatcher routine
- DOS function call handler routines
- File system support routines

These routines are further composed of the following interrupt handlers:

- INT 00H – Divide Overflow Handler

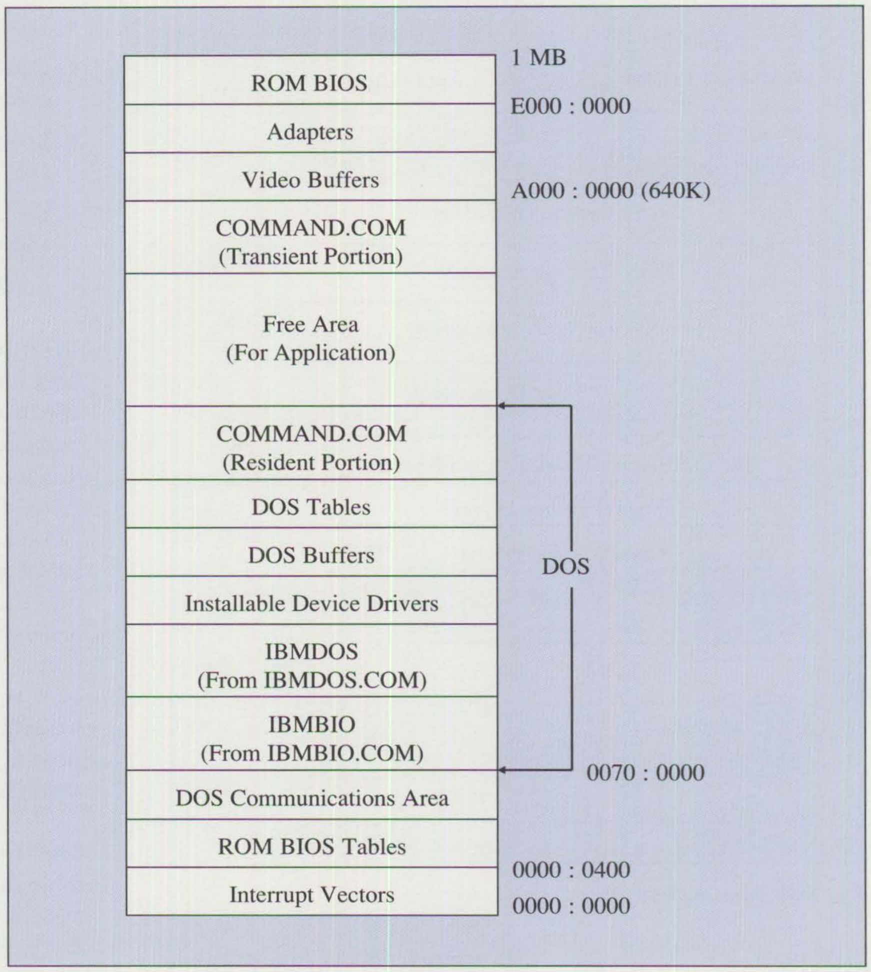


Figure 2. DOS Memory Layout

- INT 20H – Program Termination Handler
- INT 21H – DOS Function Call Handler
- INT 25H – Absolute Disk Read Handler
- INT 26H – Absolute Disk Write Handler
- INT 27H – DOS Callbacks Handler
- INT 2FH – Multiplex Interrupt Handler

A DOS INT 21H function call by an application generates an INT 21H interrupt. This interrupt is intercepted by IBMDOS. Figure 3 shows the function dispatcher, which gets the control first. The dispatcher checks the interrupt's function code and calls the appropriate DOS function handler routine to handle the call. If the call requires file or directory access, it calls the file system support routines. These routines search for the file and read the data from the disk using the DOS internal disk driver provided by IBMBIO. Upon completion, IBMDOS returns control back to the caller with the results in registers and buffers. Each of the other DOS interrupts are handled by their own interrupt handlers within IBMDOS.

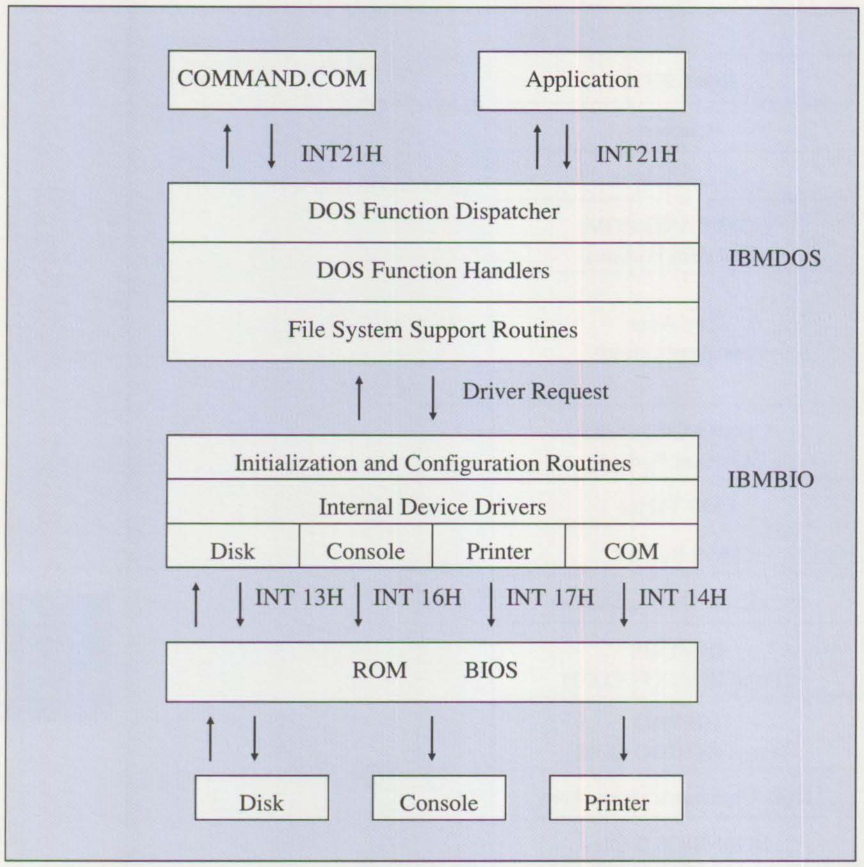


Figure 3. Flow of DOS Function Call

System Buffers

Figure 4 shows DOS system buffers, which IBMDOS maintains. This set of circular buffers keeps the sectors read during a disk read operation. The number of buffers to be allocated can be specified by the BUFFERS=N parameter in the CONFIG.SYS file. Each buffer has a buffer header and a 512-byte sector data area. The header contains the sector number, the drive associated with the sector, and flags indicating whether the buffer is free or in use. The system buffers are created by IBMBIO during system configuration time.

In addition to holding data before it being passed to the application, these buffers improve system performance by caching the recently read disk sectors. When an application issues a disk read, IBMDOS checks these buffers to see whether the desired sector already exists in one of the buffers from a previous read. If it exists, it simply copies the sector from this buffer to the application buffer, avoiding an extra disk read and improving performance. As additional sectors are read, the available buffers will be filled. Data in the least recently used buffer sectors are replaced by the latest data read from the disk.

IBMBIO

The major functions of IBMBIO are:

- Initializing the hardware and DOS internal device drivers
- Setting up the disk parameter block chain
- Loading IBMDOS
- Configuring the system using commands in CONFIG.SYS file
- Loading and initializing installable device drivers
- Setting up DOS system buffers and tables
- Loading COMMAND.COM

IBMBIO is loaded during system boot time. As shown in Figure 3, IBMBIO contains DOS configuration code and DOS internal device drivers. These drivers – CON, AUX, PRN, CLOCK, COM1, COM2, COM3, COM4, LPT1, LPT2 and LPT3 – are used to access the video, keyboard, printer and asynchronous communication ports. The system configuration code is needed only during DOS loading and initialization time. This code is discarded once the configuration is done.

The first task of IBMBIO is to determine the types of hardware on the system, especially the types of disk drives, console, keyboard, printer

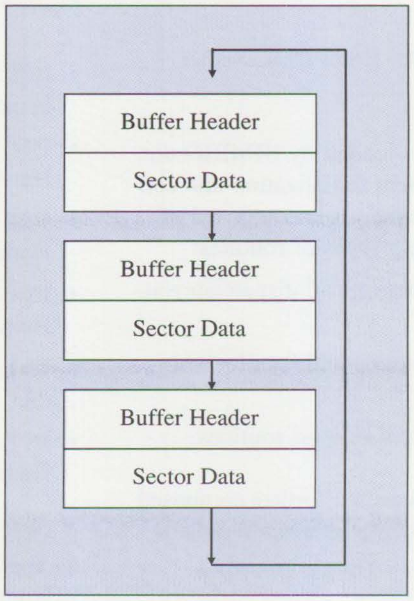


Figure 4. DOS System Buffers

and asynchronous communication lines. It then initializes the internal drivers to support the preceding devices. Next, IBMBIO creates a disk parameter block table for every disk partition and the diskette drives on the system. The table, known as Block Device Structure (BDS), contains disk-specific information. During this process, a drive letter will be assigned to each partition. The BDS tables are chained, and they will be searched by IBMDOS if disk-specific information is needed for a DOS service call. IBMBIO then loads IBMDOS from the IBMDOS.COM file on the boot disk.

Once IBMDOS is loaded, the basic DOS functions are available for further system configuration work. IBMBIO opens the system configuration file CONFIG.SYS and parses the commands. These commands could be BUFFERS=, DEVICE=, INSTALL=, BREAK=, and so on. If, for example, BUFFERS=10 is found, it creates ten system buffers, each with 512 bytes. If it is a DEVICE=XXXXXX.SYS, it loads and installs the installable device driver XXXXXX.SYS. It also loads SHARE if a disk partition greater than 32 MB exists on the disk. The final task of the IBMBIO is to load and pass control to COMMAND.COM. Figure 2 shows the DOS memory map once DOS is fully loaded.

Device Driver

A device driver is a DOS extension that contains code to handle specific hardware devices. It is the lowest DOS level, which allows all other DOS levels to work independent of the hardware devices. Device drivers allow enhancements to the device handling routines to be added independently of the DOS kernel.

There are two types of device drivers:

- Internal Device Drivers – Internal device drivers reside permanently in IBMBIO. They are initialized during system initialization time. These drivers support the standard devices such as disk, console, keyboard, printer and asynchronous communication lines. These drivers are used only by the DOS kernel.
- Installable device drivers – Installable device drivers are loaded when they are needed using the DEVICE=XXXXXX.SYS command in the CONFIG.SYS file. For example, ANSI.SYS is an installable device driver, which is used to replace the DOS built-in console driver. During system configuration, IBMBIO loads and initializes these drivers just before loading COMMAND.COM.

Figure 5 shows the structure of a device driver, which is composed of a device header, a strategy routine, and an interrupt routine. Figure 6 shows the header, which is an eight-byte block structure. The header contains the link address to the next driver in the chain as well as the address of the strategy and interrupt routines.

The interrupt and strategy routines contain the driver code. Whenever a driver service is needed, the calling routine builds a request packet containing the type of driver and the function to be performed, then calls the strategy and interrupt routines to perform a specific driver function. The strategy routine saves the request packet address, then the requested function is performed by the interrupt routine. Figure 7 shows the structure of the request packet.

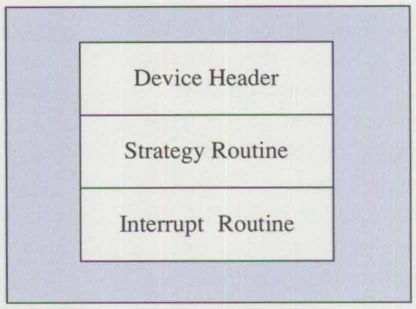


Figure 5. Device Driver Structure

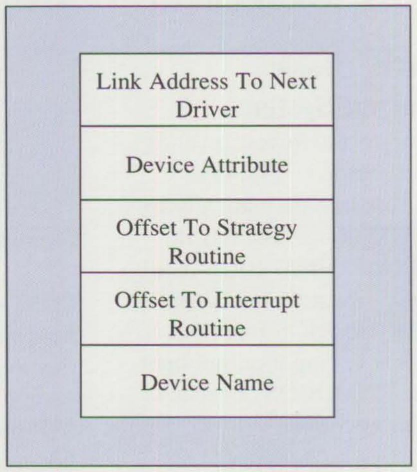

Link Address To Next Driver
Device Attribute
Offset To Strategy Routine
Offset To Interrupt Routine
Device Name

Figure 6. Device Header Structure

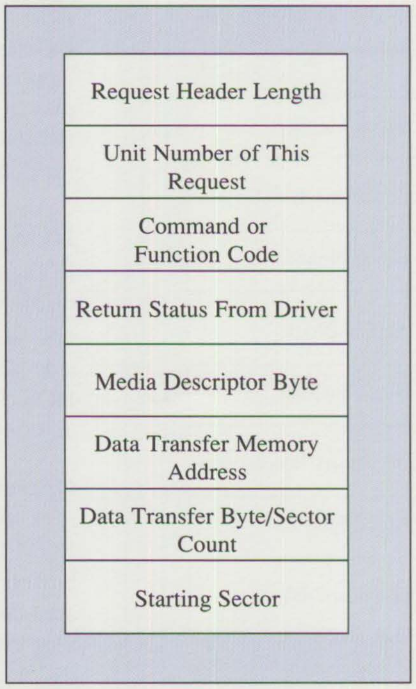

Request Header Length
Unit Number of This Request
Command or Function Code
Return Status From Driver
Media Descriptor Byte
Data Transfer Memory Address
Data Transfer Byte/Sector Count
Starting Sector

Figure 7. Request Packet Structure

For example, if IBMDOS needs to read a specific logical sector on the disk, it prepares a request packet. The size of the request packet may vary, depending on the type of driver service requested. IBMDOS then calls the DOS internal disk device driver strategy and interrupt routines. The strategy routine saves the request packet address, and the interrupt routine performs the function specified in the request packet.

ROM BIOS (Basic Input/Output System)

This software is the lowest level of input/output routines. Figure 2 shows this software permanently residing in ROM above 640 KB in segment E000H. ROM BIOS is not a part of DOS, even though DOS utilizes the ROM BIOS functions to perform low-level input/output operations. The ROM BIOS consists of the following sections:

- Power on self test (POST)
- Low-level I/O routines

When the system is turned on,

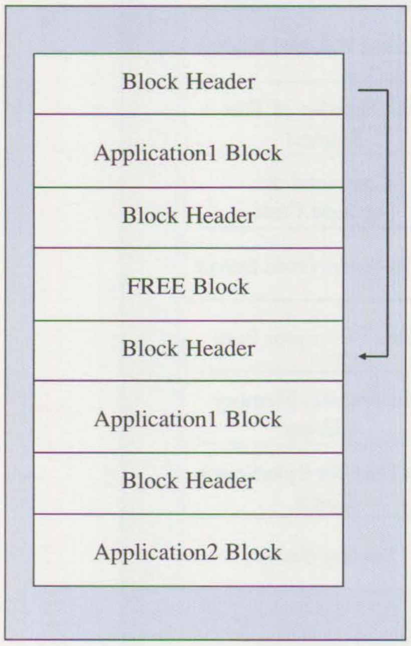


Figure 8. DOS Memory Blocks

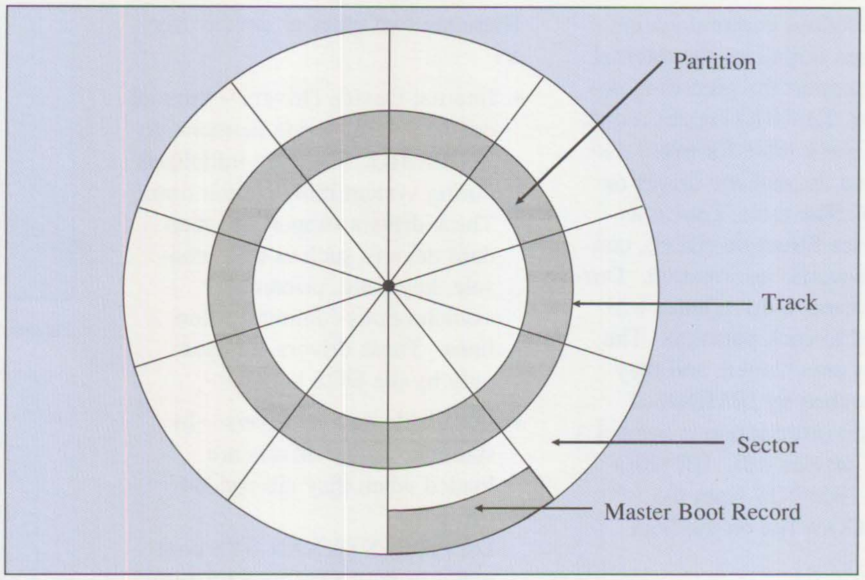


Figure 9a. DOS Disk Structure

POST routines exercise the system hardware, CPU, timer, DMA and memory, and also initiates the DOS system boot load. The low-level I/O routines, known as BIOS functions, provide a set of function calls for DOS and applications. For example, BIOS function INT 10H supports video, while INT 13H and INT 9 support disk and keyboard. Applications use INT 10H to display ASCII characters. The IBMBIO disk driver uses INT 13H for disk I/O.

DOS Memory Management

In Figure 2, DOS and applications reside below 640 KB. Video buffers, adapters, and ROM BIOS reside between 640 KB and 1 MB. DOS divides memory between 0 and 640 KB into two areas:

Operating System: Contains interrupt vectors, DOS communications, ROM BIOS tables, DOS code, DOS buffers and tables. The size of this area depends on the number of device drivers installed and number of disk buffers specified.

Applications: Contains application memory blocks and free memory blocks. DOS allocates memory to the application during run time. The memory is allocated in blocks. Figure 8 shows blocks associated with an application, which are chained together. When an application requests memory, IBMDOS looks through the memory block chain until a free block of the desired block size is found. This block is then linked to the application memory block chain. DOS provides the following memory management function calls to allocate, deallocate, and modify memory blocks:

48H – Allocate memory block (Alloc)

49H – Free memory block (Dealloc)

4AH – Modify memory block (Setblock)

Prior to loading a program for execution, DOS allocates available free memory to the application. This basically allocates the largest free

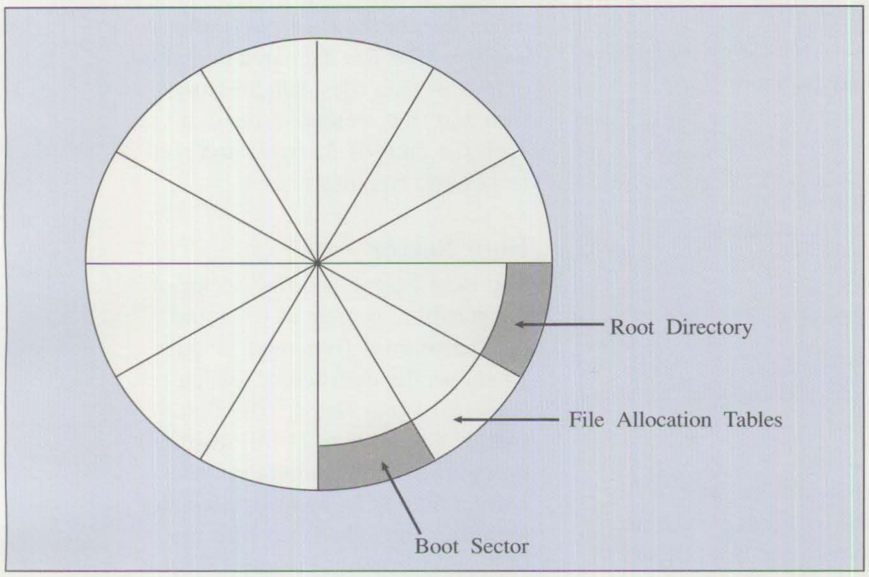


Figure 9b. Disk Partition Structure

block in the chain for the application. If the application needs more memory during run time, it has to first deallocate part of the memory, then, using the *Alloc* function, allocate the specific size of memory needed. The *Setblock* allows the application to change existing blocks. When an application terminates, DOS releases all memory blocks belonging to the application, except in the case of TSR programs. Blocks allocated to the TSR programs cannot be freed until the system is rebooted.

DOS Disk Structure

The disk is divided into tracks, and each track is subdivided into sectors, as shown in Figure 9a. A sector is the basic unit of storage for the disk. Sectors are made up of 512 bytes, and each byte consists of eight bits. During a disk read or write operation, the read/write head is positioned to the specific track first, and then the data on a specific sector is read as the sector passes by the head.

The hard disk can be divided into areas called partitions as shown in Figure 9a. Partitioning is done by the FDISK utility. The entire disk can be composed of one or more partitions. A diskette is maintained as a single partition. In Figure 9b, each partition is again subdivided into the following special areas by the FORMAT utility:

- Boot sector
- File Allocation Table (FAT)
- Root directory
- File data and directory data sectors

Master Boot Record and Partition Table

The Track 0, Head 0, Sector 1 of every hard disk contains a Master Boot Record created by the FDISK utility. Figure 10 shows the Master Boot Record, which consists of a loader program and a partition table. The purpose of this loader program is to load the Boot Sector, which is found in the bootable disk partition.

A hard disk can be divided into four areas. DOS uses the first two areas, which are called DOS Primary and DOS Extended partitions. The other two areas can be used by non-DOS operating systems such as OS/2. Figure 10 shows the partition table, which contains four entries that describe the area type and the

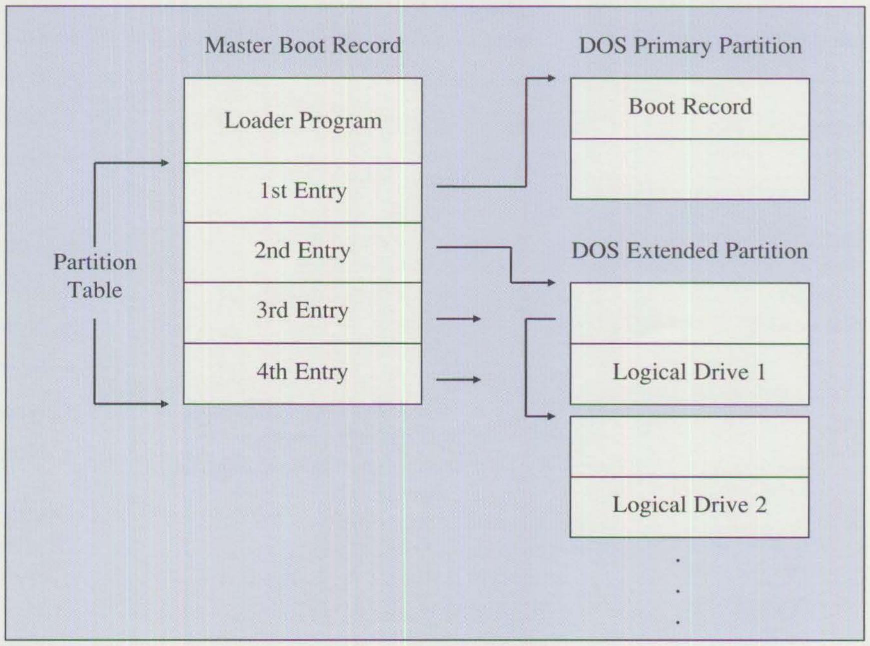


Figure 10. Layout of Master Boot Record and Partition Table

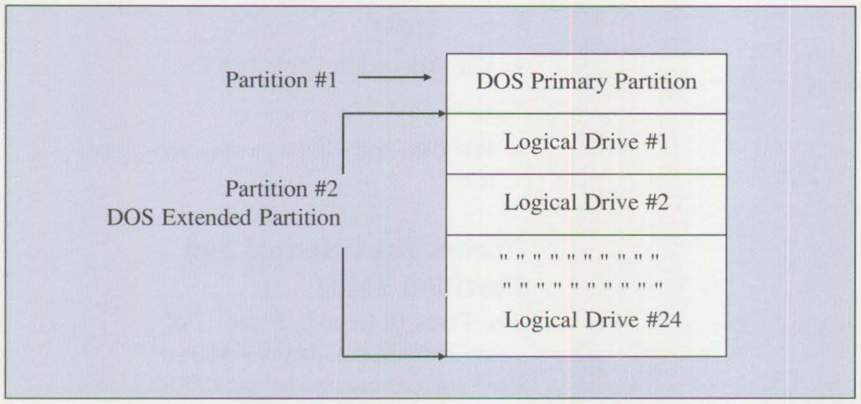


Figure 11. DOS Hard Disk Partitions

physical location of the four disk areas. Figure 11 shows the DOS extended partition, which can be subdivided into 24 subpartitions called the logical drives. DOS assigns the drive letter C to the primary partition and to each logical drive starting with C:.

Figure 12 shows the Partition Table Entry Structure. This table contains the Starting Head, Sector, and Cylinder, pointing to the beginning point for the related partition, the Boot Sector on the partition. The Ending Head, Sector, and Cylinder points to the end of the partition. The Partition Type indicates the type of file system in the partition as well as whether the partition is a primary or extended partition. The Boot Indicator field indicates whether the partition is bootable or non-bootable.

Figure 13 shows that the FORMAT utility divides each DOS partition into five areas. The first area is the Boot Sector, which contains a table of the disk characteristics as well as a boot program. The File Allocation Table contains locations of each block of every file on the disk. The disk root directory contains entries of a fixed number of files and directories in the file system root directory. The file data area contains entries of data files, subdirectory files and free sectors. Note that each file entry is 32 bytes and contains basic file information.

Boot Indicator	- Indicator whether the partition is bootable or nonbootable
	00 = Nonbootable
	80H = Bootable
Starting Head	- Starting head of the partition
Starting Sector	- Starting sector of the partition
Starting Cylinder	- Starting cylinder of the partition
Partition Type	- Type of file system in the partition
	00 = Type unknown
	01 = DOS 12-bit FAT file system
	04 = DOS 16-bit FAT file system
	05 = DOS extended partition
	06 = DOS 4.0 partition, size greater than 32 MB
Ending Head	- Ending head of the partition
Ending Sector	- Ending sector of the partition
Ending Cylinder	- Ending cylinder of the partition
First Partition Sector	- Sector number of the first partition

Figure 12. Partition Table Entry Structure

Boot Sector

The Boot Sector, the first sector of the partition, is created when the disk partition is formatted. Figure 14 shows the boot sector, which consists of four fields. The first field is a three-byte JUMP instruction to the Loader program. The second field is an eight-byte vendor identification associated with the PC manufacturer. The third field contains disk-specific information called the BIOS Parameter block (BPB). The last field contains the LOADER program. This program is loaded into memory by the loader program in the Master Boot Record at the beginning of the system boot.

BIOS Parameter Block (BPB)

The BPB field is part of the Boot Sector. This field contains the following disk-specific information:

- Bytes per sector
- Sectors per cluster
- Number of FATs
- Number of root-directory entries
- Total number of sectors
- Media descriptor byte
- Number of sectors per FAT
- Sectors per track
- Number of heads
- Number of hidden sectors

During system initialization, IBMBIO reads the BPB field of every DOS partition and diskette drive. Using the BPB information, it creates a BPB table-linked list. The IBMDOS searches the linked list for a specific drive BPB table whenever it accesses that drive.

Root Directory

The disk root directory contains a 32-byte entry for each file and directory in the DOS file system root directory. Entries of other files and subdirectories in the file system are kept in other areas on the disk. Figure 15 shows the structure of each file entry. Each entry contains the basic information of files, such as file name, attribute, date and time, and starting cluster number.

An entry is created when a file or directory is created. When a file name is specified in a DOS function, IBMDOS searches the file entry on the disk. If the entry is found, IBMDOS uses the starting cluster number of the file to locate the starting point of the file cluster chain in the File Allocation Table. Using cluster numbers, DOS can locate the file sectors.

File Allocation Table (FAT)

A file is saved on the disk as one block or several blocks scattered throughout the disk. A block is a collection of contiguous sectors on the disk. On the disk, DOS maintains a table that contains information of different blocks of every file on the disk. This table is called the File Allocation Table (FAT). During file processing, IBMDOS searches the FAT if it needs to find out the physical location of file blocks. In Figure 16, each block of the file in the FAT is identified by set of chained clusters. Each cluster points to one more continuous physical sectors on the disk. The number of sectors per cluster depends on the size of the disk.

Figure 16 shows a File Allocation Table that contains two cluster chains for files FILE1 and FILE2. Note that the starting cluster of the file is in the file entry, created when the file is created. Using the starting cluster number, IBMDOS reads each cluster number of the file from the FAT. Each cluster number represents one to four contiguous sectors on the disk. The sector is then read by calling the DOS disk driver in IBMBIO. The disk driver calls the BIOS disk read function INT 13H to read the sectors from the disk.

In Figure 16, FILE1 is stored on the disk as a single block, which is identified by clusters 5, 6, 7, and 8. Note that starting cluster 5 can be found within the FILE1 entry. An FFF indicates the end of the cluster chain, and 0 indicates a free cluster. Usually a cluster number represents two or four sectors; but, for example, assume that each cluster represents one sector. FILE1 then occupies sectors 5, 6, 7, and 8 on the disk. To read FILE1, IBMDOS reads sectors 5, 6, 7, and 8. Similarly, FILE2 is composed of two blocks. The first block consists of clusters 11 and 12. The second block consists of 15, 16, 17, and 18. Thus FILE2 occupies sectors 11, 12, 15, 16, 17, and 18 on the disk.

DOS File System

The DOS file system root directory is tree-structured. Figure 17 shows this file system root directory, which consists of a disk root directory with subdirectories (special file types) and files. The disk root directory contains an entry for every file and directory in the file system root directory. Entries of other subdirectories and files can be found on other areas of the disk. Each entry is 32 bytes, and contains basic file information such as file name, file extension, file attribute, date

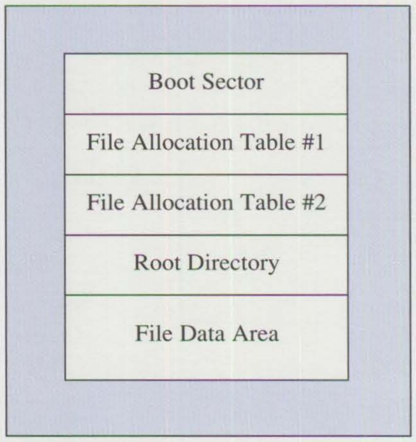


Figure 13. Partition Structure

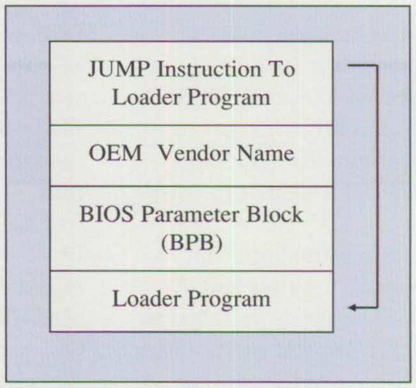

JUMP Instruction To Loader Program
OEM Vendor Name
BIOS Parameter Block (BPB)
Loader Program

Figure 14. Boot Sector Structure

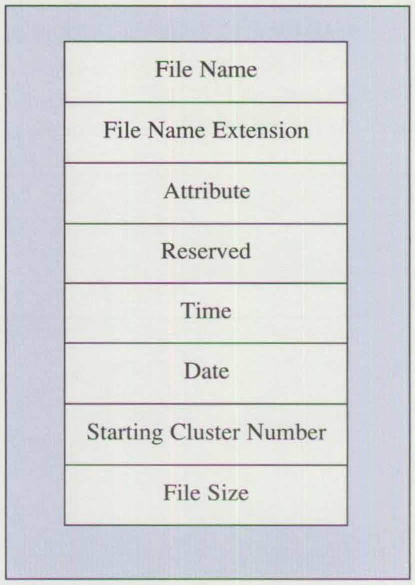


Figure 15. File Entry Structure

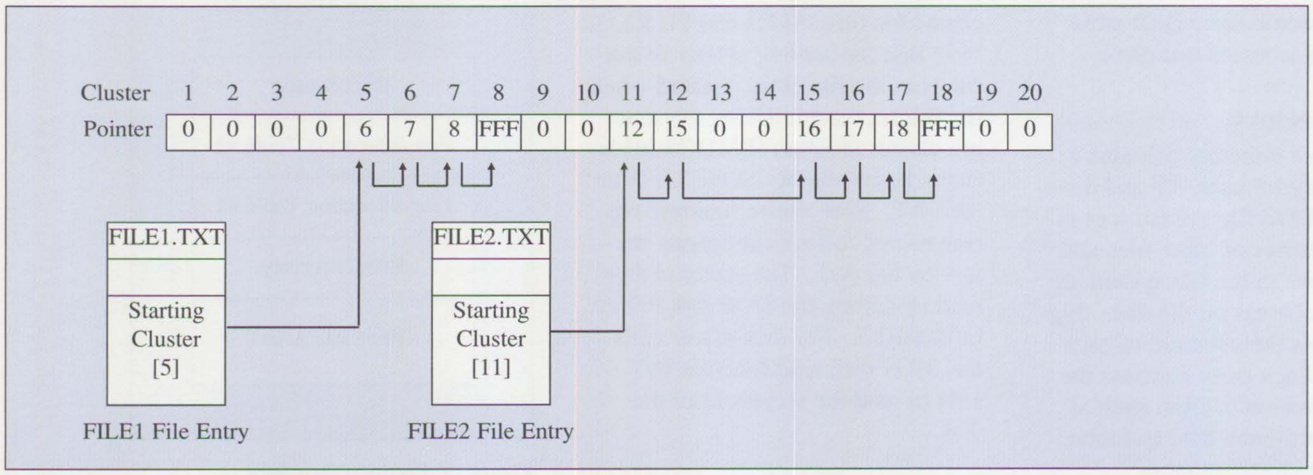

Cluster 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Pointer 0 0 0 0 6 7 8 FFF 0 0 12 15 0 0 16 17 18 FFF 0 0
FILE1.TXT
Starting Cluster [5]
FILE2.TXT
Starting Cluster [11]
FILE1 File Entry
FILE2 File Entry

Figure 16. File Allocation Table

and time of file creation, and the starting cluster of the file. A file is a collection of disk sectors. These sectors may be contiguous, in one block, or in several discontiguous blocks scattered throughout the disk.

IBMDOS uses information from FAT entries to access files and directories. A file entry is searched to find out whether the file exists. If found, IBMDOS gets the starting cluster number of the file. The starting cluster gives the entry point to the file cluster chain in the FAT. The clusters in the chain give a complete picture of the file sectors on the disk. IBMDOS uses its internal disk driver to reads these sectors.

DOS provides the FIND-FIRST and SEARCH-FIRST functions to search for a specific directory. The FIND-NEXT and SEARCH-NEXT functions are used to search for files. The CREATE, OPEN, CLOSE, READ, WRITE, and DELETE functions can be used to manipulate files. Similarly, the MKDIR, CHDIR, and RMDIR functions can be used to manipulate directories.

Useful Scenarios

Now that the functions of major DOS components have been defined, the following scenarios will examine how all these components work together:

- DOS startup and system configuration
- List a directory
- Load and execute a program

DOS Startup and System Configuration: The system startup sequence begins with the Power On Self Test (POST). The POST is part of the ROM BIOS that resides permanently in the system over 640 KB, starting at E000:0000. After the PC is powered on, the first instruction executed is a JUMP instruction at FFFF:0000. It is a *far* jump to the beginning of the POST. The POST performs the following steps:

- Exercises CPU by executing critical instructions
- Validates every byte of RAM by simple read/writes
- Exercises the system timer chip
- Exercises the system Direct Memory Access (DMA) controller chip
- Exercises the video controller and video RAM
- Verifies the keyboard for a stuck key
- Verifies the 32 KB interpreted BASIC ROM

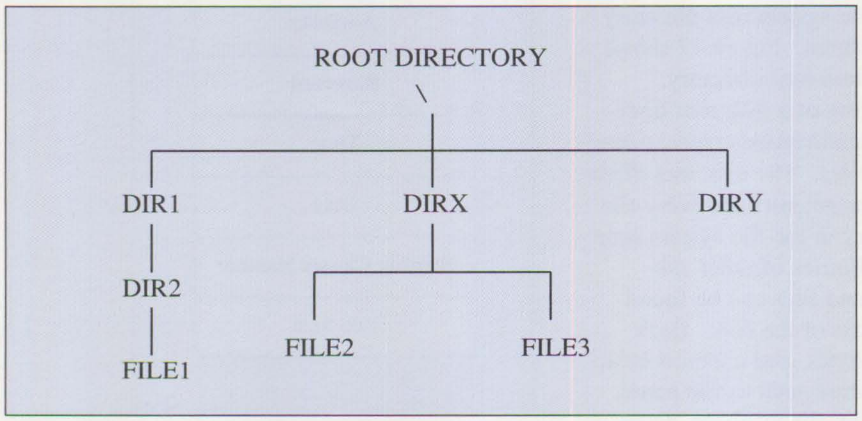


Figure 17. DOS File System Structure

An error code is displayed on the screen if any of the preceding tests fail. A successful POST is indicated by a short beep and continues to the next phase. In this phase, POST issues an INT 19H to load the DOS from the boot device. This second phase involves the following steps:

1. POST checks diskette drive A: to see if a diskette is present in the drive. If found, POST checks to see if the diskette contains a valid boot sector. If found, POST loads the loader program from the diskette boot sector. POST then goes to step 4. If no valid boot sector exists, the familiar "Non-System disk or disk error" message appears, waiting for the user to insert a proper diskette.

2. If there is no diskette in the diskette drive, POST checks the disk in drive C: for a valid master boot record.

3. If it is present, the loader program in the master boot record is loaded and executed. The loader searches the partition table for a bootable DOS partition, usually the DOS primary partition. If a bootable partition is found, it uses the information in the partition table entry to locate the boot sector. It loads the boot sector and executes the first instruction, which is a JUMP instruction to the boot sector loader program.

4. This loader program loads the first part of IBMBIO, the DOS loader. DOS loader loads the remaining parts of IBMBIO.

5. IBMBIO initializes the internal device drivers for console, keyboard, printer, and communication lines. It also creates a Block Device Structure (BDS) linked list for all disk device partitions on the system and assigns a drive letter.

6. IBMBIO then loads and initializes IBMDOS. At this point, the basic DOS functions are available for further system configuration work.

7. Using the open function, IBMBIO opens and processes the CONFIG.SYS file.

8. Using the instructions in the CONFIG.SYS, IBMBIO loads and initializes the installable device drivers based on the DEVICE= command. It sets up system buffers and file control blocks based on BUFFERS= and FCB= commands.

9. Finally, IBMBIO uses the DOS EXEC function to load and execute COMMAND.COM. The COMMAND.COM splits itself into a resident section and a transient section. The resident section is loaded next to the system buffers, and the transient section is loaded just below the 640 KB boundary. The COMMAND.COM loads and executes the AUTOEXEC.BAT file, which displays the DOS prompt and issues the DOS function 0AH to read input from the keyboard. At this point, DOS is fully loaded and ready to run.

Note that the Power On Self Test will be executed only if the system is turned on. POST will not be executed if the Ctrl-Alt-Del keys are pressed to reboot the system. Instead, BIOS issues an INT 19H, and system loading starts from the second phase.

List Directory: The List Directory command (DIR) performs the following steps:

1. Locates the current directory entry on the disk using the SEARCH-FIRST (11H) function.

2. Locates file entry of next file in the directory using the SEARCH-NEXT (12H) function.

3. Displays file entry information using the Write (40H) function.

4. Repeats steps 2 and 3 for subsequent files under the current directory.

Load and Execute a Program: If the user types a program name to be executed, the COMMAND.COM assumes the control. COMMAND.COM calls the EXEC (4BH) function to load and execute the program. The EXEC function performs the following steps:

- Reads the file entry from the disk and assigns a file handle.
- Allocates the largest free memory block to the program.
- Reads the file clusters from the FAT using the starting cluster from the file entry.
- Reads the file sectors from the disk using cluster numbers.
- Finally, transfers control to the program.

ABOUT THE AUTHOR

Pylee Lennil is a staff programmer at IBM Entry Systems Division in Boca Raton, Florida. He is presently involved in the development of PC DOS. Pylee, who joined IBM in 1983, received his B.S. in physics from Kerala University in India, and an M.S. in computer engineering from the University of Lowell in Lowell, Massachusetts.

Memory Management in a DOS Environment

Bruce G. Borkosky
IBM Corporation
Boca Raton, Florida

This article has two goals: to help the reader understand different types of memory, and to suggest several ways to use memory types in a DOS environment. An understanding of these two principles is important because different systems support different types of memory, and different applications may either require or use one or more of these memory types.

Four Types of Memory

There are four basic types of memory that can be supported by DOS: conventional, expanded memory (EMS), extended memory, and high memory. Figure 1 shows how these memory types show up in a DOS system. Figures 2 and 3 show the different types of memory and on which types of system units they are supported.

Conventional Memory: Conventional memory was the first and only type of memory for the early IBM PCs. It had a maximum size of 640 KB, which seemed more than adequate back in the days when the maximum RAM that could be put on a PC was 128 KB.

Conventional memory is supported on all systems and can be used for all application purposes: code, data, stack, and heap (temporary data storage during application processing).

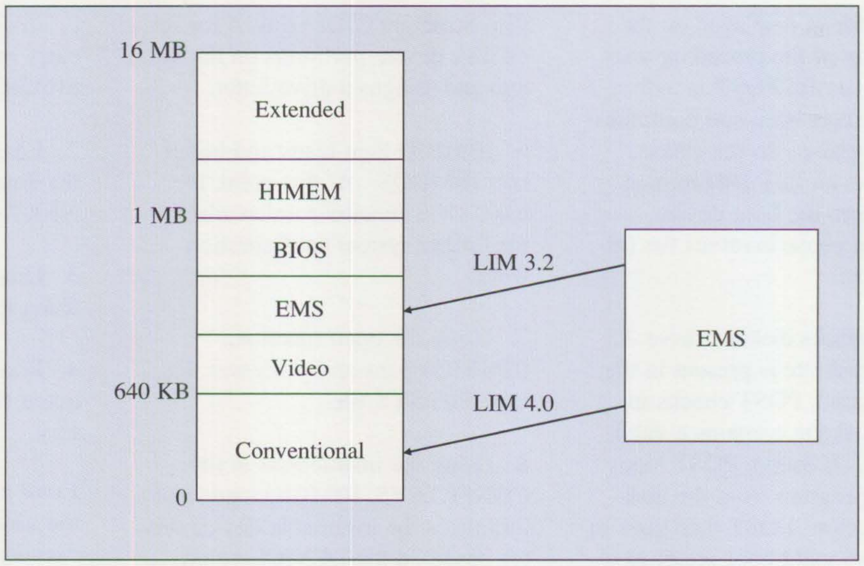


Figure 1. DOS Memory Map

All programs today use at least a portion of conventional memory, and many use only conventional memory. Many application programs have not been recoded to use the newer types of memory because some users do not have other types of memory. And conventional memory, despite its limitation of 640 KB, is still the fastest memory to use under DOS.

In fact, there are really only two reasons to use the other types of memories: either the application has outgrown 640 KB, or the application has to coexist with other large applications.

EMS Memory: The most widely used of the newer types of memory is Expanded Memory Specification (EMS). Depending on the application, EMS memory can be used to contain code, data, and heap.

EMS memory has medium access speed for applications in the DOS

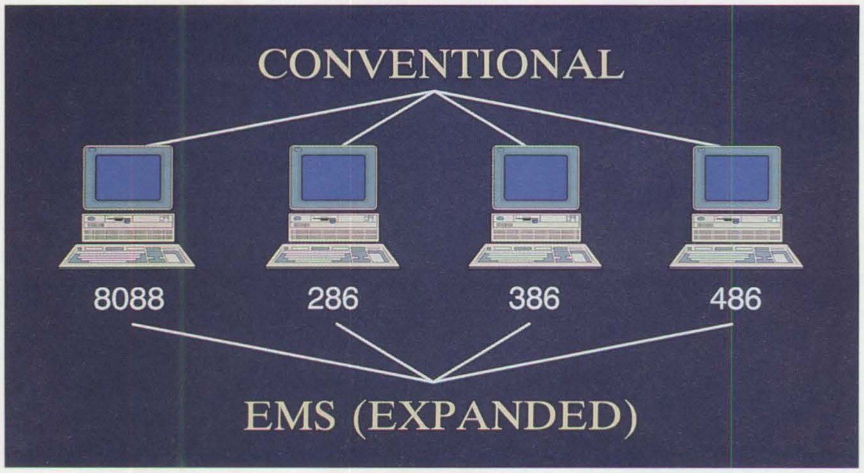


Figure 2. System Unit Support – EMS (Expanded)

environment, and all systems are capable of providing some type of EMS support. The maximum amount of EMS memory available is 32 MB; however, this 32 MB limit has not yet been implemented in most systems.

Implementing EMS on either a 286 or 386® and above system requires hardware "memory relocation" logic and an EMS driver (EMM), which interfaces between the application and the hardware remapping logic. It acts as the "BIOS" for EMS. For 386 and above systems, this hardware logic is located in the processor. For 286 systems and below, the hardware can be added through an EMS-capable memory card or a hardware device such as the ALL CHARGECARD™. It can also be added to the planar of the system unit.

To support EMS on most 286 systems, memory cards can be converted to either extended or EMS memory. These cards contain enough logic to be used either as extended or EMS memory or as some combination of both. By contrast, most 286 planar memory must remain as extended memory. On 386 and above systems, all memory – both planar and on cards – can be either extended or EMS. A 386-specific EMS device driver, which uses the processor's own logic, is used to convert extended memory to EMS.

Figure 4 shows which items are needed to have EMS on any PC system. *All* system units require at least an EMS-aware application and an EMS device driver. For 8088 and 80286 systems, a hardware card is also required.

In supporting EMS on any machine, two factors must be taken into account: the physical and logical pages available on the system. "Physical pages" refer to how large a "window" the system has, while "logical pages" refer to the total amount of memory on the EMS card or cards.

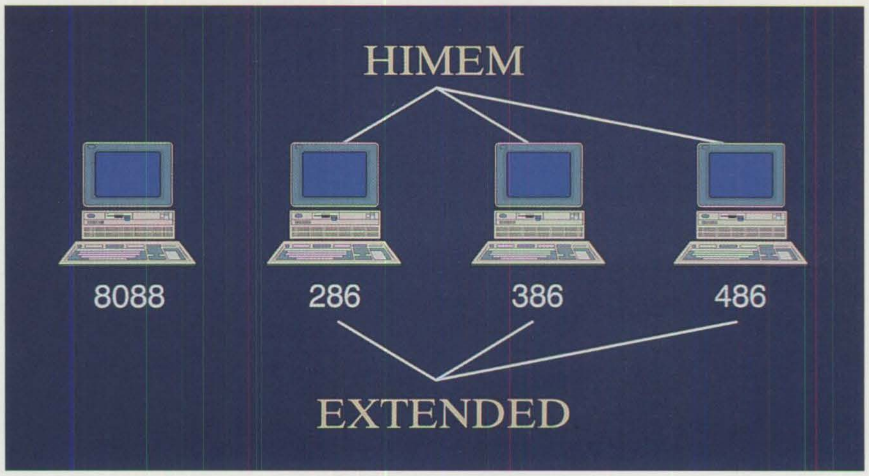


Figure 3. System Unit Support (Extended)

In order to support the Lotus-Intel-Microsoft (LIM) 3.2 specification, a machine must provide a 64 KB window that is located between 640 KB and 1 MB. To support the LIM 4.0 specification, the system, EMS card, and EMS driver must all support more than the 64 KB "window." For the LIM 4.0 specification, the system should also provide physical pages below 640 KB.

Figure 5 shows how LIM 3.2 memory is configured in a typical DOS system. Notice that there is a single application using all of the EMS memory. Also, there is only a single window, which is located at D000 hex.

Figure 6 shows how the LIM 4.0 specification has made several enhancements over LIM 3.2. In this example, there are two applications sharing the EMS memory. They

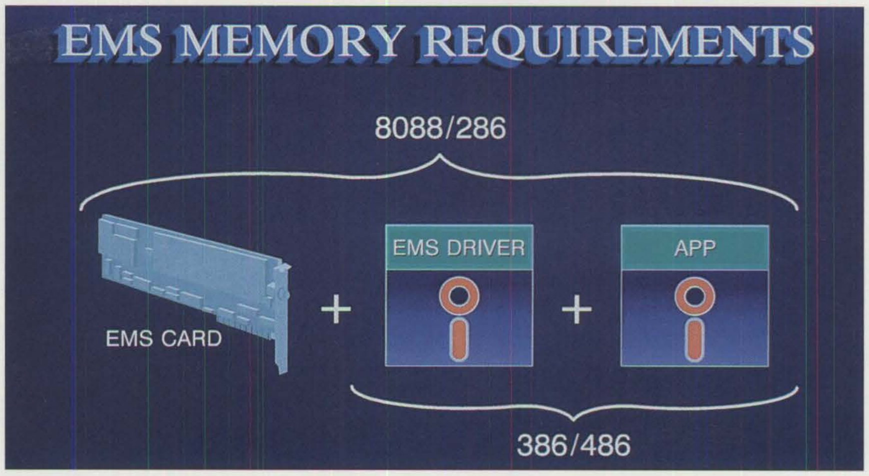


Figure 4. EMS Memory Requirements

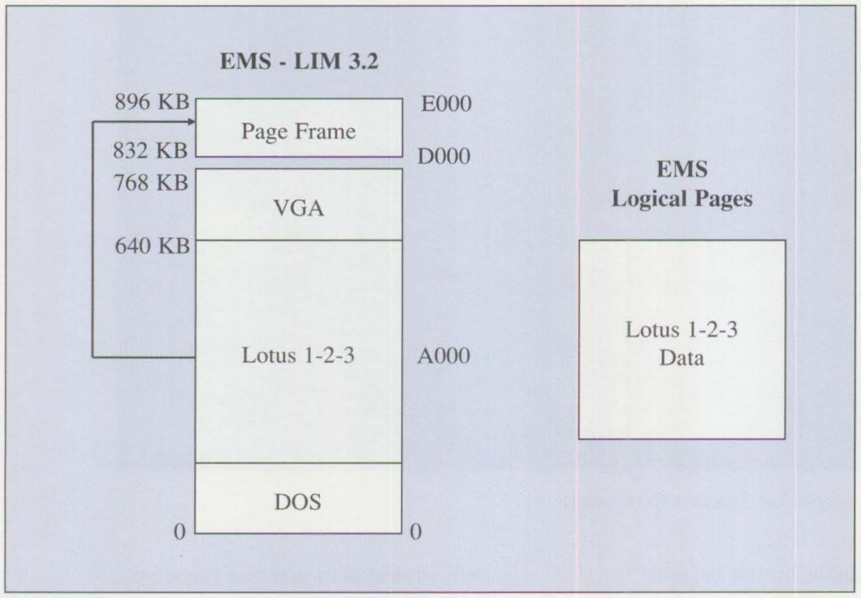


Figure 5. LIM 3.2 EMS Configuration

are using it for both data and code segments. In addition, they are both using more physical pages than the EMS page frame.

Extended Memory: Extended memory is located at 1 MB and above. As a result, it can only be supported by systems that have 286 and above processors, because 8088/8086 processors cannot address above 1 MB. Extended memory can only contain data in real mode. It is usually used for device drivers such as virtual disks, disk caches, and printer spoolers.

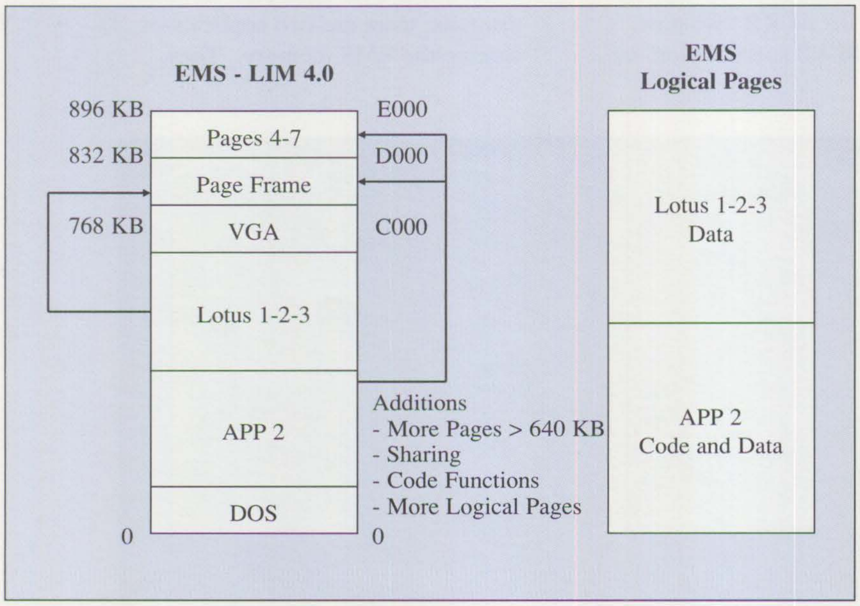


Figure 6. LIM 4.0 EMS Configuration

The maximum amount of extended memory available is limited by several factors, including address space of the processor, address space of the Direct Memory Access (DMA) chip (which allows devices to transfer data from a device such as a hard file directly to and from memory) and the physical limits of the system unit bus and the cards themselves.

The maximum amount of extended memory on most machines today is 15 MB. For 286 systems, that limit will remain at 15 MB, because the 286 processor has an address limit of 16 MB. But for 386 and 486® processors, the address limit is 4 GB; on most systems today the DMA chip still has a limit of 16 MB. As memory requirements grow, the maximum memory limit will exceed 15 MB and will then be limited only by the physical limits of card size and number of bus slots.

Figure 7 demonstrates that extended memory is used by some TSR applications to store data. In this example, VDISK and IBMCACHE are sharing extended memory. The code for these TSRs exists below 640 KB.

High Memory: Technically, this memory is just extended memory that is located at the 1 MB boundary and is almost 64 KB in size. Because of an anomaly in the 286 processor that has been replicated on subsequent processors, the first 64 KB of extended memory can be accessed from real – rather than protected – mode.

That real mode access makes high memory almost as fast as conventional memory, and, unlike true extended memory, both code and data can be placed into high memory. For DOS, high memory has the effect of increasing the real mode memory beyond the 640 KB limit.

High memory has two drawbacks, however. It is small – only 64 KB – and difficult to share. As a result, it is normally used for single applications that will reserve the entire 64 KB area for themselves.

Figure 8 displays the memory map of the high memory area. Notice how it exists within the first 64 KB of extended memory. The high memory area is normally used by a single application.

Application Types

In the DOS environment, there are four different types of memory, each with different characteristics, from size to speed to the application program interfaces (API) needed to access that type of memory. There are also different types of applications, some of which require certain memory, and each of which may be able to make use of certain types of memory.

Unaware Applications: Unaware applications were created according to the model of the original PCs and PC XTs, and as a result, are not aware of any memory type except conventional memory. These applications are usually text-oriented and may have some Color Graphics Adapter (CGA) graphic capabilities.

Memory-Aware Applications: Memory aware applications have been coded to interface with one or more of the newer types of memory. They have been modified, or in the case of new applications, created to use the appropriate API for each type of memory used.

The most common of the memory-aware applications are EMS-aware. They are usually coded to the LIM 3.2 specification, which calls for 64 KB of physical pages above 640 KB. Some common examples of these application are Lotus 1-2-3™ Version 2.X, Excel, WordPerfect™, Microsoft Word™, and Paradox®.

Some EMS-aware applications use the LIM 4.0 specification to make use of more than the LIM 3.2 page frame. An example of this type of application is Advanced Revelation Version 2.0.

Applications use EMS memory for two reasons. First, the application may be larger than available memory. Before the LIM specification was created, developers could only solve that problem by using disk

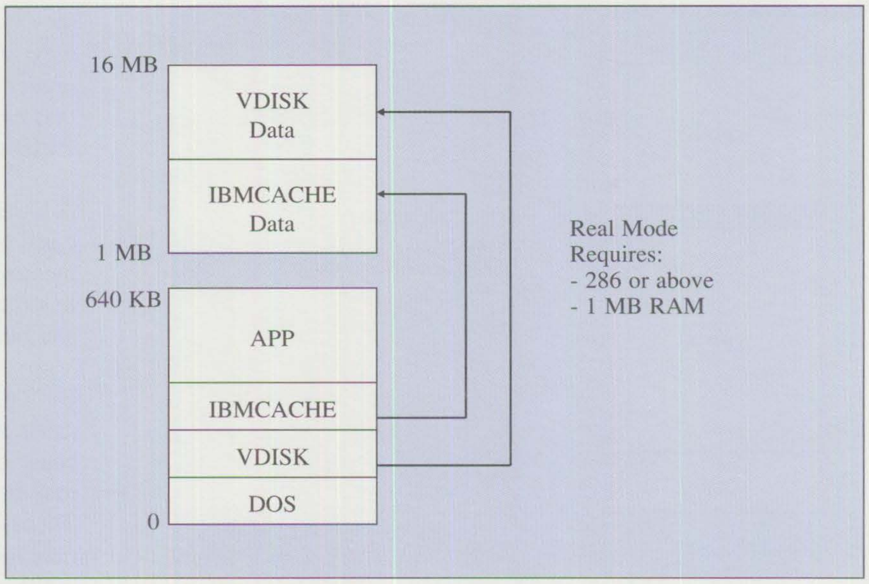


Figure 7. Extended Memory Map

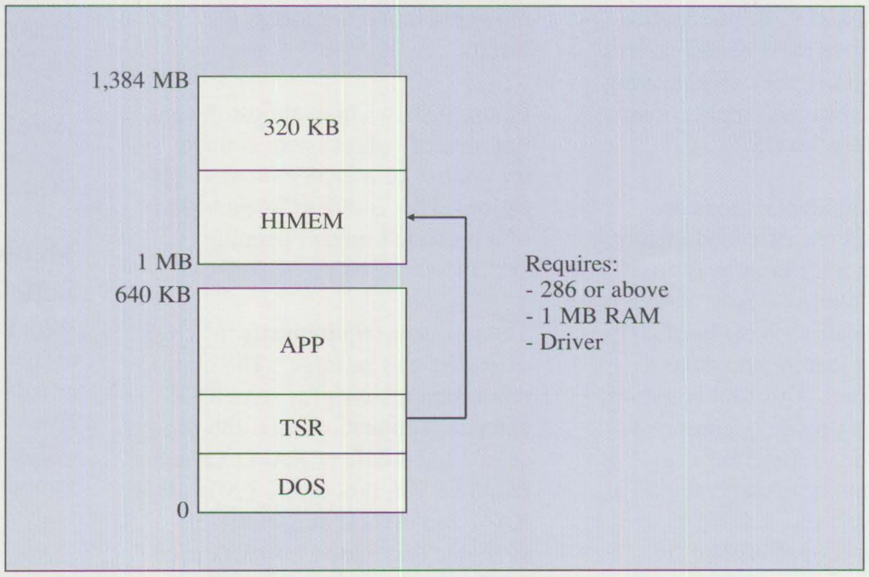


Figure 8. HIMEM Memory Map

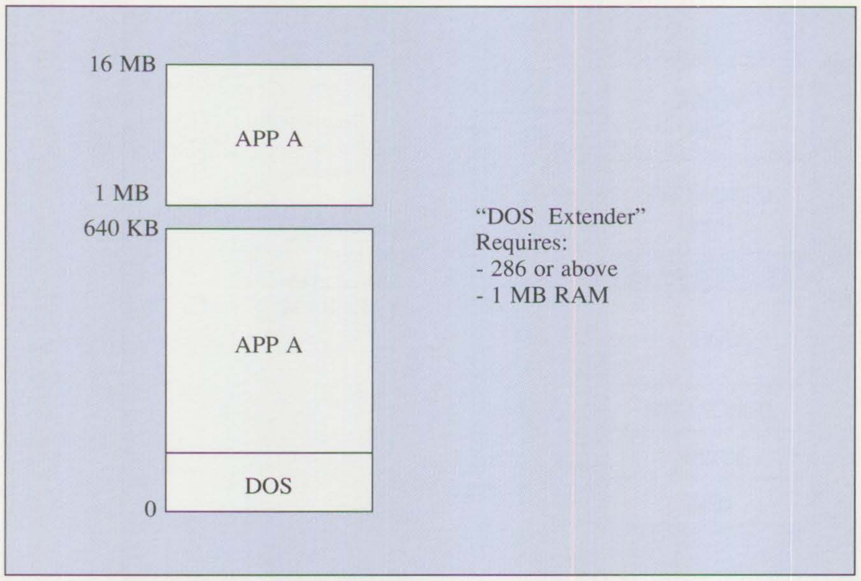


Figure 9. Extended Memory Map

overlays. But disk overlays are slow for the user and complicated for the developer. Second, EMS is also used to allow the application to keep large amounts of data in memory all the time. Otherwise, most of the data would have to be kept on disk, which slows performance.

High memory-aware applications are mostly Operating System Extensions (OS/Es), such as Windows® and DESQview™. These applications can move parts of themselves into the high memory area, leaving more room for other applications that run under the OS/E.

DOS extender applications are aware of and use extended memory, but do so in a different way from most applications. These memory-aware applications have been compiled with a feature known as a DOS extender. This tool is a minimal protected-mode operating system whose main function is to provide memory management to a single application. Examples of DOS extender applications are Lotus 1-2-3 Version 3.0, Paradox 386, IBM Interleaf Publisher, and Mathematica.

A DOS extender application loads itself into both conventional and extended memory, taking over almost the entire system memory. It normally runs in the protected mode of the processor. But when it needs to access a DOS or BIOS function, it passes back into real mode, performs the function, then returns to protected mode to process the results.

Figure 9 shows how almost the entire memory of a system is under the control of a DOS extender application. This is due to its inclusion of a protected-mode operating system.

The resource requirements for DOS extenders can be large. The minimum requirements for some DOS extender applications is a 286 processor and 2 MB of RAM. Others require a 386 processor, 3 MB of RAM and several megabytes of available disk space. If users need applications as powerful as these, they will get the best performance if they run the applications on a higher-end hardware system, one that has at least a 386 processor and 2 MB of RAM.

A DOS extender offers developers a consistent interface for all types of memory and lets the developer use that memory for code, data, stack, and heap.

But DOS extender applications can cause problems for the user, because other resident applications may be incompatible with them. These other resident applications may not follow all the protected mode rules that the DOS extender application follows, which can make the DOS extender perform poorly.

These DOS extender compatibility problems are likely to become less severe in the future. For instance, Windows 3.0 is a DOS extender that, for 386s, executes DOS applications in their own virtual 8086 box. This has the effect of increasing compatibility by isolating any programs that perform poorly in protected mode. In addition, industry standards will probably emerge in the DOS extender area. Applications may be developed that will share a single extender, thus increasing the stability of systems running DOS extenders.

Memory Management Solutions

With this background on the types of memory and applications, let us now look at memory management – how to put memory types and application types to work in the DOS environment.

Solutions for Unaware Applications: With older applications, users can access either EMS or high memory to extend capabilities of their systems. EMS solutions are compatible across all systems, while high memory solutions require at least a 286 processor.

If you are using a Local Area Network, the DOS LAN Redirector (DLR) from the OS/2 LAN Manager will place part of itself into high memory. This reduces the resident size of the DLR below 640 KB, which increases the space available for applications by about 40 KB.

If you have EMS and are using DOS 4.00, you can create the same result by placing parts of DOS itself (such as BUFFERS, FASTOPEN, and VDISK) into EMS. Corrective Service Diskette (CSD) number UR38619, dated April 1989 or later, makes DOS compatible with either LIM 3.2 or LIM 4.0 EMS cards and drivers. DOS 4.00 (dated July 1988), however, requires a LIM 4.0 card.

Users who communicate with an IBM mainframe computer through Personal Communications/3270 (PC3270) can place parts of PC3270 into either EMS or high memory. This will increase the application space from about 30 KB to 70 KB, depending on the system configuration. In addition, the latest CSD from this program should be used.

Other utility programs, such as local area networks, can also use either EMS, high memory or extended memory. Users should consult their vendors to determine whether and how they can use each memory type.

These solutions operate on systems that have unaware applications. The methods described involve placing one or more TSRs into either EMS, high memory or both. To implement these solutions, you should have at least 384 KB of extended memory (1 MB total system memory) or 512 KB of EMS memory. Even though high memory uses only 64 KB of extended memory, 384 KB is usually the minimum you can buy. On systems with 1 MB or more, the 384 KB is usually included. In those cases, all users need to implement high memory is the appropriate driver (for example, HIMEM.SYS). Similarly, 512 KB is usually the minimum amount of EMS that can be purchased, even though the application or utility will probably use less. Having the full 512 KB allows users to later add other EMS-aware applications.

There is another way to use newer memory types with unaware applications. This approach uses EMS-like capabilities, but is not a pure EMS solution. It places memory or EMS pages starting at 640 KB and backfills the memory from 640 KB to the start of the video buffer. For CGA systems, this increases the total memory from 640 KB to 736 KB, for an additional 96 KB of application workspace.

This solution requires several items that must all work together in order to implement it. First, the unaware applications must not use higher graphics modes of the EGA or VGA graphics adapters. If you currently have either a monochrome or CGA card, it is acceptable. If you have an EGA or VGA card, you may still be okay if your application uses only the CGA-mode graphics.

Another required item is an EMS card and driver. On 386 systems, you will need extended memory and a 386 EMS driver. You can also use extended memory on a 286 system if you use a hardware device such as the ALL CHARGECARD.

The EMS drivers you use must also be capable of extending the DOS box past the 640 KB limit. If it is, you will have an additional 64 KB to 96 KB of space for your applications.

This solution uses an aspect of the PC architecture that slightly bends the rules. As originally designed, applications were limited to 640 KB, while the rest of the 384 KB address space was "reserved."

However, for monochrome and CGA systems, much of this reserved space went unused. Fortunately, some of this unused space can be reclaimed.

An example of how this solution is implemented is shown in Figure 10. This user has a CGA system, and is using the ALL CHARGECARD. The DEVICE= statement was placed in the CONFIG.SYS file, and the size of the DOS box is enlarged to 736 KB.

Figure 11 provides a detailed look at the reserved area of the PC architecture. Monochrome and CGA video buffers are smaller, then, and fit inside of the VGA buffers.

Solutions for EMS-Aware Applications: Using EMS-aware applications may require either upgrading software or purchasing applications that are aware of EMS. In some cases, this is as simple as buying a new version for an existing application program. In other cases, the applications may support EMS, but the system on which they are running may not have EMS.

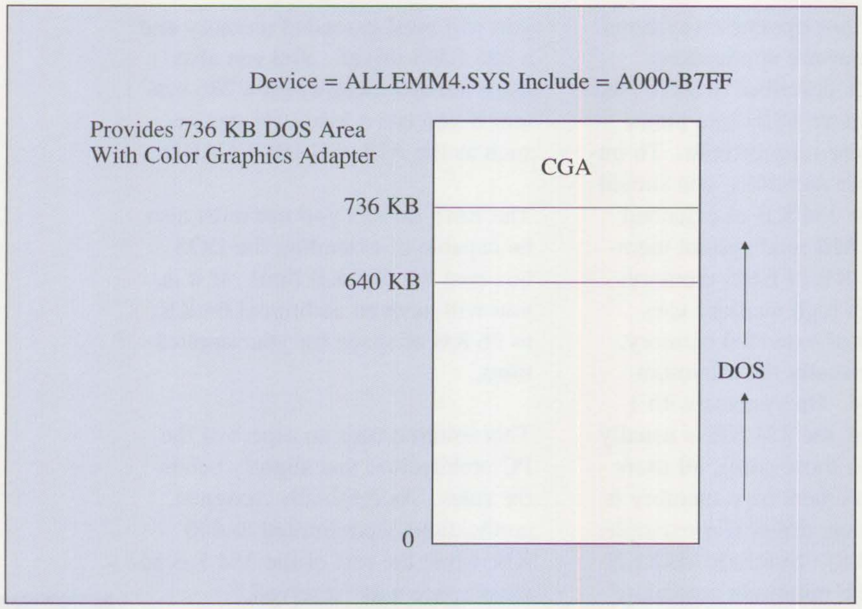

Device = ALLEMM4.SYS Include = A000-B7FF
Provides 736 KB DOS Area
With Color Graphics Adapter
CGA
736 KB
640 KB
DOS
0

Figure 10. Solution for Unaware Applications

Then the solution is to purchase an EMS card and driver.

In general, this requires at least 512 KB of EMS memory. Because one of the assumptions of EMS memory is that it will be shared, it is best to buy at least 1 MB of EMS memory so that adding EMS applications or increasing the size of the data does not present problems.

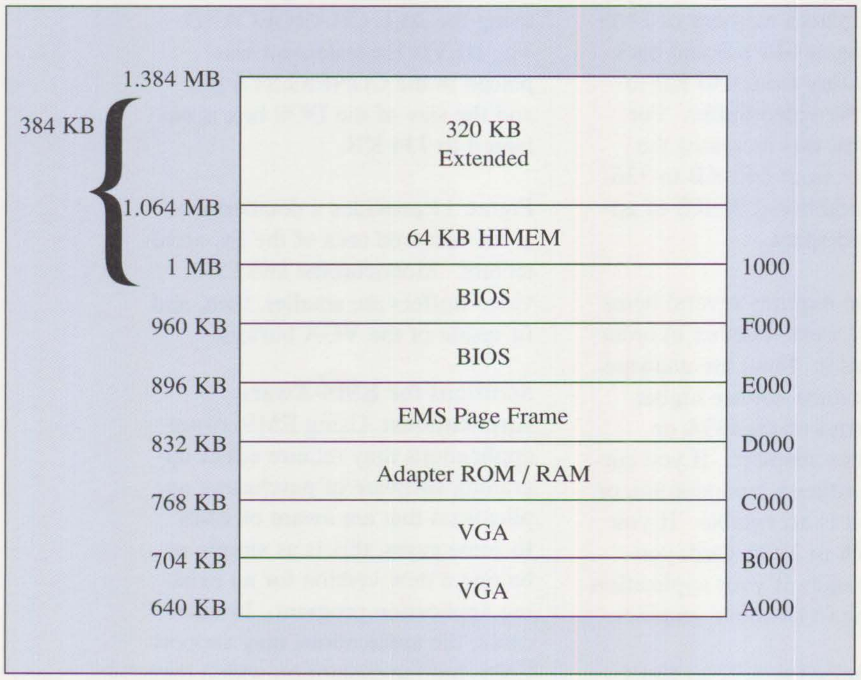


Figure 11. PC Memory Map Reserved Area

In all cases, users should ask their vendors whether an application uses EMS and, if so, how it uses it and what its specific requirements are. Because the LIM specification is vague, some EMS implementations may not meet an application's requirements. Some of the questions that should be asked of a vendor are:

- How many 16 KB physical pages are required?
- Does the application use only the LIM 3.2 page frame, or does it require more?
- Can it work with just a single physical page?
- How much EMS memory (logical pages) does it require?
- What are the minimum and maximum amounts of memory it can use?

Solutions Using Operating System Extensions (OS/E): OS/E applications manage memory for multiple applications. They allow one or more applications to be loaded into memory at the same time. Depending on the design of the OS/E, they may or may not provide multitasking, a graphics interface (GUI) and a programming interface. Some examples of OS/E applications are Windows 2.11, Windows 3.0, DESQview and Software Carousel™.

OS/Es provide some OS/2 benefits at a lower cost than OS/2. These applications also may be used on machines that cannot run OS/2. As a result, OS/Es allow users to retain their investments in current machines, while giving a common interface across all machines at the site.

As older machines are gradually replaced with OS/2-capable machines, users can make the transition quickly and easily, since they will have already begun working in an OS/2-like environment.

Requirements for this memory management solution vary, depending on the OS/E and the application. Some applications perform adequately on 8088 machines. But graphics-intensive OS/Es and applications need at least a 286 processor and from 1 MB to 2 MB of RAM. The total amount of RAM needed depends on the applications. The larger and more powerful the application, the more memory it requires. In general, users can assume from 512 KB to 1 MB of memory per application. Therefore, to have three applications in memory at once, a system needs from 2 MB to 4 MB of RAM.

For most OS/Es, the primary memory interface is EMS because it is large, reasonably fast to access, can be shared, and supports multiple applications. OS/Es using EMS require full LIM 4.0 services: pages below 640 KB, pages anywhere in the 1 MB address space, aliasing and at least one fast register set for every program that is to run simultaneously.

Figure 12 demonstrates how an operating system extension might use memory. In this example, four applications are loaded into memory, but only one – Excel – is currently active. The other three applications are not running. However, the OS/E has loaded the entire application – code, data, stack, and heap – into EMS.

By contrast, Windows 3.0 uses extended memory by placing the processor into protected mode. This may prove to be a better solution than EMS for 386 and above machines because 386s and 486s have features designed to support multiple DOS applications.

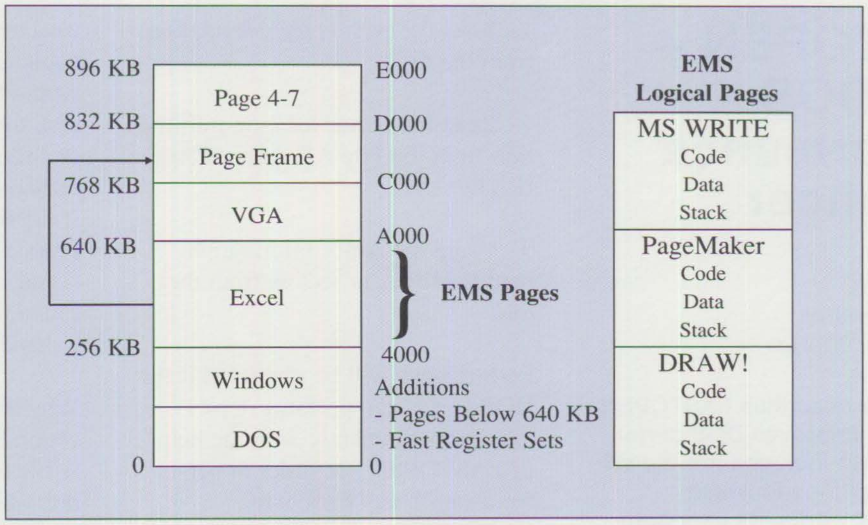


Figure 12. EMS LIM 4.0 Operating System Extensions

Conclusion

OS/2 is – and will be – the best solution for using different types of memory. In OS/2, all memory is really extended memory because all memory is treated the same by all applications. Whether an application is running above or below 1 MB, it runs the same way.

In DOS, as was described here, things are much different. For instance, under DOS, extended memory can only be accessed in protected mode, and the system then must be reset to real mode. This access is time-consuming, slowing the application's performance under DOS.

But the primary consideration for any user is the application. If a required application runs under OS/2, users should determine whether that environment might be a better choice than DOS. Because the standard OS/2 platform of the 1990s will be a 386 processor with 4 MB of RAM and a 60 MB hard file, users who choose OS/2 ought to consider that the minimum hardware investment.

ABOUT THE AUTHOR

Bruce Borkosky is a senior associate programmer at IBM's Entry Systems Division in Boca Raton, Florida. He received a B.A. in music from Ohio Wesleyan University, an M.S. in computer science from the University of Dayton, and is a 1991 Ph.D. candidate from Miami University. After joining IBM in 1984, Bruce's responsibilities have included developing PC DOS and working as an advisor for new memory card development.

FASTOPEN – The DOS Performance Enhancer

Pylee Lennil
IBM Corporation
Boca Raton, Florida

This article describes FASTOPEN and how it improves DOS performance. Also included are the different conditions in which FASTOPEN works, its user commands, and a comparison between FASTOPEN and system buffers.

FASTOPEN is one of the least-understood DOS components. The primary reason is that FASTOPEN functions are closely associated with the internal operations of the DOS kernel, an area quite alien to most people. Basically, FASTOPEN is an extension of the DOS kernel. It can improve DOS performance by helping DOS access directories and files more efficiently.

FASTOPEN is a stay-resident program. When loaded, it becomes a part of the DOS kernel. The kernel uses the FASTOPEN features to cache the file and directory path information as well as the location of the sectors of most recently accessed files. This information can be reused during subsequent access of the same file and directory, which eliminates further disk reading, resulting in an improvement in DOS performance.

To learn how FASTOPEN enhances performance, it's important to know that DOS uses the following steps to access files:

1. Read file path entry information from the disk

2. Read file sector location information from the File Allocation Table (FAT)

3. Using the sector information, read the file data sectors from the disk

Performance will be improved if the DOS kernel can perform steps 1 and 2 less frequently with the help of FASTOPEN. In order to perform step 1 less frequently, FASTOPEN caches the path information of recently accessed files and directories. Similarly, step 2 can be performed less frequently by caching the file sector location information of recently opened files in a cache buffer. Both path and sector information in the cache buffer can be reused for subsequent access of the same path and file, resulting in improved performance.

Why does DOS need path information from the disk? Usually a file path consists of or one more subdirectories plus a file name. If an application calls a DOS function with a file path, DOS searches the disk for entries of each directory and file in the path. The primary reason for this search is to find out if the file exists on the disk. If not found, DOS displays the "File not found" message. If the file entry is found, it uses the entry information to handle the file operations.

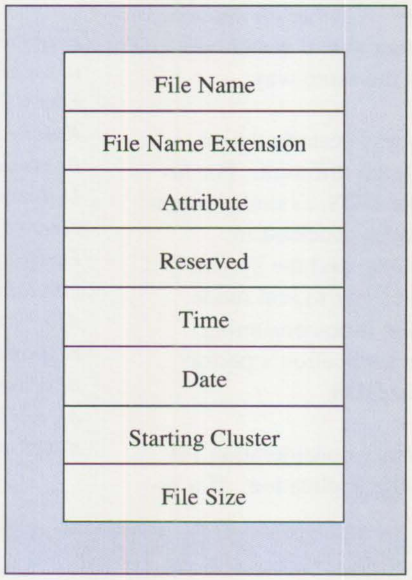


Figure 1. File Entry Structure

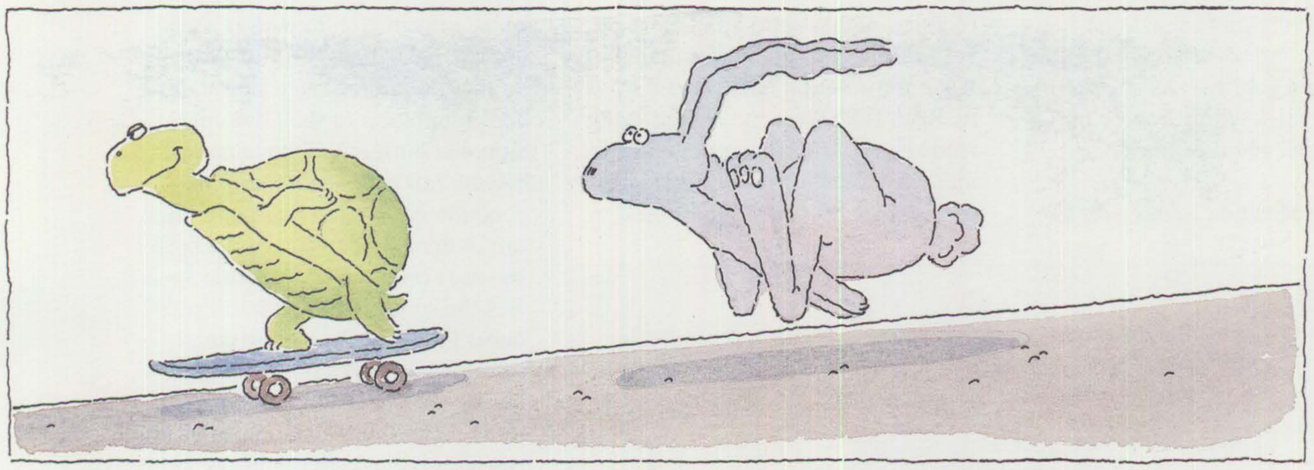
Why does DOS need file sector location information from the FAT? A file usually consists of a collection of discontiguous blocks. Each block consists of a collection of contiguous sectors. The file blocks are scattered throughout the the disk. To access a file, DOS must find the location of file sectors from the FAT. Once the sectors are located, DOS reads the file sectors by calling the disk driver.

File and Directory Entries

DOS maintains an entry associated with every file and directory under the file system. These entries are stored on the disk. Because a directory is a special type of file, the entry of a file and directory has the same structure. The file structure is shown in Figure 1. Each entry has 32 bytes and contains information such as name, attribute, date and time, and starting cluster number. When DOS accesses a file or directory, it needs the entry information of each directory and file in the path. Once the path entries are found, they can be stored in the FASTOPEN path caching buffers for future use.

File Allocation Table (FAT)

DOS needs some mechanism to locate file blocks during file access. For this, DOS maintains a table in a special area of the disk called the File Allocation Table (FAT). The



basic structure of a File Allocation Table is shown in Figure 2.

The FAT contains cluster chains of every file on the disk. The cluster chain contains groups of contiguous clusters for every block in the file. The beginning of a file cluster chain is identified by a starting cluster. The starting cluster is recorded in the file entry. As shown in Figure 2, FILE1 consists of one block and this block is identified by clusters 5, 6, 7, and 8. Note that starting cluster 5 is recorded in the file entry.

To access a file, DOS reads the starting cluster from file entry. Using the starting cluster number, it locates the beginning of the file cluster chain in the FAT. DOS then reads the rest of the file clusters. Using the file clusters, the corresponding file sectors are read from the disk by calling the disk driver.

Caching Path Information

FASTOPEN maintains a cache buffer for saving the information of most recently accessed paths. This cache memory is a collection of fixed-size buffers. The number of buffers can be specified by **n** in the FASTOPEN command:

FASTOPEN D: (n,m)

Here **n** is the number of buffers allocated in the cache area. Each buffer can hold either one file or directory entry information. Each buffer contains the basic file information such as file name, attribute, date and time of file, and starting cluster of the file. If a path consists

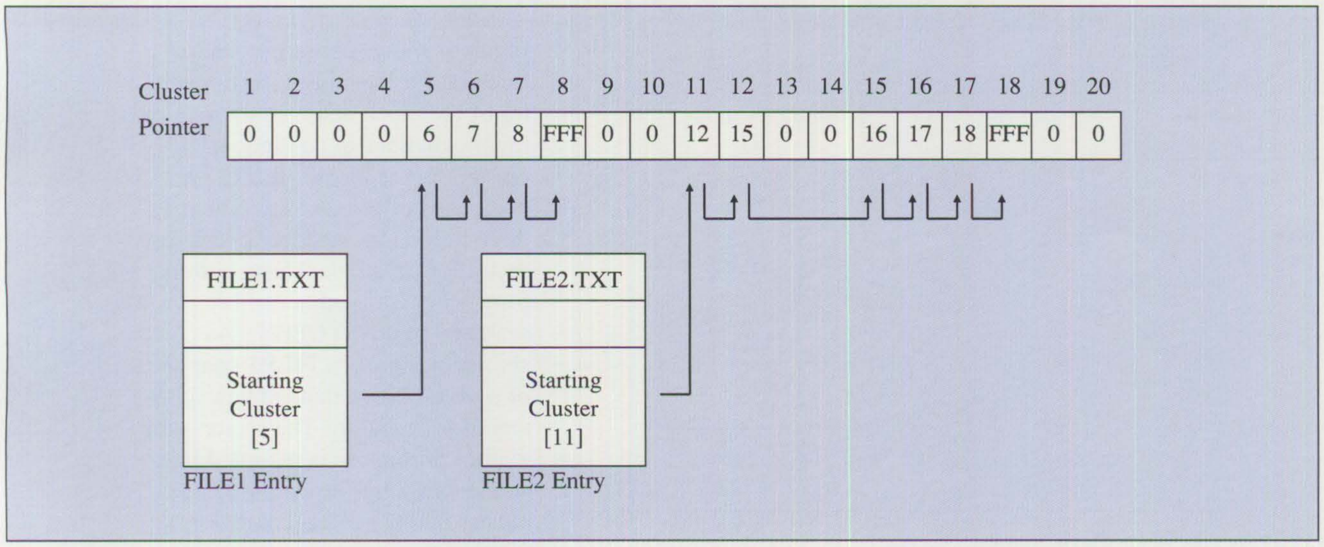


Figure 2. File Allocation Table Structure

of two directory names and one file name, three buffers are needed to save the three entries. As shown in Figure 3, FILE1 requires three buffers and FILE2 requires only one.

When an application issues a DOS function with a path, DOS calls the FASTOPEN for path information. If the information is not found in the FASTOPEN cache buffer, DOS reads entry information of each element in the path from the disk. As each entry is found, it gives the entry information to FASTOPEN for caching. FASTOPEN saves information in the linked buffers. At the completion of the DOS function, the buffers will have entries for the full path specified in the function call. Now that the entries are in the FASTOPEN cache buffer, the next time a DOS function specifies the same path, DOS will request FASTOPEN for the path information. FASTOPEN looks through the cache buffers and returns the path information to DOS. Because the information is available from the FASTOPEN cache buffer, additional disk reads are eliminated, which results in improved DOS performance.

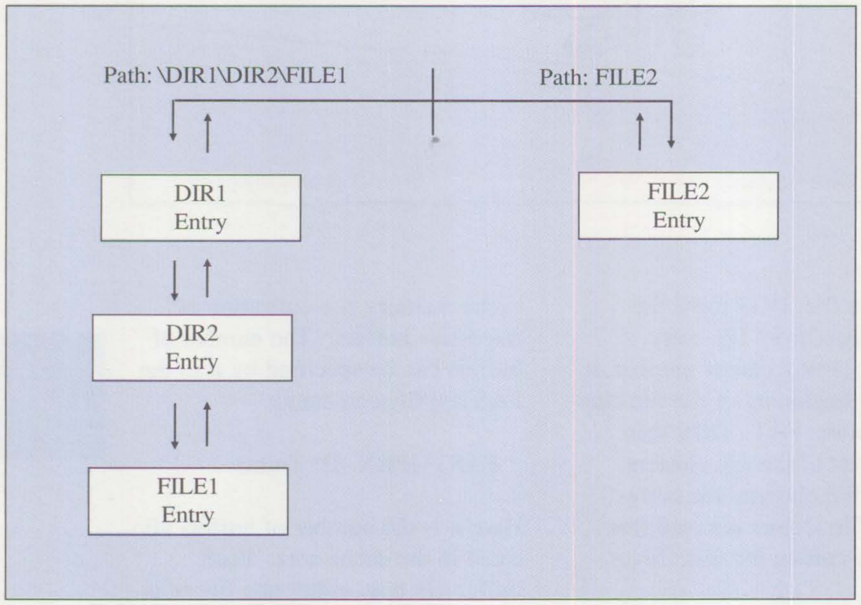


Figure 3. FASTOPEN Disk Directory Information Cache Buffers

Caching File Sector Location

FASTOPEN maintains a cache memory area for caching file sector information from the FAT. The cache memory is composed of a collection of buffers. Each buffer has 16 bytes and holds the cluster information of a file block. The number of buffers can be specified using the "**m**" parameter in the FASTOPEN command:

FASTOPEN D: (n,m)

Figure 4 shows the buffer setup. Each buffer contains a starting cluster of the block and a count indicating the number of contiguous clusters in the block. If a file is composed of three blocks, then three buffers are needed to save its sector information. These buffers form a linked list that can be searched by FASTOPEN. As shown in Figure 4, FILE1 consists of a single block that contains clusters 5, 6, 7, and 8. The sector information of this file is recorded in a single cache buffer with starting cluster number 5 and count 4.

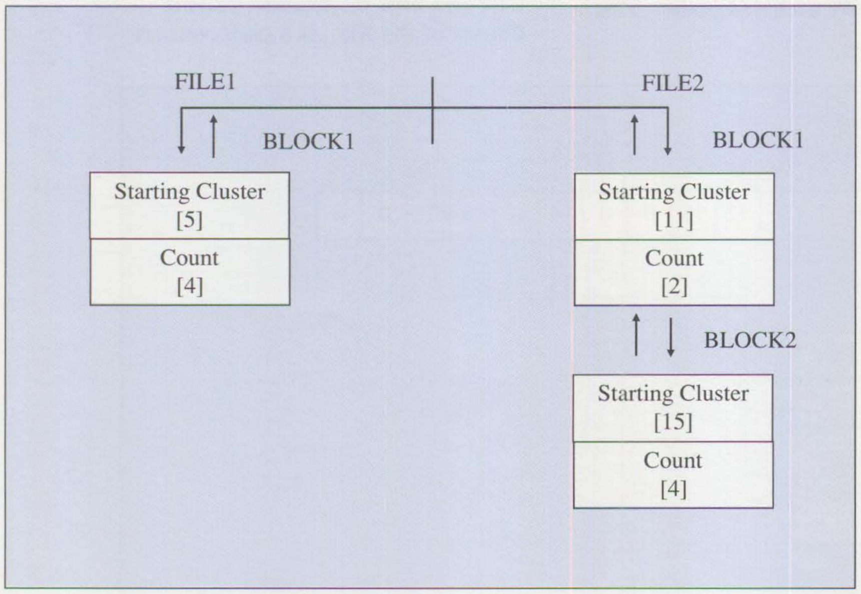


Figure 4. FASTOPEN File Sector Cache Buffers

To access FILE1, DOS needs to know the FILE1 sector location information. DOS calls FASTOPEN for this information. In Figure 4, FASTOPEN looks through the buffer chain and locates the single buffer associated with FILE1. It reads the starting cluster 5 and computes the remaining clusters 6, 7, and 8, using the count 4. The FILE1 cluster information is then sent to DOS.

FASTOPEN Performance Considerations

Performance improvement by FASTOPEN strictly depends on the behavior of the application. Caching the path information will improve the performance of an application that does constant directory and file access. Caching the file sector location will improve the performance of an application that does continuous random disk access on large, less fragmented files.

FASTOPEN Versus System Buffers

What is the advantage of FASTOPEN over system buffers? System buffers cache all sectors read during disk read operations.

FASTOPEN can provide better performance that may not be available from system buffers.

These sectors include sectors associated with the file entries and FAT information. But it is unlikely that these sectors will stay long enough in the system buffers for future use, because these sectors will be replaced by sectors from subsequent disk reads. If FASTOPEN is used, it is guaranteed that the file entry and FAT information will stay in the cache buffer unless the least recently used information is replaced by the recent ones because the buffer is full. Usually with sufficient cache buffers and less fragmented files, the information stays until the file is deleted or the path is changed. FASTOPEN can therefore provide better performance that may not be available from system buffers.

ABOUT THE AUTHOR

Pylee Lennil is a staff programmer at IBM Entry Systems Division in Boca Raton, Florida. He is presently involved in the development of PC DOS. Pylee joined IBM in 1983, and received his B.S. in physics from Kerala University in India and an M.S. in computer engineering from the University of Lowell, in Lowell, Massachusetts.

DOS 4.00 Compatibility Issues

Bruce G. Borkosky
IBM Corporation
Boca Raton, Florida

This article investigates the confusion about the so-called incompatibility of DOS 4.00 with some of its application programs. The causes of these alleged incompatibilities will be discussed here, and solutions given when incompatibilities do exist.

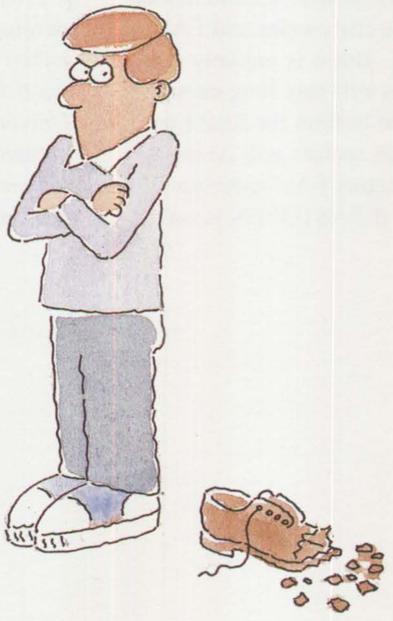
All versions of software are, to some extent, incompatible with previous versions of the same software. Often, there are better ways of doing things that are different from the older ways. For example, sometimes fewer key strokes are required to perform the same function. This new way is better than the old, but it is still "incompatible."

Some of the previous DOS versions were less compatible with older applications than DOS 4.00. However, the incompatibilities were tolerated because these DOS versions were announced simultaneously with new personal computers. The changes to DOS were expected and dealt with because the operating system took advantage of the new hardware features.

PS/2 Hardware Enhancements

Changes made to DOS 4.00 take advantage of some new features in IBM PS/2® systems. These enhancements had not been made in DOS 3.30. DOS 4.00 supports these new features, which include large hard file support, expanded memory (EMS), enhanced keyboard, and video graphic array (VGA) graphics. These enhancements are the cause of most of the reported incompatibilities.

One of these hardware enhancements was actually released in DOS 3.30. The IBM BIOS has always had the capability for multi-track I/O to the hard file. In DOS 3.30 and 4.00, DOS uses this feature to improve performance. This may cause a problem for some users who have purchased "IBM-compatible" machines. Some of these machines do *not* support multi-track I/O.



If you have an IBM-compatible machine, are upgrading DOS from version 3.21 or below, and are experiencing problems, either the system or hard file manufacturer should be contacted. The manufacturer may have developed a new BIOS that supports multi-track I/O.

MODE now supports display screens of greater than 25 lines. Some applications may have difficulty with these large number of lines and may not work properly. If this is the case, simply set the lines per screen back to 25 before running the application, and set it back when the application is complete.

Use of Undocumented Interfaces

Another reason for incompatibility is in the application programs themselves. Almost all applications are compatible with DOS 4.00 because they are coded to the standard DOS interfaces. Some applications are not compatible because these standard interfaces are not used. Undocumented disk or memory system areas may be used. These areas are undocumented because they may frequently change, as they did for DOS 4.00.

DOS Version Change

Other applications check for a specific DOS version number. These applications may actually be compatible with DOS 4.00. However, they specifically check for a version level of 3.30. When they do so, they refuse to run on DOS versions greater than 3.30. Therefore, the application is perceived by the user to be incompatible with DOS 4.00.

There are two types of applications that will not load in the DOS 4.00 environment. There are those that use undocumented areas of DOS, and those that check for a specific version. If you have one of these applications, contact the vendor to determine if those applications have been updated for DOS 4.00. Most application vendors have updated their code to be compatible with DOS 4.00.

Backwards Compatibility Designed Into DOS 4.00

Some applications were written for previous versions of DOS. There are features in DOS 4.00 that make it more compatible with those applications. Likewise, there are features in DOS 4.00 that make the user interface more compatible with previous DOS versions. The following DOS commands contain features that increase compatibility:

FDISK: Some applications cannot handle logical drives larger than 32 MB. These are usually utility programs that access system areas of the disk. Most of these utility applications have been updated for DOS 4.00. Most of the utilities will work correctly if the logical drives are defined as less than 32 MB.

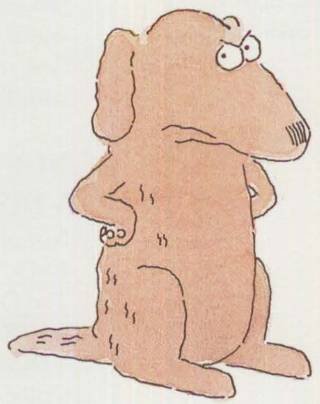
SHARE: Some applications do not run correctly when SHARE is loaded. Because SHARE must be loaded with logical drives larger than 32 MB, keeping the logical drives below 32 MB will also solve this problem.

Most applications that have problems with SHARE are single-user versions of database programs. The applications that seem to have a problem with SHARE do not handle file accesses in a standard, error-recoverable manner. Usually, multi-user versions of the database management systems (DBMSs), which can run on network servers, can correctly work with SHARE.

DOSSHELL: Many Terminate-and-Stay-Resident (TSR) programs cannot handle the graphics mode of the DOSSHELL. These programs usually pop up onto the screen on top of whatever application is running at the time. The programs may run correctly, but do not restore the screen properly upon returning to the application.

An example of this type of program is a 3270 emulation program, which allows the personal computer to act as a terminal to a 370-type mainframe. When the user presses the "hot key" to access the emulator, the program takes control of keyboard and screen accesses, and runs the emulator program. But when the user presses the "hot key" to go back, the display is garbled.

The DOSSHELL uses VGA graphics in PS/2 systems. The problem encountered when using TSR programs affects any application that places the display into graphics mode. The actual problem lies in the TSR program and not the application running when the "hot key" is pressed.



If you have a TSR program with this problem, there are three alternatives. First, obtain an update of the TSR program from your vendor. Determine whether the application is compatible with DOS 4.00, and which version number is compatible. Second, start the DOSSHELL in text mode by using the /TEXT switch on the command line while starting the DOSSHELL (for example, DOSSHELL /TEXT).

Third, the DOSSHELL can still be used in graphics mode with your TSR program. Use the COMMAND prompt to temporarily exit from the DOSSHELL to the DOS command line. This command will load another copy of COMMAND.COM and will place the display into text mode. Another way to get to the command prompt is by pressing the Shift and F9 keys.

From this "shell", you can then "hot key" to your TSR, and then return to it after completing your task. Then type "exit" on the command line to return to the DOSSHELL.

SWITCHES: Some applications cannot handle the key codes from the new keys on the PS/2 enhanced keyboard, which is now standard on most PS/2 systems. These new keys include the F11 and F12 keys, which produce extended key codes. If the command SWITCHES=/K is placed in the CONFIG.SYS file, these new, extended keys will not be used by the system.

Applications that have a problem with the new keys are often TSR programs that intercept all key code values from the system, such as keyboard translation programs.

XMAEM: Some applications use a "DOS extender." These extenders allow the application to use portions of extended memory; XMAEM implements EMS for 386 systems by using the same extended memory. Both DOS extenders and XMAEM use extended memory to place the system into protected mode; therefore, they both cannot be active at the same time.

Currently, XMAEM is incompatible with these DOS extenders, but compatibility is planned. To use both, the CONFIG.SYS file must be changed and XMAEM must be removed before using the application. Then reboot the system without running XMAEM, and the application can be loaded.

Also, XMAEM is not currently compatible with HIMEM.SYS. One of the main features of HIMEM is to make the 64 KB area available, beginning at 1 MB. XMAEM was designed to use this area itself, and it is not compatible with any other application that uses the same memory location.

BUFFERS, XMA2EMS: Some applications cannot handle the additional physical EMS (see note) pages that XMA2EMS can allocate (pages P254 and P255). If you have such an application, do not place the P254= and/or P255= parameters on XMA2EMS. You should be able to use the FRAME= parameter without problem.

Likewise, do not use the /XS parameter on BUFFERS. The /XS parameter on BUFFERS tells DOS to use pages P254 and P255; instead, use the /X parameter, which tells DOS to use the EMS page frame (pages 0-3).

Note: For more details about EMS, refer to the article entitled "Memory Management in a DOS Environment," which begins on page 12.

XMA2EMS / XMAEM: The implementation and use of EMS in DOS 4.00 has caused some confusion among some users. This confusion has created the impression with some of those users that DOS was incompatible with EMS and the LIM specification. The issues involved will be explored in order to end the confusion.

First, the IBM device driver XMA2EMS only works with, and was tested for, IBM (and fully compatible) memory cards. This is the normal and expected operating procedure for EMS cards, and those device drivers work only with that vendor's EMS hardware. Because IBM did not originally provide the drivers for its EMS-capable memory cards, it was decided that the best vehicle was with the IBM version of DOS. All IBM EMS-capable cards are also compatible with the LIM 4.0 and 3.2 specifications.

XMAEM is less IBM-hardware specific, although some users may experience difficulty. XMAEM runs on systems with 80386® (or above) processors; it uses any extended memory that is physically placed in the system unit, up to 16 MB. In conjunction with XMA2EMS, it converts this extended memory to EMS memory. It checks the system model byte (in the system ROM BIOS) for a model byte. On IBM systems, this model byte indicates that the system has a 386 or above processor. Other systems and accelerator cards may have a PC AT-type model byte for compatibility with PC AT® hardware.

BUFFERS, FASTOPEN and VDISK

Not only does DOS 4.00 *implement* EMS, but parts of it also *use* EMS. The DOS components that use EMS are BUFFERS, FASTOPEN, and VDISK. These components, as originally designed, make use of the LIM 4.0 features that were built into IBM EMS cards and the IBM device drivers.

However, this caused a problem for non-IBM EMS cards. Even though these non-IBM cards had been updated via software to the LIM 4.0 specification, the hardware was designed for the LIM 3.2 specification, and some features could not be implemented via software alone. These features relate directly to the page frame or number of physical pages on the card.

The LIM 3.2 specification allows for four physical pages of 16 KB each, which makes a total of a 64 KB window into EMS memory. This is called the EMS page frame. The LIM 4.0 specification allows physical pages to be placed anywhere in the 1 MB address space, thus allowing a theoretical maximum of 64 physical pages.

BUFFERS, FASTOPEN, and VDISK use these LIM 4.0 pages to access EMS memory. By going outside the LIM 3.2 page frame, these components can provide faster and more reliable performance than by sharing the page frame with other applications.

Because the DOS components used the LIM 4.0 features that these other cards could not provide, it was considered an incompatibility. These components were not designed to exclude these other cards, but rather to take unique advantage of the IBM EMS card capabilities. IBM's cards can provide more than the LIM 3.2 EMS page frame; these additional pages are specified in the LIM 4.0 specification.

Enhanced EMS Usage

As a result of user requests, these components can now *share* the LIM 3.2 EMS page frame with other applications. This will allow these components to operate with these non-IBM EMS cards. It is important to remember that operating in this mode has some limitations, and is somewhat slower than the original mode.

For example, if /X or /XD is specified in the BUFFERS command, BUFFERS will use the EMS page frame instead of using page 255.

FASTOPEN and VDISK can also use the EMS page frame. To implement this option, do the following: First, place an /X parameter on FASTOPEN and/or VDISK. Then, specify on the BUFFERS command either /X, /XD, or no parameter. In other words, do *not* specify an /XS parameter on BUFFERS.

BUFFERS, FASTOPEN, and VDISK can still use the additional EMS pages. /XS must be specified on the BUFFERS command line, and /X on the command line for FASTOPEN and for VDISK. Ask your dealer for the latest DOS 4.00 Corrective Service Diskette (CSD), dated May 1989 or later.

Installing Over MS-DOS and OS/2

The original version of DOS 4.00 checked for the character string "IBM " in the disk boot record. This caused problems when customers attempted to use DOS 4.00 after previous versions of MS-DOS or OS/2 were installed. This requirement has been removed in later versions of DOS 4.00. Check with your vendor to obtain the latest DOS 4.00 CSD.

DOS 4.00 cannot be used with the OS/2 Version 1.0 compatibility box. This compatibility box was designed using DOS 3.30 as a base. If DOS 4.00 compatibility is needed, it is recommended that you upgrade to OS/2 Version 1.2.

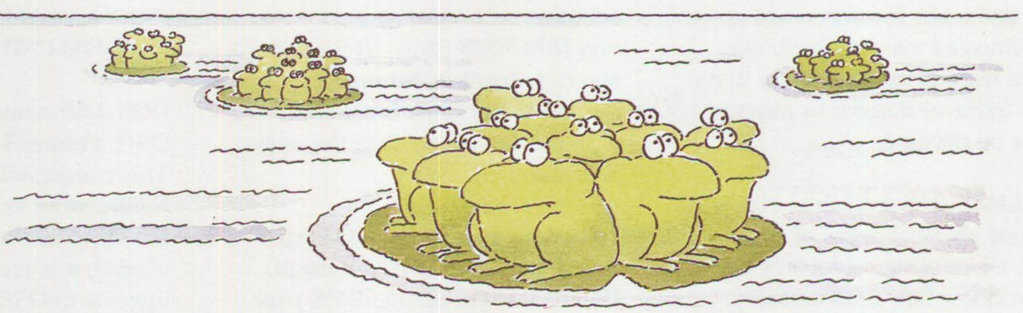
Conclusion

Today, almost all applications are compatible with DOS 4.00. If you experience problems, try one of the solutions mentioned in this article. You may also want to consider contacting your vendor to determine if your software was written for a specific DOS version. If you have not tried DOS 4.00 because you have heard there are problems, you may want to give it a try – there are many new features available that are worth the upgrade.

ABOUT THE AUTHOR

Bruce G. Borkosky is a senior associate programmer at IBM's Entry Systems Division in Boca Raton, Florida. He received a B.A. in music from Ohio Wesleyan University, an M.S. in computer science from the University of Dayton, and is a 1991 Ph.D. candidate from Miami University. After joining IBM in 1984, Bruce's responsibilities have included developing PC DOS and working as an advisor for new memory card development.

'Out of Environment Space' Errors



Thomas H. Sutherland
IBM Corporation
Dallas, Texas

In this article you'll learn how to use the MEM.EXE program to monitor the DOS master environment. You'll also be shown how "Out of Environment Space" errors can be eliminated by using a work-around technique to ensure that enough environment space has been allotted.

Good grief! What is this error, and how can we fix it?

First, let's define an environment. Beginning with DOS Version 2, the concept of an environment was introduced. An environment is a memory area that contains variable-length, zero-terminated ASCII strings. These strings provide information to COMMAND.COM and other programs, such as the current search path, PROMPT command format, and even the location of COMMAND.COM itself.

COMMAND.COM creates the system's master environment from the contents of the CONFIG.SYS and the AUTOEXEC.BAT files, via the PATH, PROMPT, and SET commands. Beginning with DOS Version 3.2, the initial size of this environment can be adjusted via the SHELL command in the CONFIG.SYS file by using the /E parameter. For example:

```
SHELL=C:\COMMAND.COM /P/E:2048
```

sets the size of the environment to 2 KB. The default size (no parameter) is 160 bytes.

The SET command entered with no parameters displays the current environment. Also, the SET command can be used to modify the current environment.

Now that we know how to set the size of this environment, what happens if the provided strings are larger than the allotted environment space? We'll get the "Out of Environment Space" message.

There are a variety of ways to work with this environment space and use it to your advantage. However, the scope of this article is limited to showing how you can use the MEM.EXE program to watch what happens to your environment space as you request changes. We'll also show you how to change this space via batch files to avoid the "Out of Environment Space" message.

Figure 1 contains listings intended to show you how I set up the initial environment. I used a PS/2 Model 80 running DOS 4.01 with correc-

```
AUTOEXEC.BAT:

@ECHO OFF
SET COMSPEC=C:\DOS40\COMMAND.COM
PATH=C:\TDIR;C:\DOS40;D:\PCTOOLS;D:\WORKS;D:\E3;D:\
PROMPT $L$P$G

CONFIG.SYS:

BREAK=ON
BUFFERS=20 /X
FILES=8
LASTDRIVE=G
SHELL=C:\DOS40\COMMAND.COM /P [NO PARAMETER]
DEVICE=C:\DOS40\XMAEM.SYS
DEVICE=C:\DOS40\XMA2EMS.SYS FRAME=D000 P254=C000 P255=C400
DEVICE=C:\DOS40\ANSI.SYS
DEVICE=C:\DOS40\VDISK.SYS 1440 /X
DEVICE=C:\MOUSE.SYS
INSTALL=C:\DOS40\FASTOPEN.EXE D:=(50,25)
```

Figure 1. My Example Files

```
MEM /PROGRAM <ENTER>

Address   Name      Size      Type
_______   _______   _______   _______

000000              000400    Interrupt Vector
000400              000100    ROM Communication Area
000500              000200    DOS Communication Area

000700    IBMBIO    002540    System Program

002C40    IBMDOS    008D50    System Program

00B990    IBMBIO    00A280    System Data
           XMAEM    000140     DEVICE=
           XMA2EMS  004B70     DEVICE=
           ANSI     0011B0     DEVICE=
           VDISK    000890     DEVICE=
           MOUSE    0027D0     DEVICE=
                    0000C0     FILES=
                    000100     FCBS=
                    000200
                    000010     BUFFERS=
                    000270     LASTDRIVE=
                    000CD0     STACKS=
015C20    MEM       000080    Environment
015CB0    IBMDOS    000030    - Free -
015CF0    FASTOPEN  0027C0    Program
0184C0    SHARE     0018A0    Program
019D70    COMMAND   001640    Program
01B3C0    COMMAND   0000A0    Environment ←— Master Environment Of
01B470    COMMAND   000040    Data           160 Bytes (Hex A0)
01B4C0    MEM       012F60    Program
02E430    IBMDOS    0717C0    - Free -

   655360 bytes total memory
   654336 bytes available
   542512 largest executable program size

  8519680 bytes total EMS memory
  7028736 bytes free EMS memory

  8650752 bytes total extended memory
        0 bytes available extended memory
```

Figure 2. MEM /Program Printout

```
SET <enter>

COMSPEC=C:\DOS40\COMMAND.COM
PATH=C:\TDIR;C:\DOS40;D:\PCTOOLS;D:\WORKS;D:\E3;D:\
PROMPT=$L$P$G
```

Figure 3. Master Environment Contents

tive service diskette (CSD) UR25066 installed. The results we'll get should be the same regardless of the PC model used. What we'll be observing is a function of DOS itself.

Figure 2 shows the actual environment count while running the MEM.EXE program in DOS 4.01. The command environment is the master environment. The environment that you see with the MEM entry is the one that COMMAND.COM passes to a program before it is loaded.

Note: In Figures 5, 7, 8, and 11, lines 1 through 17 are the same as in Figure 2.

The SET, PATH, PROMPT, and SHELL commands set up the master environment. Because I used the 160-byte default, the remaining environment space will be 160 minus the number of bytes consumed by the four commands. We can confirm this space usage by entering the SET command without any parameters (Figure 3). Simply count the number of characters shown. Your remaining environment space will be 64 bytes (160 – 96 = 64). Four bytes are for zero termination at the end of each line plus end of file. Therefore, if you have any programs that would add to this environment, the set string should not be any longer than 64 bytes (63 + zero terminator).

However, the following test will show us that we can, in fact, use a string longer than 64 bytes under certain conditions. In Figure 4, SETE.BAT is our test file to prove this theory. The total number of bytes requested is 760 (750 + 10, zero terminators). For those who are counting, the remaining eight bytes are the result of the 16-byte

```
SETE.BAT

SET LINE1=..........................................................
SET LINE2=..........................................................
SET LINE3=..........................................................
SET LINE4=..........................................................
SET LINE5=..........................................................
SET LINE6=..........................................................
SET LINE7=..........................................................
SET LINE8=..........................................................
SET LINE9=..........................................................
SET LINEA=..........................................................
```

Figure 4. SETE.BAT File Contents

```
MEM /PROGRAM <ENTER>
Address  Name      Size     Type
_______  _______   _______  _______
 (Lines 1 through 17 are the same as in Figure 2)

015C20   IBMDOS    0000C0   - Free -
015CF0   FASTOPEN  0027C0   Program
0184C0   SHARE     0018A0   Program
019D70   COMMAND   001640   Program
01B3C0   COMMAND   000360   Environment <--- Master Environment Is
01B730   COMMAND   000040   Data             Now 864 Bytes (Hex 360)
01B780   MEM       000370   Environment
01BB00   MEM       012F60   Program
02EA70   IBMDOS    071180   - Free -

  655360 bytes total memory
  654336 bytes available
  540912 largest executable program size

 8519680 bytes total EMS memory
 7028736 bytes free EMS memory

 8650752 bytes total extended memory
       0 bytes available extended memory
```

Figure 5. Master Environment Size After Running SETE.BAT

paragraph boundary rule. The MEM command shows the expanded space in Figure 5.

Figure 6 shows the result after re-entering the SET command to confirm the newly expanded environment. However, when I invoke a Terminate-and-Stay-Resident (TSR) program, we'll find out that the environment will get "locked in" its memory location. Figure 2 shows the result after rebooting and returning to my 160-byte environment. Figure 7 shows the results after simply loading the PRINT.COM program, which is a TSR.

Now, when I run SETE.BAT as shown in Figure 4, we'll get the "Out of Environment Space" message from DOS. Figure 8 will show

```
SET <enter>

COMSPEC=C:\DOS40\COMMAND.COM
PATH=C:\TDIR;C:\DOS40;D:\PCTOOLS;D:\WORKS;D:\E3;D:\
PROMPT=$L$P$G
LINE1=..........................................................
LINE2=..........................................................
LINE3=..........................................................
LINE4=..........................................................
LINE5=..........................................................
LINE6=..........................................................
LINE7=..........................................................
LINE8=..........................................................
LINE9=..........................................................
LINEA=..........................................................
```

Figure 6. Master Environment Contents After Running SETE.BAT

```
MEM /PROGRAM <ENTER>

Address   Name      Size      Type
_______   _______   _______   _______

 (Lines 1 through 17 are the same as in Figure 2.)

015C20    MEM       000080    Environment
015CB0    IBMDOS    000030    - Free -
015CF0    FASTOPEN  0027C0    Program
0184C0    SHARE     0018A0    Program
019D70    COMMAND   001640    Program
01B3C0    COMMAND   0000A0    Environment ←—— 160 Bytes – No Change
01B470    IBMDOS    000040    - Free - ←———— Free Space
01B4C0    PRINT     0016B0    Program ←————— TSR
01CB80    COMMAND   000040    Data
01CBD0    MEM       012F60    Program
02FB40    IBMDOS    0700B0 - Free -

   655360 bytes total memory
   654336 bytes available
   536608 largest executable program size

 8519680 bytes total EMS memory
 7028736 bytes free EMS memory

 8650752 bytes total extended memory
       0 bytes available extended memory
```

Figure 7. Results After Loading PRINT.COM

```
MEM /PROGRAM <ENTER>

Address   Name      Size      Type
_______   _______   _______   _______

 (Lines 1 through 17 are the same as in Figure 2)

015C20    IBMDOS    0000C0    - Free -
015CF0    FASTOPEN  0027C0    Program
0184C0    SHARE     0018A0    Program
019D70    COMMAND   001640    Program
01B3C0    COMMAND   0000F0    Environment ←—— Size Expanded to 240
01B4C0    PRINT     0016B0    Program          Bytes
01CB80    COMMAND   000040    Data
01CBD0    MEM       000110    Environment
01CCF0    MEM       012F60    Program
02FC60    IBMDOS    06FF90    - Free -

   655360 bytes total memory
   654336 bytes available
   536320 largest executable program size

 8519680 bytes total EMS memory
 7028736 bytes free EMS memory

 8650752 bytes total extended memory
       0 bytes available extended memory
```

Figure 8. Results After Running SETE.BAT

us what happened. The master environment expanded by taking in the "IBMDOS – Free –" area of memory, which was created by DOS as free space when I loaded the TSR. However, because the TSR is now preventing further expansion, you will continue to get the messages and cannot increase or add to your environment space. Again, running the SET command (Figure 9) will confirm our finding: 6 zero terminations + 28 + 51 + 13 + 75+ 67 = 240 bytes. So, how do we increase the environment space for our programs? There are two techniques we can employ:

1. Set the /E: parameter higher in the SHELL= statement of your CONFIG.SYS.

Prior to DOS Version 3.1, the default environment space was 160 bytes and could not be changed. Therefore, the only solution was to keep your path statement to a minimum. With DOS Version 3.1 (see note below) and higher, we can set the size of the master environment. Simply increase the decimal number high enough to accommodate your requirements. The maximum is 32 K. Remember, this use of memory deducts from available conventional memory. Therefore, don't set this any higher than needed.

Note: Under DOS 3.1, the number you specified was not the actual size, but a multiple of the number, times 16 (16-byte paragraphs). For example, /E:20 really means 320 bytes (20 x 16). With the release of DOS 3.2 and higher, the number you chose was the actual decimal size of the environment.

2. Run COMMAND /E:XXX

The second method is to create a temporary environment (secondary

```
SET <enter>

COMSPEC=C:\DOS40\COMMAND.COM
PATH=C:\TDIR;C:\DOS40;D:\PCTOOLS;D:\WORKS;D:\E3;D:\
PROMPT=$L$P$G
LINE1=..........................................................................
LINE2=..........................................................................
```

Figure 9. Master Environment Contents With a TSR Loaded

command processor) shell by entering the following statement at the command line or from within a batch file. Remember, you are using up conventional memory, and therefore you should keep the size of this space to a minimum. Also, you release this space if you have no further need for it by entering "exit" from the command line.

Example:

```
C:COMMAND /E:XXXX
```

Be sure to use the appropriate decimal number based on your DOS version as mentioned previously.

Here is one way you can make use of the shell environment. I have made a batch file called COMSET.BAT, which contains one line:

```
C:\COMMAND /E:2048 SETEE.BAT
```

Next, I load the Print TSR (Figure 7) to lock in the environment space. We already know that we'll get the "Out of Environment Space" message if we run the SETEE.BAT file as shown in Figure 10. However, if I run the COMSET.BAT file first, we'll temporarily increase the environment space to run the SETEE.BAT file. This batch file will create a file showing the MEM /PROGRAM output and the SET program output. Note, these files are created before the EXIT statement is executed to prove that you did have a larger environment during the execution of the two output requests. Figure 11 shows the MEM printout, and Figure 12 is the SET output. Once you have run this program, due to the exit statement your environment will revert back to the original setting and should look like Figure 3.

Summary

With some basic understanding of how DOS creates and employs the environment space, and knowing how to use the MEM.EXE program, you should be able to use and handle programs that require large environments. By increasing your environment and then releasing it, you can minimize the impact on memory usage that is so vital to

```
SETEE.BAT

SET 1INE1=......................................................................
SET 2INE2=......................................................................
SET 3INE3=......................................................................
SET 4INE4=......................................................................
SET 5INE5=......................................................................
SET 6INE6=......................................................................
SET 7INE7=......................................................................
SET 8INE8=......................................................................
SET 9INE9=......................................................................
SET AINEA=......................................................................
SET BINEB=......................................................................
SET CINEC=......................................................................
SET DINED=......................................................................
SET EINEE=......................................................................
SET FINEF=......................................................................
SET PATH=D:\TELIX;%PATH%;
MEM /PROGRAM >COMSET.TST
SET >SET.TST
EXIT
```

Figure 10. SETEE.BAT Contents

```
 MEM /PROGRAM <ENTER>

Address  Name      Size     Type
_______  _______   _______  _______

 (Lines 1 through 17 are the same as in Figure 2)

015C20   COMMAND   000080   Data
015CB0   IBMDOS    000030   - Free -
015CF0   FASTOPEN  0027C0   Program
0184C0   SHARE     0018A0   Program
019D70   COMMAND   001640   Program
01B3C0   COMMAND   0000A0   Environment ←—— Original  Environment
01B470   IBMDOS    000040   - Free -
01B4C0   PRINT     0016B0   Program ←——— TSR
01CB80   COMMAND  000040    Data
01CBD0   COMMAND   001640   Program
01E220   COMMAND   000800   Environment ←— New Environment – 2 KB
01EA30   COMMAND   000040   Data           in Size as Requested
01EA80   MEM       0004F0   Environment
01EF80   MEM       012F60   Program
031EF0   IBMDOS    06DD00   -Free -

  655360 bytes total memory
  654336 bytes available
  527472 largest executable program size

 8519640 bytes total EMS memory
 7028736 bytes free EMS memory

 8650752 bytes total extended memory
       0 bytes available extended memory
```

Figure 11. Results Showing Size of New Temporary Environment

DOS users. The actual byte usage count you have seen varies between systems. The concept of taking the guesswork out of setting up the environment and avoiding the "Out of Environment Space" message is what we're trying to achieve.

ABOUT THE AUTHOR

Thomas Sutherland is an associate market support representative in the Personal Systems Technical Support Center in Dallas, Texas. He received his B.A. in business from Texas Wesleyan College in Fort Worth, Texas. He joined IBM in 1966 as a field customer engineer, and presently supports PCs for the DOS and OS/2 operating systems.

```
 SET <enter>

COMSPEC=C:\DOS40\COMMAND.COM
PROMPT=$L$P$G
1INE1=...........................................................................
2INE2=...........................................................................
3INE3=...........................................................................
4INE4=...........................................................................
5INE5=...........................................................................
6INE6=...........................................................................
7INE7=...........................................................................
8INE8=...........................................................................
9INE9=...........................................................................
AINEA=...........................................................................
BINEB=...........................................................................
CINEC=...........................................................................
DINED=...........................................................................
EINEE=...........................................................................
FINEF=...........................................................................
PATH=D:\TELIX;C:\TDIR;C:\DOS40;D:\PCTOOLS;D:\WORKS;D:\E3;D:\;
```

Figure 12. New Temporary Environment Contents

A New LAN Requester for DOS Systems

Roy A. Feigel, Steven French, and Albert Chang
IBM Corporation
Austin, Texas

This article presents an overview of the DOS LAN Requester (DLR) program. DLR, which is a part of the OS/2 LAN Server 1.2 package and replaces PC LAN Program 1.3 (PCLP), is for workstations used in an OS/2 LAN Server 1.2 domain. DLR provides many functional enhancements over PCLP including flexible installation options, reduced low memory requirement and improved performance.

Packaging

The DOS LAN Requester (DLR), designed for use in an OS/2 LAN Server 1.2 environment, is a replacement product for users of the PC LAN Program 1.3 (PCLP). DLR and the required NETBIOS support provided by LAN Support Program 1.1 (LSP) are included in the OS/2 LAN Server 1.2 package. Purchase of the server package permits users to copy DLR and LSP for use on up to 128 workstations. The server package contains separate diskettes for the DLR and LSP.

Installation

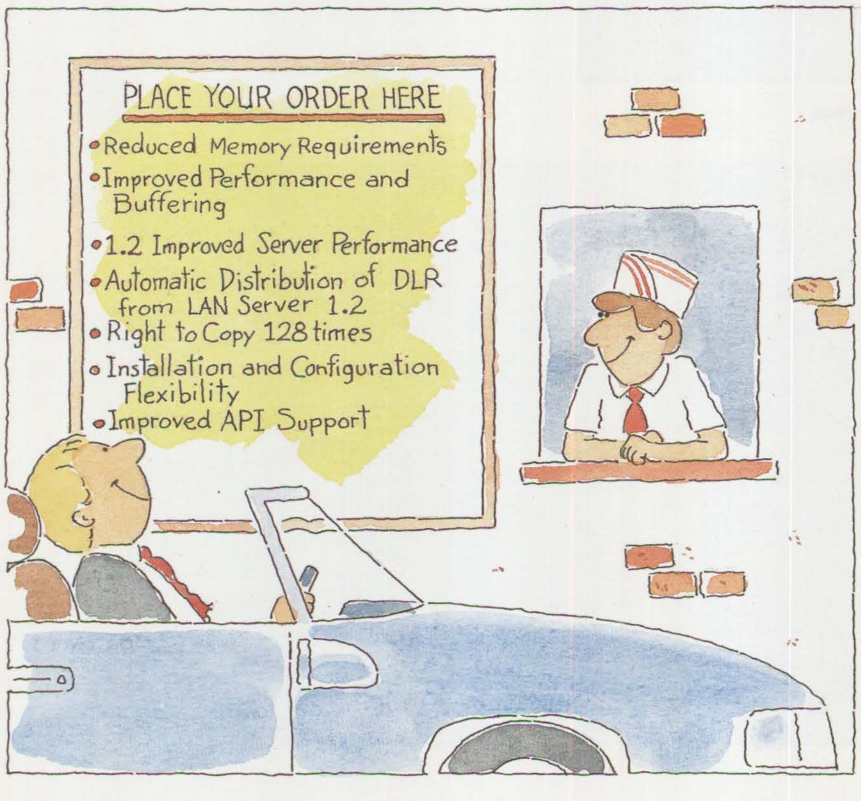
Installation and setup of various workstation configurations of DLR are straightforward. Among the available configurations are systems that start from a hard disk or diskette, and systems set up for Remote Initial Program Load (these systems may or may not contain media).

The OS/2 LAN Server installation process, which is performed after installing Extended Edition 1.2, also has an option to install DOS, LSP, and DLR onto the server, which is necessary to support all but the first of the following environments:

Hard Disk Based Systems: DLR and LSP may be installed directly onto a hard disk system using the respective installation utilities. Both processes require the user to answer a few simple questions. Installation on a system already configured with DOS 3.30 or 4.00 can take as little as 10 to 15 minutes. The DLR installation utility has been improved from PCLP by allowing the user to install the program in any directory on any partition (C: through F:). The program consists of approximately 60 files and is installed into a single directory. The default destination is C:\DOSLAN.

Automatic Download of DOS LAN Requester: A second option for hard disk systems requires that DLR and LSP be installed on a OS/2 LAN Server 1.2 domain controller.

When a PCLP 1.3 user attempts to log on to an OS/2 LAN Server 1.2 domain, the DLRINST service running on the OS/2 LAN Server 1.2 domain controller intercepts the logon request. The logon will appear successful, but the user will notice that the usual entries on the PCLP applications menu have been replaced with an option to initiate the DLR download process. When selected, the DLR and LSP programs will be copied from the server to the hard disk of the workstation. The DLRINST service examines the current configuration of


PLACE YOUR ORDER HERE
• Reduced Memory Requirements
• Improved Performance and Buffering
• 1.2 Improved Server Performance
• Automatic Distribution of DLR from LAN Server 1.2
• Right to Copy 128 times
• Installation and Configuration Flexibility
• Improved API Support

PCLP to determine the default DLR configuration, and the AUTOEXEC.BAT and CONFIG.SYS files are updated to allow DLR and LSP to start.

PCLP programs are removed from the disk to ensure enough disk space is available to complete the process. The process takes approximately three minutes. When complete, the workstation is automatically rebooted, DLR is started, and the user is presented with the DLR logon panel, at which point the user can log on to the OS/2 LAN Server 1.2 domain.

Diskette-Based Systems/Remote Initial Program Load: For systems that start from diskette or that are configured for Remote Initial Program Load (RIPL), the system administrator can create IPL diskettes or RIPL images from standard definitions. These definitions include the portions of DOS, LSP, and DLR needed to start the system and the DLR program. These diskette images define proper connections to the domain controller for subsequent access to DOS and DLR program files. Standard image definitions can be edited or new ones can be created to meet user needs.

Memory Requirements

The available memory within the DOS addressable range below 640 KB is critical to users of the large DOS applications available today. A major design point of DLR is to provide functional enhancements to PCLP while using less resident memory below 640 KB. This has been accomplished by allowing configuration of optional functions and by using extended and expanded memory technologies.

DLR allows for the optional loading of several functions at initialization time so users can load only those services desired. Figure 1 shows the Redirector or RDR configuration, which includes the basic file/print redirection functions and several functional enhancements over PCLP, including remote named pipe support, auto session reconnect, and performance enhancements.

Optional services for the RDR configuration include local Application Program Interface (API) support and password encryption, which provides the same password encryption algorithm used by OS/2 Extended Edition 1.2 requesters.

The local API can be loaded in two sections: mailslot and server enumeration, and full API support. The former is a prerequisite for the latter. It should be noted that the preceding support is needed only for

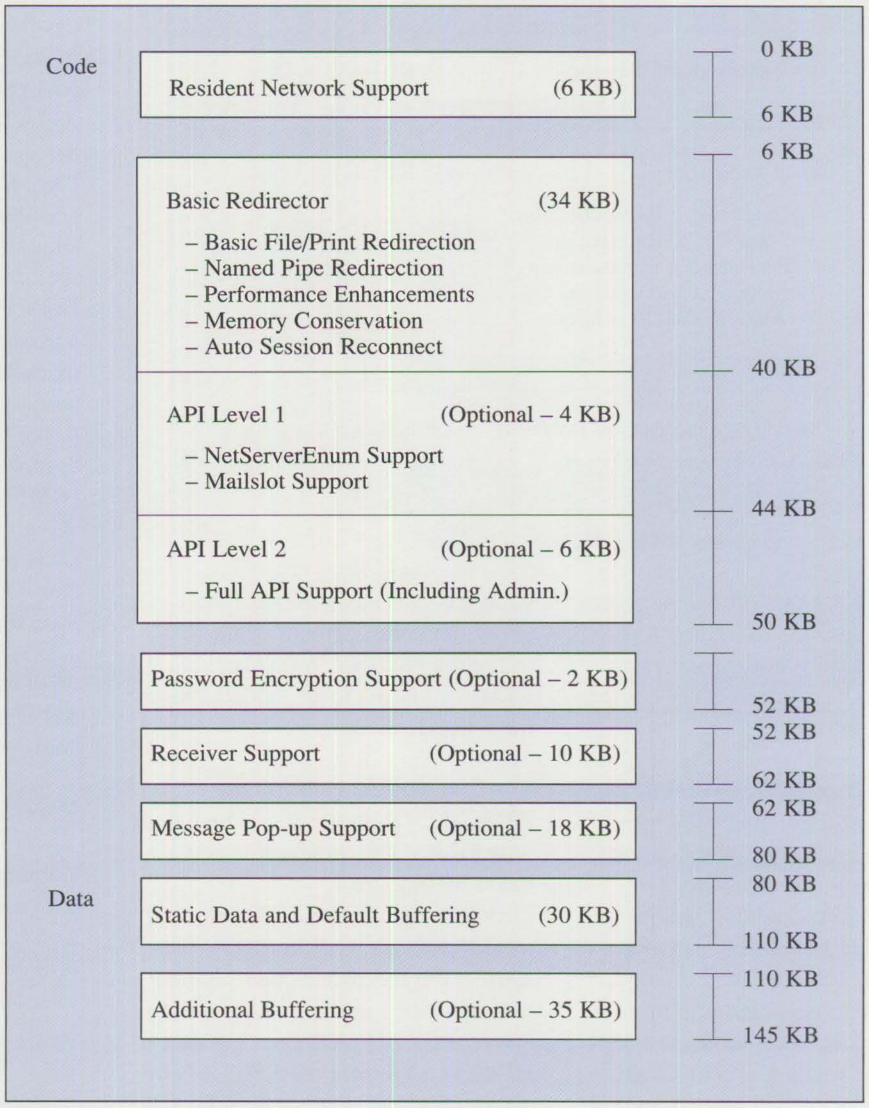

Code
Resident Network Support (6 KB)
Basic Redirector (34 KB)
– Basic File/Print Redirection
– Named Pipe Redirection
– Performance Enhancements
– Memory Conservation
– Auto Session Reconnect
API Level 1 (Optional – 4 KB)
– NetServerEnum Support
– Mailslot Support
API Level 2 (Optional – 6 KB)
– Full API Support (Including Admin.)
Password Encryption Support (Optional – 2 KB)
Receiver Support (Optional – 10 KB)
Message Pop-up Support (Optional – 18 KB)
Data
Static Data and Default Buffering (30 KB)
Additional Buffering (Optional – 35 KB)
0 KB
6 KB
6 KB
40 KB
44 KB
50 KB
52 KB
52 KB
62 KB
62 KB
80 KB
80 KB
110 KB
110 KB
145 KB

Figure 1. DOS LAN Requester RAM Utilization – Maximum Configuration

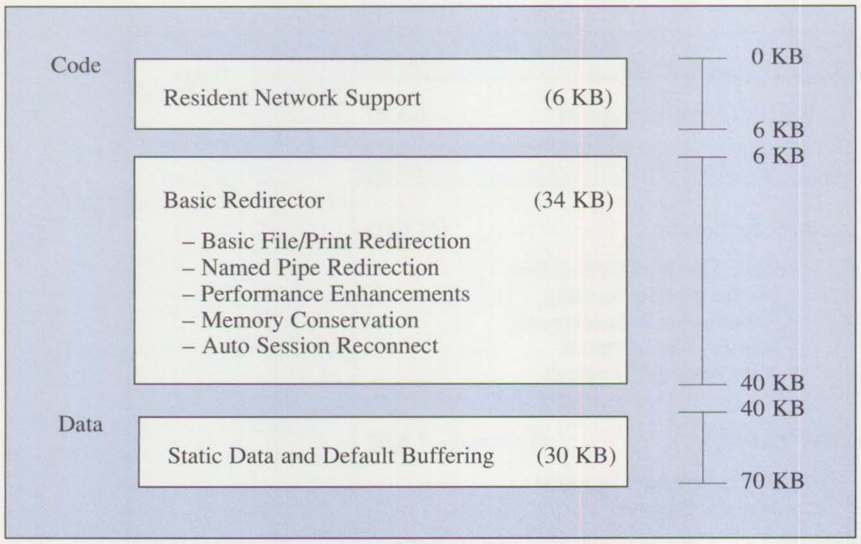

Code
Resident Network Support (6 KB)
Basic Redirector (34 KB)
– Basic File/Print Redirection
– Named Pipe Redirection
– Performance Enhancements
– Memory Conservation
– Auto Session Reconnect
Data
Static Data and Default Buffering (30 KB)
0 KB
6 KB
6 KB
40 KB
40 KB
70 KB

Figure 2. DOS LAN Requester RAM Utilization – Default

local API. Remote API call support is provided in the base redirector function and does not require local API support. For more details, refer to the API section.

The Receiver or RCV configuration includes all the functions and options of the Redirector and contains message reception capabilities. These messages can be logged to a user-specified file. Also, the message pop-up service can be loaded, which causes a message pop-up to be displayed once messages are received. The logging and message pop-up can be paused and continued independently.

DLR does not require the IFSFUNC.EXE, as is the case with PCLP 1.3, which saves approximately 15 KB in a DOS 4.00 environment.

DLR has two mechanisms that allow the code segment of the redirector (REDIR33.EXE or REDIR40.EXE) to be loaded above the 640 KB mark.

The first method is to use the HIMEM.SYS driver that is included with DLR. This function is activated by placing a DEVICE=HIMEM.SYS statement in CONFIG.SYS. When DLR is started, the redirector code segment will be placed in the first segment above the 1 MB address. Use of HIMEM.SYS requires a 80286® or above processor, and at least 64 KB of extended memory.

The second way to free up memory in the 640 KB range is to use the DOS 4.00 EMS support. For 8088 and 80286 processors, an Expanded Memory Adapter (XMA) and the use of XMA2EMS.SYS is required.

```
RDR RAFEIGEL LS12DOM
/SRV:8  /ASG:29   /NBC:4  /NBS:1K   /BBC:1   /BBS:4K
/PBC:4  /PBS:128  /PFS:32 /PFT:900  /PWT:250 /KUC:600 /KST:600
/WRK:1111211012
```

Figure 3. Default DOSLAN.INI

The extended memory of 80386 processors can be used as expanded memory by including XMAEM.SYS along with XMA2EMS.SYS.

Figure 1 represents the maximum configuration. Figures 2 and 3 show the the default DLR memory requirements and the default DOSLAN.INI file. Figure 4 shows a minimum DLR configuration using HIMEM.SYS and reduced network buffers. Note that the optional local API support, if specified, is also loaded into the HIMEM area. Figures 5 and 6 show the CONFIG.SYS and the DLR parameter values (displayed from the NET START command) used to achieve this configuration in a DOS 3.30 environment. Total DOS available memory below 640 KB is approximately 520 KB in this configuration.

Performance Improvements

Concepts: The new DOS LAN Requester delivers better performance than its predecessor, the PC LAN Program (PCLP), through improved internal buffering, lower memory usage, an improved network command dialect ("SMB protocol"), more configuration flexibility, and a faster server platform.

Buffering: The DOS LAN Requester has three types of user-configurable buffers: network buffers, big buffers, and pipe buffers. There is also a non-configurable facility to do transfers directly to memory for large file reads. Users are permitted to set the size and number of each of the three buffer types. Because default values are not optimal for all customer environments, network administrators may change the default values to improve the performance of commonly used applications. Because each buffer takes away from memory in the lower 640 KB available to an application, delicate balancing between application program and DOS LAN Requester memory needs may be required.

Network buffers are used for small reads and writes. Because many database applications read and write small records, the number of network buffers can affect performance dramatically in database-intensive environments. The size of typical database records may vary, so changing the network buffer size accordingly can also improve performance.

If a read or write is too large to use a network buffer, DLR will attempt to use a larger "big" buffer instead. Some network operations, such as printing to a network server printer, or writing a small word processing document to a redirected drive, can use big buffers. In some environments big buffers can be disabled with little performance impact.

For the largest reads and writes, "raw" I/O can be performed, which allows data to be read from or written directly into the applications memory. As long as the server has a large (usually 64 KB) buffer available at the time, raw I/O can be performed very quickly. Raw I/O is especially beneficial when loading a large program or data file. Although this feature is not configurable, it can be disabled from either the server or requester.

The third, user-configurable set of buffers is the pipe buffers. These allow the DOS requester to cache read and write requests when accessing remote-named pipes. As with the two other buffer types, the size and number are configurable.

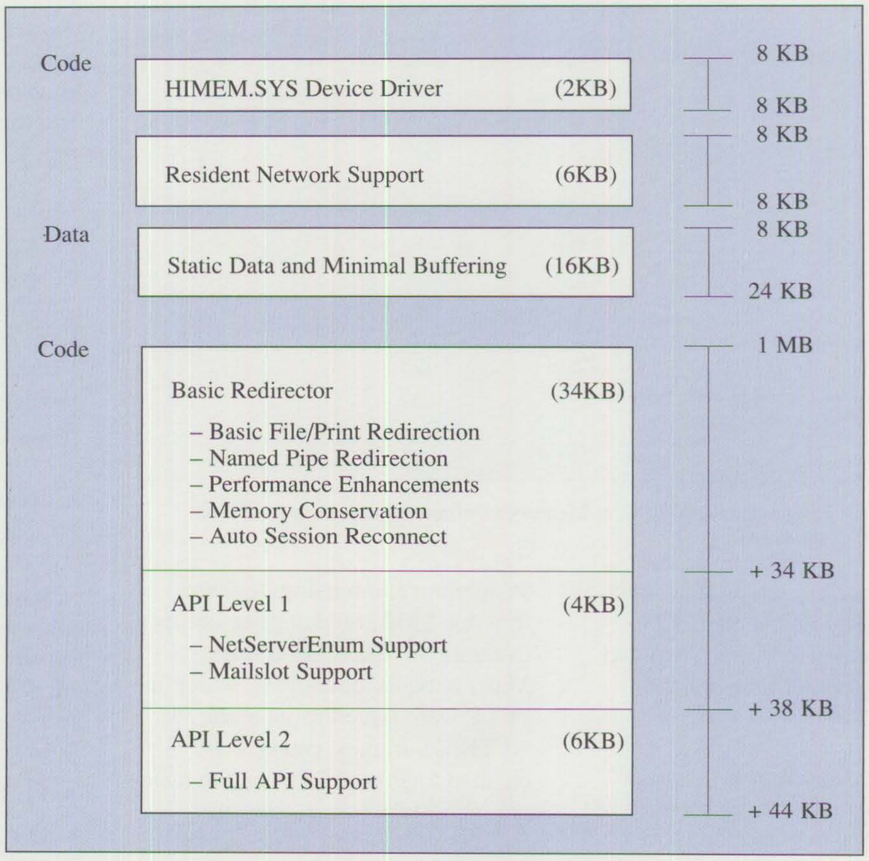


Figure 4. DOS LAN Requester RAM Utilization – Minimum Configuration

Because DLR does not require pipe buffers for proper functioning, named pipes can be disabled, especially in environments where named pipe applications are used infrequently.

Server Performance: DLR is custom-designed to take advantage of a number of performance improvements in LAN Server 1.2. Therefore, it performs best when accessing LAN Server 1.2 resources. Also, DLR improvements provide better performance than PCLP requesters when accessing older servers (LAN Server 1.0 and PCLP). Among the advantages of the new

```
files=8
fcbs=4,0
BREAK=ON
LASTDRIVE=F
SHELL=C:\DOS33\COMMAND.COM /E:2000 /P
BUFFERS=20
device=C:\DOSLAN\himem.sys
DEVICE=C:\DOSLAN\LSP\DXMAOMOD.SYS 001
DEVICE=C:\DOSLAN\LSP\DXMCOMOD.SYS
DEVICE=C:\DOSLAN\LSP\DXMTOMOD.SYS S=4  C=8  ST=1  O=Y  N=6
```

Figure 5. CONFIG.SYS for DLR Minimum Memory Configuration

```
IBM DOS Local Area Network Requester
(C) Copyright IBM Corporation 1984, 1989
Level:  WR 4000

Configuration:  REDIRECTOR
Computer Name:  RAFEIGEL
Default Domain: LS12DOM

/SRV:     2                  /ASG:       5
/NBC:     3                  /NBS:     128
/BBC:     0                  /BBS:       0
/PBC:     0                  /PBS:       0
/PFS:     0                  /PFT:       0
/PWT:   250                  /KUC:     600
/KST:   600                  /HIM
/WRK: 1111211012
```

Figure 6. DLR Parameter for Minimum Memory Configuration as Displayed from NET START Command

1.2 release of the server are access control improvements, the new High Performance File System and cache, and a faster network file manager.

Server Message Blocks: One of the principal differences between PC LAN Program and the DOS LAN Requester is the Server Message Block (SMB) dialects that each supports. Commands are passed between the server and requester through SMBs, which can be thought of as the network equivalent of many of the functions available in the DOS API. For example, there are SMBs for reading, locking, and writing to files, just as there are similar API calls in DOS and OS/2. Because these SMBs are shorthand commands, they simplify the communication of common network operations. SMB headers and their data are placed in NETBIOS frames for transfer across the network. While changes to the SMB protocol are invisible to application programs, they can significantly impact performance.

The newer SMB dialect that DOS LAN Requester uses allows it to combine common operations such as opening and reading, together into one SMB, reducing the number of required NETBIOS frames. Many network operations, which in PCLP were issued in separate NETBIOS frames, can now be chained together and placed in a single NETBIOS frame. Because reads are frequently done immediately after opening a file, and writes are frequently done after locking a range of bytes, these combined SMBs are very useful.

A facility exists that allows requesters and servers to negotiate the level of SMB dialect to be used. This facility allows for interoperability among current and future LAN products. When DOS LAN Requester machines use resources on PCLP servers, they use the basic "core" SMB protocol. Although more restricted, they support most common operations. The same core protocol is used when PCLP requesters access OS/2 LAN Server 1.2.

Configuration Flexibility: With more than 30 separate configurable parameters, the DOS LAN Requester allows users to customize a machine for individual needs. While the default parameter values are set for typical environments, users can change a range of options such as the buffer sizes and counts, and the network behavior on reading, locking, and writing.

Miscellaneous:
PC LAN Program used a function in DOS 4.00, IFSFUNC.EXE, which caused execution of an extra layer of code when accessing the redirector. By eliminating that intermediate step, performance has improved in DLR.

Connectivity

DLR users are required to log on to a OS/2 LAN Server 1.2 domain before access to network resources are allowed from either the command line or menu interface. Once logged on, connections can be made to resources on the following server products:

- PC LAN Program 1.3
- OS/2 LAN Server 1.0
- OS/2 LAN Server 1.2

Application Program Interface

DOS LAN Requester (DLR) Application Programming Interface Support: The Application Programming Interface (API) is designed so users can easily perform functions in their application programs. It shields users from the underlying implementation details. For example, an application program could enumerate the shared network resources on a server and link those that are needed automatically using the LAN API.

Most functions in the OS/2 LAN Server 1.2 are supported on DOS LAN Requester 1.2. There are more than 100 function calls for DOS LAN Requester application de-

Category	Purpose
Access Permission	View or modify user or group access permission on a resource.
Auditing	View or clear the audit file, which contains an audit trail of operations that occur on a server.
Configuration	Obtain network configuration information of the system from the IBMLAN.INI file.
Connection	List all connections made to a server by a requester client, or all connections made to a server's shared resource.
Domain	Provide domain-wide information.
File	Provide functions to monitor which file, device, and pipe resources are open on a server, and to close one of these resources, if necessary.
Group	Control groups in the User Accounts Subsystem (UAS).
Handle	Get and set information on a per-handle basis.
Mailslot	Provide one-way interprocess communication.
Message	Send, read, and log messages.
Named Pipe	Supplies interprocess communications between server and requester
Remote Utility	Enable applications to copy and move remote files, and access the time-of-day on a remote server.
Requester	Control the operation of requesters.
Serial Device	Control shared serial devices and their associated queues.
Server	Enable remote administration tasks to be performed on a local or remote server, and to get and set server information
Service	Start and control network service programs.
Session	Control network sessions established between requesters and servers.
Share	Control shared resources on a server.
Statistics	Obtain and clear the operating statisitics for requesters and servers.
Use	Use or control the uses between shared resources on a server and a requester.

Figure 7. Application Programming Interface Categories

velopers to write network-aware applications, client/server applications, distributed processing applications, new network services, and so on.

These API functions allow users to perform network functions, such as remote interprocess communication, messaging, access control, resource sharing and use, statistics, file transfer, and so forth. The API categories supported on DOS LAN Requester 1.2 are shown in Figure 7. The most common types of operation the users can perform are add, delete, enumerate, get, and set.

Differences From OS/2 LAN Server/Requester API Support: The API support on DOS LAN Requester is primarily for existing DOS users. Be aware that DOS, unlike OS/2, does not support pointer checking, semaphores, or shared memory segments. All file names, directory names, or parts of a path name, including Universal Naming Convention (UNC) server and network names, must follow DOS naming conventions.

Unlike OS/2, DOS does not support dynamic link libraries (DLL). Under DOS, the application has to be linked with DOSNET.LIB and SYSCALL0.LIB libraries.

For more details about using the DLR APIs, refer to the sections of DOS considerations in the *OS/2 LAN Server Version 1.2 Application Programmer's Reference*, S01F-0256.

Configuration

This section gives a detailed description of the various DLR parameters along with their memory requirements. The NETBIOS requirements for each configuration are also given.

The parameter defaults have been chosen to balance memory usage and function. Values should be selected based upon the specific user needs. The DOSLAN.INI file contains the parameter values to be used each time DLR is started. Any parameter value can be overridden by explicitly indicating the desired parameter and value on the command line. System default values are assumed for all parameters not specified in DOSLAN.INI or on the command line. DOSLAN.INI must be in the same directory as the DOS LAN Requester program files.

DLR Parameters:

- /SRV:n – Defines the maximum number of unique servers that can be accessed at one time. The value of **n** must be from 1 through 251. The default value for **n** is 8. Each /SRV requires approximately 28 bytes of memory.
- /ASG:n – Defines the maximum number of network resources that can be accessed at one time. The value of **n** must be from 1 through 254. The default value for **n** is 29. Each /ASG requires approximately 75 bytes of memory.
- /NBC:n – Defines the maximum number of buffers used for network file or print caching. The value of **n** must be from 3 through 64. The default value for **n** is 4.
- /NBS:n[K] – Defines the buffer size used for network file or print caching. The value of **n** must be from 128 through 16 KB. The default value for **n** is 1 KB. /NBS times /NBC bytes of memory are required. If /API is specified, the minimum is 1 KB.
- /BBC:n – Defines the number of large buffers used for file caching (or printing). Values for **n** must be from 0 through 64. The default for **n** is 1.
- /BBS:n [K] – Defines the size of the large buffers used for file caching (or printing). Values for **n** must be from 1 KB through 32 KB. The default for **n** is 4 KB. If /BBC is 0, then /BBS is implicitly set to 0. /BBC times /BBS bytes of memory are used. /BBS must be greater than /NBS.
- /PBC:n – Defines the maximum number of buffers used for remote pipe caching. The value of **n** must be from 0 through 16. The default value for **n** is 4. A value of **0** indicates no pipe buffering.
- /PBS:n[K] – Defines the size of the buffers used for remote pipe read-ahead. The value of **n** must be from 128 through 16 KB. The default value for **n** is 128. If /PBC is 0, /PBS is implicitly set to 0. /PBS times /PBC bytes of memory are required.
- /PFS:n[K] – Defines the maximum size pipe write that can be buffered. A single write of more than **n** characters is written directly to the server. The value of **n** must be from 0 through 32 KB. The default value for **n** is 32. If n is greater than the /PBS value, /PFS is set to the /PBS value. If /PFS is 0, no pipe writes are buffered. If /PBC is 0, /PFS is automatically set to 0.
- /PFT:n[K] – Defines the number of quarter seconds to wait before flushing a pipe write buffer. The value of **n** must be from 0 through 65535. The default value for **n** is 900. If /PFT is 0, no pipe writes are buffered. If /PBC is 0, /PFT is automatically set to 0.
- /PWT:n[K] Defines the number of milliseconds to wait for blocked pipe I/O. The value of **n** must be from 0 through 65535. The default value for **n** is 250. A value of **0** indicates no wait time. A value of **65535** indicates wait forever.
- /KUC:n[K] – Defines the number of seconds after the last reference to keep a UNC connection active. When the time expires, the session is disconnected. The value of **n** must be from 0 through 32 KB. The default value for **n** is 600.
- /KST:n[K] – Defines the number of seconds to keep search first and next directory entries in a local buffer. When the time expires, the entries are no longer valid. The value of **n** must be from 0 through 32 KB. The default value for **n** is 600.
- /NVS:n – Indicates NetServer-Enum API support and defines the number of server entries supported. If this parameter is not present, there is no NetServer-Enum support. No NetServer-Enum support is the default. If used, the value of **n** must be from 1 through 255. NetServer-Enum requires 70 bytes of memory for each specified entry.
- /NMS – Indicates mailslot API support and defines the number of mailslots that can be created. If this parameter is not present, there is no Mailslot API support. No Mailslot support is the default. If present, the value of **n** must be from 1 through 64. Receiver configuration requires a minimum value of **2**. A value of **1** is automatically adjusted to 2 for the Receiver service. If /NVS is present, the minimum number of **n** for the configuration is implied.
- /NDB:n – Indicates datagram support and defines the number of buffers used to receive the mailslot and server announcement datagrams. If present, the value of **n** must be from 2 through 255. The default is is 2 (3 if both /NVS and /NMS are present). Each buffer requires approximately 580 bytes of memory.
- /API – Indicates support for general network application programming interfaces (API), if present.

Nonpresence is the default. API support requires approximately 9.5 KB of memory and /NBC, /NBS, /NVS, and /NMS support as prerequisites. See these parameters for additional information.

- /EMS – Indicates that the redirector component is loaded into high memory using the LIM 4.0 specification. This parameter is valid only for 386-based machines with expanded memory or 286-based machines with an XMA card available. The default action is to load the redirector into low memory (below 640 KB). /EMS and /HIM parameters are mutually exclusive.
- /HIM – Indicates that the code segment of the redirector component is loaded into the first segment above the 1 MB address. This parameter is valid only for 286- and 386-based machines with extended memory available. The default action is to load the redirector into low memory (below 640 KB). /HIM and /EMS parameters are mutually exclusive.
- /ENC – Indicates support for password encryption. The default is nonpresence. Password encryption requires 1.5 KB of memory. Password encryption support is not required.
- /MBI:n[K] – Indicates that the size of the message buffer **n** can be from 512 through 4 KB. The default value for **n** is 512.
- /POP – Specifies that the pop-up message service is to be used during this network session. Nonpresence is the default.
- /RPL – Indicates a remote-IPLed workstation. Required for workstations that require DOS LAN Requester to be accessed from a remote-IPL server. Nonpresence is the default.

A.	Write Behind/Write through behavior
0	Set the write-through bit for Open&X and WriteBlockRaw protocols. This disables any server disk cache or write-behind operations on all files.
1	Clear the write-through bit for Open&X and WriteBlockRaw protocols. This allows the server to write behind data written to the file and removes one SMB exchange on raw I/O. This is the default.
B.	Write&X behind
0	Disallows all Write&X behind behavior.
1	Issue Write&Close, Write&Unlock, and Close SMBs asynchronously behind. This means that DOS LAN Requester does not wait for SMBs to be responded (default).
C.	Specifies NETBIOS behavior. Valid values are:
0	NETBIOS cannot accept NCBs at interrupt.
1	NETBIOS can accept NCBs at interrupt time (default).
D.	Core read protocol behavior
0	Limit transfers using core protocols to the minimum negotiated buffer size.
1	Uses "message incomplete" by posting a receive for the SMB header, then posting a receive for data once the SMB has been received (default).
2	Break reads larger than the Redirector buffer size into two read operations; the first read for the SMB header and the second directly into the reader address.
E.	Buffer read-ahead behavior
0	Always read in a full buffer when the read is less than the buffer size.
1	Always read in only the user's requested read amount.
2	Read /NBS bytes if sequential (default).
F.	Hard error behavior
0	Return errors without popping them up on the screen
1	Pop-up error messages on the screen.
G.	Big buffer read-ahead behavior
0	When a read is larger than the redirectory buffer but smaller than the big buffer, read only the requested amount.
1	When a read is larger than the redirector buffer, but smaller than a big buffer, fill in the big buffer even if it is smaller than the requested amount (default).
H.	Process exit behavior
0	Never send process exit SMBs.
1	Always send process exit SMBs
2	Send process exit SMBs depending on RPDB structure.
I.	Opportunistic locking behavior
0	Do not use Oplock.
1	Use Oplock.
J.	Open&Read behavior
0	Never Open&Read
1	Open&Read on read and execute files.
2	Open&Read on read, execute, and write files.

Figure 8. DLR Work Heuristic Descriptions

```
64K ≅ (/SRV * 28) + (/ASG * 75) + (/NBC * (/NBS + 90))
+ (/BBC * (/BBS + 90)) + (2 * /PBC * (/PBS + 90))
+ (/NMS * 130) + (LASTDRIVE * 80) + ((FILES + FCBS) * 30)
+ 10K
```

Figure 9. Formula for Determining Maximum Parameter Values

Minimum NETBIOS resource requirements are:

Resource:	Redirector:	Receiver:
Sessions	/SRV + 1	/SRV + 2
Commands	/NBC + 5	/NBC + 8
Names	3 (4 if remote IPL)	5 (6 if remote IPL)

Figure 10. NETBIOS Requirements

- /WRK: *ABCDEFGHIJ* – Defines the behavior of a set of internal working heuristics. This parameter provides the ability to specify how to use a set of redirector component functions. The default value is 1111211012. The functions of each heuristic and its values are in Figure 8.

Notes:

Make sure your workstation has enough memory to support the configuration selected and still leave enough memory to allow your applications to run.

A maximum of 64 KB is available for the data segment of DLR. Each parameter in the following formula requires memory allocation within this segment. The sum of all of the memory required as indicated cannot exceed 64 KB (Figure 9).

The minimum NETBIOS resource requirements are indicated in Figure 10. These parameters are specified in the DEVICE=DXMT0MOD.SYS statement on the CONFIG.SYS file.

ABOUT THE AUTHORS

Roy Feigel is an advisory programmer in IBM's Entry Systems Division in Austin, Texas. He is the lead designer for the OS/2 LAN Server 1.2/Extended Edition 1.2 LAN Requester/DOS LAN Requester. In 1982, Roy joined IBM in Austin as a programmer assigned to text applications on the 5520 Administrative System. He received a B.S in computer science from the University of Southwestern Louisiana.

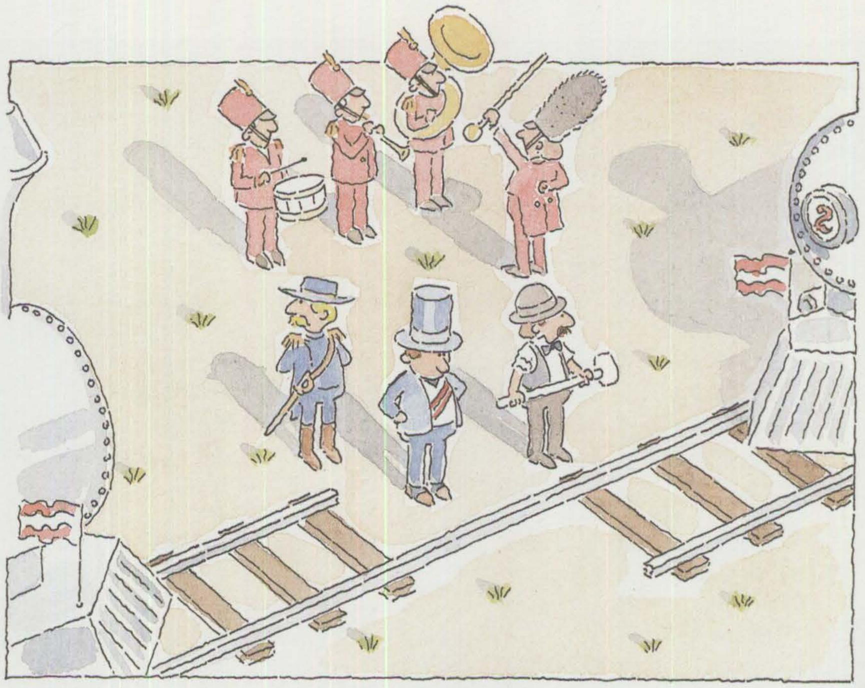
Steven French is a senior associate programmer in IBM's Entry Systems Division in Austin, Texas. He joined IBM in 1989 and presently is the technical interface to LAN development. Steven received a B.A. in computer science and an M.S. in electrical and computer engineering from Rice University.

Albert Chang is a senior associate programmer in IBM's Entry Systems Division in Austin, Texas. Currently, he is a member of the OS/2 LAN Server 1.2 design team. Albert joined IBM in 1983 and has held development and test positions in AIX, OS/2 Database Manager and OS/2 LAN Server. He received a B.S. in chemistry from Chung Yuan University of Science and Engineering in Taiwan, Republic of China, and an M.S. in computer science from the University of Houston.

Creating a Dialog Box Dynamically Using WinCreateDlg

Eric Hauser
Dallas, Texas

This article shows how to dynamically create the necessary structures for the WinCreateDlg function call in OS/2 Presentation Manager. An example program demonstrates the process. Some knowledge of Presentation Manager programming techniques is assumed.



There are two ways to create a dialog box for Presentation Manager. The easiest, and most straightforward way, is by using the resource compiler. To define a dialog box using resource script, a programmer must simply define the size of the box and list the types and positions of the controls within the box. While this is simple, it is limiting in that the dialog box must be completely defined prior to running the program. In some situations, the particular controls on a dialog box may be data dependent. In these situations, the dialog box must be defined during program execution by using the **WinCreateDlg** function. The use of this function is the second way to create a dialog box.

The **WinCreateDlg** function expects a parent and owner window handle, a dialog box procedure, a pointer to the dialog template structure, and a pointer to data passed in the **WM_INITDLG** message. The difficult part is setting up the array of structures in memory that - **WinCreatDlg** expects to find.

The Example Program

The example program creates a dialog box containing a number of check boxes that are equal to the number of files in the current directory whenever the menu item **Create Dialog** is selected. The dialog box will also contain **OK** and **Cancel** pushbuttons. When the user presses the **OK** pushbutton, all files checked are deleted. Pressing the **Cancel** pushbutton will dismiss the dialog box without erasing any files.

The function that actually creates the dialog box is called **check_dialog**. The **check_dialog** function is broken down into eight steps:

- Variable declaration
- Frame setup
- Check box setup
- Header definition
- Frame definition
- Check box definition
- Pushbutton definition
- Function call

The steps in the process are explained in Figures 1a through 1c. The program is shown in Figures 2a through 2f. The numbers on the left are not a part of the program; they are for reference only.

Conclusion

Dynamically creating a dialog box is a tedious endeavor. It is recommended that, if possible, programmers should create their own dialog boxes with the aid of the resource compiler. The intention of this article is to explain the process of creating a dialog box dynamically for those situations where a static dialog box is not practical.

ABOUT THE AUTHOR

Eric Hauser is a cooperative education student at IBM in Dallas, Texas. He has extensive training in OS/2 programming techniques and has specialized in Presentation Manager. Eric is a sophomore at the University of North Texas pursuing a degree in computer science.

Step 1 – Variable declaration (lines 104-121)

The **check_dialog** function uses a global data area, two user-defined structures, and several local variables.

The following structure is not necessary to create a dialog box dynamically. It does help to simplify the creation process.

struct _checkstruct{

CHAR text[21];	**text** is used to define the text that appears for dialog control.
POINTL pt;	**pt** is the point at which the control will appear.
SIZEL szl;	**szl** defines the actual size of the control.
SHORT id;	**id** is the window ID of the control
SHORT offset;	**offset** defines the number of bytes from the dialog header that the text for the control appears. This will become clearer in Step 5.

}

The following structure is used to pass data to the dialog procedure. More information can be placed in the structure as required.

struct_dialog {

int total;	**total** is the number of files in the current directory.

}

BYTE DataArea[2048]; – **DataArea** is used as a workspace in memory in which the dialog box will be created. The actual size of this workspace depends on how complicated or simple the dialog box will be. This is the only global variable used.

DLGTEMPLATE *dt; – **dt** is a pointer to the DLGTEMPLATE structure that is defined in the PMWIN.H header file included with the OS/2 Toolkit. All information about the dialog box will be placed in memory relative to **dt.**

CHAR *cTemp; – **cTemp** is used to position the text for a dialog control at the proper offset into the DataArea. This offset is identified in the **_checkstruct** structure and will be explained in detail in Step 5.

HWND hwndDlg; – **hwndDlg** is the handle to the dialog box that is returned from the **WinCreateDlg** function call in Step 8.

struct _checkstruct cs[20]; – **cs** is an array of the user-defined structure **_checkstruct**. **cs[0]** defines the frame of the dialog box. The remaining elements of the **cs** array each define a check box for the files in the current directory.

struct _dialog *data; – **data** is a pointer to the **_dialog** structure. It is passed as the last parameter in the **WinCreateDlg** function call. This pointer appears in message parameter 2 in the dialog procedure on a WM_INITDLG message.

HDIR hdir; USHORT usSearch- Count; FILEFINDBUF findbuf; – These variables are not used for the actual creation of the dialog box. They are used in conjunction with the **DosFindFirst** and **DosFindNext** functions. These functions will not be discussed in this article.

Step 2 – Frame Setup (lines 126-132)

The first element of the **cs** array is defined to be the dialog box frame. These assignments position the dialog frame at the point 100,100 with a size of 180,100. The window ID for the frame is defined as 1000, and the text offset is defined to be 500 bytes. The text offset will be described in detail in Step 5.

Step 3 – Check box Setup (lines 137-157)

The remaining elements of the **cs** array are defined in this section. Line 137 allocates memory for the structure that is to be passed to the dialog procedure when it is created. This structure will contain the total number of files in the current directory. This number is assigned on line 157.

Figure 1a. Creating a Dialog Box

On line 138, a **DosFindFirst** function call is used to obtain a pointer to the first file in the directory. The next line marks the start of a **do{ .. }while( );** loop. This loop goes through all the files in the current directory using the **DosFindNext** function. The loop sets up the a new element in the **cs** array on every pass. The **cs** text field is loaded with the current file name. The **x** and **y** location is defined by using a simple algorithm; each check box will appear 10 pels below the previous one. The size of each check box is defined to be the same for every file. The window ID is set to 1000 + the pass number of the loop. The text offset, described in Step 5, is defined as 500 + the passnumber times 20.

Step 4 – Header Definition (lines 162-169)

At this time, the actual structures that **WinCreateDlg** expects to find will be filled out. The first structure is DLGTEMPLATE. The DLGTEMPLATE structure contains eight fields:

USHORT cbTemplate	**cbTemplate** defines the total number of bytes used by the dialog structures.
USHORT type	**type** must be set to zero.
USHORT codepage	**codepage** must be set correctly. The US keyboard requires a codepage = 850.
USHORT offadlgti	**offadlgti** is the number of bytes from the header structure that the array of control structures begins.
USHORT fsTemplateStatus	**fsTemplateStatus** must be set to zero.
USHORT iItemFocus	**iItemFocus** identifies the index of the control that will begin with the focus.
USHORT coffPresParams	**coffPresParams** defines the offset into the DataArea at which the presentation parameters exist.
DLGTITEM adlgti[1]	**adlgti** is a pointer to the first element of an array of DLGTITEM structures. Each DLGTITEM structure defines a single control in the dialog box.

The variable **dt** is a pointer to this structure type. On line 162, **dt** is set to the beginning of DataArea. The **cbTemplate** field is set to 1024 bytes. This number is larger than the actual amount of memory used to define the dialog box, but using a static number is simpler for example purposes. The remaining fields are set up to allow the system to use default values in the creation of the dialog box.

Step 5 – Frame Definition (lines 177-197)

This step defines the frame for the dialog box based on the first element of the **cs** array. The frame, like all of the dialog box controls, is defined in a DLGTITEM structure. This structure contains 14 fields:

USHORT fsItemStatus	**fsItemStatus** must be set to zero.
USHORT cChildren	**cChildren** is the number of controls existing under this control.
USHORT cchClassName	**cchClassName** should be set to zero for all system-defined classes.
USHORT offClassName	**offClassName** is the **WC_** window class for this control. For example, the frame would use **WC_FRAME**.
USHORT cchText	**cchText** is the length of the window text for this control.
USHORT offText	**offText** is the number of bytes into DataArea that the text for this control exists.
ULONG flStyle	**flStyle** is the creation window style for this control. It is usually a number of **WS_** and **FS_** constants or'd together.
SHORT x, y	**x** and **y** define the location of the control within the dialog box.
SHORT cx, cy	**cx** and **cy** define the size of the control.
USHORT id	**id** is the window ID of the control.
USHORT offPresParams	**offPresParams** is the number of bytes into DataArea that the presentation parameters for this control exist.

Figure 1b. Creating a Dialog Box

USHORT offCltData	**offCtlData** define the frame creation for the dialog box. For example, FCF_TITLEBAR.

Line 178 sets **cChildren** to **total+2**. The variable **total** was set to the number of files in the current directory. There is one check box for each file. There are also two pushbuttons, OK and Cancel. Therefore, the total number of controls under the dialog box frame is equal to the total number of check boxes plus the two pushbuttons: total+2.

The **offText** field defines the number of bytes into DataArea that the text for this control will appear. This offset must be beyond the array of DLGTITEM structures. The offset for the frame's window text was set to 500 in Step 2. The **WinCreateDlg** function then knows to look 500 bytes into DataArea to find the window text for this control. By examining the **cchText** field, the function knows how many characters to read from the offset.

Many fields in the DLGTITEM structure are defined by assigning the value contained in the related **cs** field. This was done to simplify the process of setting up the structures initially.

On lines 196 and 197, the actual window text is written to the proper offset by using **strcpy**.

Step 6 – Checkbox Definition (lines 203-221)
This step is simply a loop through all of the remaining **cs** elements. One **adlgti** element is defined for each **cs** element. The actual assignments are similar to Step 5, although there are some important differences.

First, notice that on line 205, the **cChildren** field is set to zero because check boxes have no controls under them. The **offClassName** is set to WC_CHECKBOX to identify the control for what it is. The rest of the **adlgti** fields follow a similar setup process to Step 5. Notice that **cTemp** is again used to write the window text at the proper offset into DataArea.

Step 7 – Pushbutton Definition (lines 226-260)
This step is simply an extension of the previous two. Here, the two pushbuttons, OK and Cancel, are defined. The setup is again very similar to those in Steps 5 and 6. Notice that on line 237, the ID field is set to DID_OK. This is so the dialog box procedure can intercept a DID_OK when the OK button is pressed. Line 255 is set to DID_CANCEL for the same reason.

Step 8 – Function Call (line 262)
Line 262 is the actual call to the **WinCreateDlg** function. The function is in the following form:

```
HWND APIENTRY WinCreateDlg(HWND hwndParent,
                           HWND hwndOwner,
                           PFNWP pfnDlgProc,
                           PDLGTEMPLATE pdlgt,
                           PVOID pCreateParams);
```

The fourth parameter is the pointer to the dialog template header structure, **dt**. The fifth parameter is a pointer to the user-defined data structure containing the number of files found in the current directory.

Figure 1c. Creating a Dialog Box

```
#define INCL_DOS
#define INCL_WIN
#include <os2.h>
#include <stdio.h>
#include <string.h>
#include <malloc.h>
/*************************************************************************************
Function Prototypes
*************************************************************************************/
MRESULT EXPENTRY DlgProc1(HWND, USHORT, MPARAM, MPARAM);
MRESULT EXPENTRY WinProc(HWND, USHORT, MPARAM, MPARAM);
PVOID MemAllocMemory(USHORT);
HWND check_dialog(HWND hwnd);

/*************************************************************************************
This is the data area in which the dialog box will be created.
*************************************************************************************/
BYTE DataArea[2048];                             /* Memory in which structures are placed     */

/*************************************************************************************
main
*************************************************************************************/
main()
{
   ULONG flags;
   HAB   hab;                                    /* Handle to anchor block                    */
   HMQ   hmq;                                    /* Handle to message queue                   */
   HWND  hwndFrame                               /* Main window frame                         */
         hwndClient;                             /* Main window client                        */
   QMSG qmsg;                                    /* Message queue                             */
   char primeclass[20];                          /* Class name of primary window              */

   strcpy(primeclass,"primeclass")
   flags=FCF_TITLEBAR | FCF_SYSMENU | FCF_SIZEBORDER | FCF_MINMAX |
   FCF_SHELLPOSITION | FCF_TASKLIST | FCF_MENU;
   hab=WinInitialize(0);
   hmq=WinCreateMsgQueue(hab,0);
   WinRegisterClass(hab, primeclass, WinProc, CS_SIZEREDRAW, 0);
   hwndFrame=WinCreateStdWindow(HWND_DESKTOP,WS_VISIBLE,&flags,primeclass,
               NULL,0l,NULL,256,&hwndClient);

   while(WinGetMsg(hab, &qmsg, NULL, 0, 0))
      WinDispatchMsg(hab, &qmsg);

   WinDestroyWindow(hwndFrame);
   WinDestroyMsgQueue(hmq);
   WinTerminate(hab);
}

/*************************************************************************************
DlgProc1 – This is the window procedure for the dialog box we have created. It is just like any other
dialog box procedure you might use for a resource-loaded dialog box.
*************************************************************************************/
MRESULT EXPENTRY DlgProc1(HWND hwnd, USHORT msg, MPARAM mp1, MPARAM mp2)
{
```

Figure 2a. Example Program

```
   typedef struct {
      int total;
   }_dialog;
   static _dialog *data;
   static char filename[20][13];
   int i;
   RECTL rcl;
   SWP swp;
   char buffer[81];

   switch(msg){
      case WM_INITDLG:
         data=(_dialog *)(MPFROMP(mp2));
         for(i=0;i<data->total;i++){
            WinQueryWindowText(WinWindowFromID(hwnd,1000+i+1),13,filename[i]);
         }
         break;
      case WM_COMMAND:
         switch(COMMANDMSG(&msg)->cmd){
            case DID_OK:
               for(i=0;i<data->total;i++){
                  if(WinSendDlgItemMsg(hwnd,1000+i+1,BM_QUERYCHECK,NULL,NULL)){
                     if(WinMessageBox(HWND_DESKTOP,hwnd,filename[i],
                                      "Delete File?",0,MB_OKCANCEL)==DID_OK){
                        unlink(filename[i]);
                     }
                  }
               }
               free (data);
               WinDismissDlg(hwnd,FALSE);
               return 0;
            case DID_CANCEL:
               WinDismissDlg(hwnd,FALSE);
               free (data);
               return 0;
         }
      default:
         return WinDefDlgProc(hwnd,msg,mp1,mp2);
   }
}

/********************************************************************************************
check_dialog - This is the function that actually creates the dialog box. It is called from the WinProc window
procedure whenever mouse button 2 is pressed. It sets up the necessary structures and makes the
WinCreateDlg function call. From here, the WM_INITDLG message is sent to the DlgProc1 function.
********************************************************************************************/
HWND check_dialog(HWND hwnd)
{
   DLGTEMPLATE *dt;                            /* Pointer to the DLGTEMPLATE structure   */
   CHAR *cTemp;                                /* Character pointer for string placement */
   HWND hwndDlg;                               /* Handle to the dialog box created       */
   struct _checkstruct{                        /* Check box information structure        */
      CHAR text[21];                           /* The text next to the check box         */
      POINTL pt;                               /* Position of text                       */
      SIZEL szl;                               /* Size of text rectangle                 */
```

Figure 2b. Example Program

```
        SHORT id;                              /* ID of text                          */
        SHORT offset;                          /* offset of text into memory          */
    }cs[20];
    SHORT i, total=1;                          /* Some counters                       */
    HDIR hdir = HDIR_CREATE;                   /* Used in DosFindFirst...             */
    USHORT usSearchCount=1;
    FILEFINDBUF findbuf;

    struct _dialog{
        int total;
    }*data;

/*****************************************************************************************
This is the first structure in the list. It represents the dimensions of the dialog box frame.
*****************************************************************************************/
    strcpy(cs[0].text,"Delete Boxes");         /* Title of dialog box                 */
    cs[0].pt.x=100;                            /* x coord from parent window          */
    cs[0].pt.y=100;                            /* y coord from parent window          */
    cs[0].szl.cx=180;                          /* width                               */
    cs[0].szl.cy=110;                          /* height                              */
    cs[0].id=1000;                             /* resource ID to associate to dialog  */
    cs[0].offset=500;                          /* offset of text into data area       */

/*****************************************************************************************
This section searches the current directory for all files and creates a structure array element for each file.
*****************************************************************************************/
    data=(struct _dialog *)malloc(sizeof(struct _dialog));
    DosFindFirst("*.*",&hdir,FILE_NORMAL,&findbuf,sizeof(findbuf),&usSearchCount,0l);
    do{
        strcpy(cs[total].text,findbuf.achName);

        if(total<9){
            cs[total].pt.x=5;
            cs[total].pt.y=100-(10*total);
        }
        else{
            cs[total].pt.x=85;
            cs[total].pt.y=100-(10*(total-8));
        }
        cs[total].szl.cx=80;
        cs[total].szl.cy=10;
        cs[total].id=1000+total;
        cs[total].offset=500+(total*20);
        total++;
    }while(!(DosFindNext(hdir,&findbuf,sizeof(findbuf),&usSearchCount)));
    total--;
    data->total=total;

/*****************************************************************************************
This section defines the dialog template header.
*****************************************************************************************/
    dt=(DLGTEMPLATE *)DataArea;
    dt->cbTemplate=1024;                       /* Size of this stuff (too big is okay) */
    dt->type=0;                                /* must be zero                        */
    dt->codepage=850;
```

Figure 2c. Example Program

```
   dt->offadlgti=14;
   dt->fsTemplateStatus=0;
   dt->iItemFocus=1;                         /* item number to start with focus         */
   dt->coffPresParams=0;                     /* must be zero                            */

/*****************************************************************************************
This section actually creates the array structure element for the dialog frame. The frame is always the first
element in the structure array. The second parameter must equal the number of dialog items that will follow.
The total variable is equal to the number of file names found. One check box will be created for each file
name located. Also, an OK button and a cancel button will be placed at the bottom of the dialog.
*****************************************************************************************/
   dt->adlgti[0].fsItemStatus=0;             /* must be zero                            */
   dt->adlgti[0].cChildren=total+2;          /* total check boxes and two buttons       */
   dt->adlgti[0].cchClassName=0;             /* 0 = system defined                      */
   dt->adlgti[0].offClassName=(USHORT)WC_FRAME;
   dt->adlgti[0].cchText=strlen(cs[0].text); /* length of text                          */
   dt->adlgti[0].offText=cs[0].offset;       /* offset to text                          */
   dt->adlgti[0].flStyle=FS_NOBYTEALIGN | FS_DLGBORDER | WS_VISIBLE | WS_SAVEBITS;
   dt->adlgti[0].x=cs[0].pt.x;               /* pos of dlg                              */
   dt->adlgti[0].y=cs[0].pt.y;
   dt->adlgti[0].cx=cs[0].szl.cx;            /* size of dlg                             */
   dt->adlgti[0].cy=cs[0].szl.cy;
   dt->adlgti[0].id=cs[0].id;
   dt->adlgti[0].offPresParams=0;
   dt->adlgti[0].offCtlData=(USHORT)FCF_TITLEBAR;

/*****************************************************************************************
This section assigns a pointer to the offset into the data area defined by the sixth parameter in the frame's
structure. The text for the title of the dialog box is placed at this offset by using the strcpy function.
*****************************************************************************************/
   cTemp=&DataArea[cs[0].offset];
   strcpy(cTemp, cs[0].text);

/*****************************************************************************************
This loop creates the array of structures needed to place the check boxes with the file names into the dialog
box. The second parameter for all of these items must be zero. Otherwise, it is similar to the frame's structure.
*****************************************************************************************/
for(i=1;i<total+1;i++){
   dt->adlgti[i].fsItemStatus=0;
   dt->adlgti[i].cChildren=0;
   dt->adlgti[i].cchClassName=0;
   dt->adlgti[i].offClassName=(USHORT)WC_BUTTON;
   dt->adlgti[i].cchText=strlen(cs[i].text);
   dt->adlgti[i].offText=cs[i].offset;
   dt->adlgti[i].flStyle=BS_AUTOCHECKBOX | WS_TABSTOP | WS_VISIBLE;
   dt->adlgti[i].x=cs[i].pt.x;
   dt->adlgti[i].y=cs[i].pt.y;
   dt->adlgti[i].cx=cs[i].szl.cx;
   dt->adlgti[i].cy=cs[i].szl.cy;
   dt->adlgti[i].id=cs[i].id;
   dt->adlgti[i].offPresParams=0;
   dt->adlgti[i].offCtlData=0;

   cTemp=&DataArea[cs[i].offset];
   strcpy(cTemp, cs[i].text);
```

Figure 2d. Example Program

```
}

/**********************************************************************************************
This section creates the OK pushbutton placed at the bottom of the screen.
**********************************************************************************************/
   dt->adlgti[total+1].fsItemStatus=0;
   dt->adlgti[total+1].cChildren=0;
   dt->adlgti[total+1].cchClassName=0;
   dt->adlgti[total+1].offClassName=(USHORT)WC_BUTTON;
   dt->adlgti[total+1].cchText=2;
   dt->adlgti[total+1].offText=cs[total].offset+20;
   dt->adlgti[total+1].flStyle=BS_PUSHBUTTON | WS_TABSTOP | WS_VISIBLE;
   dt->adlgti[total+1].x=50;
   dt->adlgti[total+1].y=5;
   dt->adlgti[total+1].cx=40;
   dt->adlgti[total+1].cy=15;
   dt->adlgti[total+1].id=DID_OK;
   dt->adlgti[total+1].offPresParams=0;
   dt->adlgti[total+1].offCtlData=0;

   cTemp=&DataArea[cs[total].offset+20];
   strcpy(cTemp,"OK");

   dt->adlgti[total+2].fsItemStatus=0;
   dt->adlgti[total+2].cChildren=0;
   dt->adlgti[total+2].cchClassName=0;
   dt->adlgti[total+2].offClassName=(USHORT)WC_BUTTON;
   dt->adlgti[total+2].cchText=6;
   dt->adlgti[total+2].offText=cs[total].offset+40;
   dt->adlgti[total+2].flStyle=BS_PUSHBUTTON | WS_TABSTOP | WS_VISIBLE;
   dt->adlgti[total+2].x=100;
   dt->adlgti[total+2].y=5;
   dt->adlgti[total+2].cx=40;
   dt->adlgti[total+2].cy=15;
   dt->adlgti[total+2].id=DID_CANCEL;
   dt->adlgti[total+2].offPresParams=0;
   dt->adlgti[total+2].offCtlData=0;

   cTemp=&DataArea[cs[total].offset+40];
   strcpy(cTemp,"Cancel");

   hwndDlg=WinCreateDlg(hwnd, hwnd, DlgProc1, dt, (long far*)data);
   return hwndDlg;
   }

/**********************************************************************************************
winproc — This is the primary window procedure.
**********************************************************************************************/
MRESULT EXPENTRY WinProc(HWND hwnd, USHORT msg, MPARAM mp1, MPARAM mp2)
{
   HPS hps;
   int index,i;
   static HWND hwndDlg;
   RECTL rcl;
   DLGTEMPLATE *dt;
```

Figure 2e. Example Program

```
    SEL sel;

    switch(msg){
       case WM_COMMAND:
          switch(COMMANDMSG(&msg)->cmd){
             case 258:
                hwndDlg=check_dialog(hwnd);
                break;
             case 259:
                WinSendMsg(hwnd,WM_CLOSE,0l,0l);
                break;
          }
          return 0;
       case WM_PAINT:
          hps=WinBeginPaint(hwnd,NULL,&rcl);
          WinFillRect(hps,&rcl,CLR_CYAN);
          WinEndPaint(hps);
          break;
       default:
          return WinDefWindowProc(hwnd,msg,mp1,mp2);
    }
 }
```

Figure 2f. Example Program

An Alternative for the OS/2 START Command

William J. Wen
Houston, Texas

START2.EXE is a utility program written in C. It fully utilizes all options of the DosStartSession API call for starting independent OS/2 sessions. This utility is similar to the OS/2 START command, but contains more options. This article demonstrates how to set up a program for various OS/2 API calls.

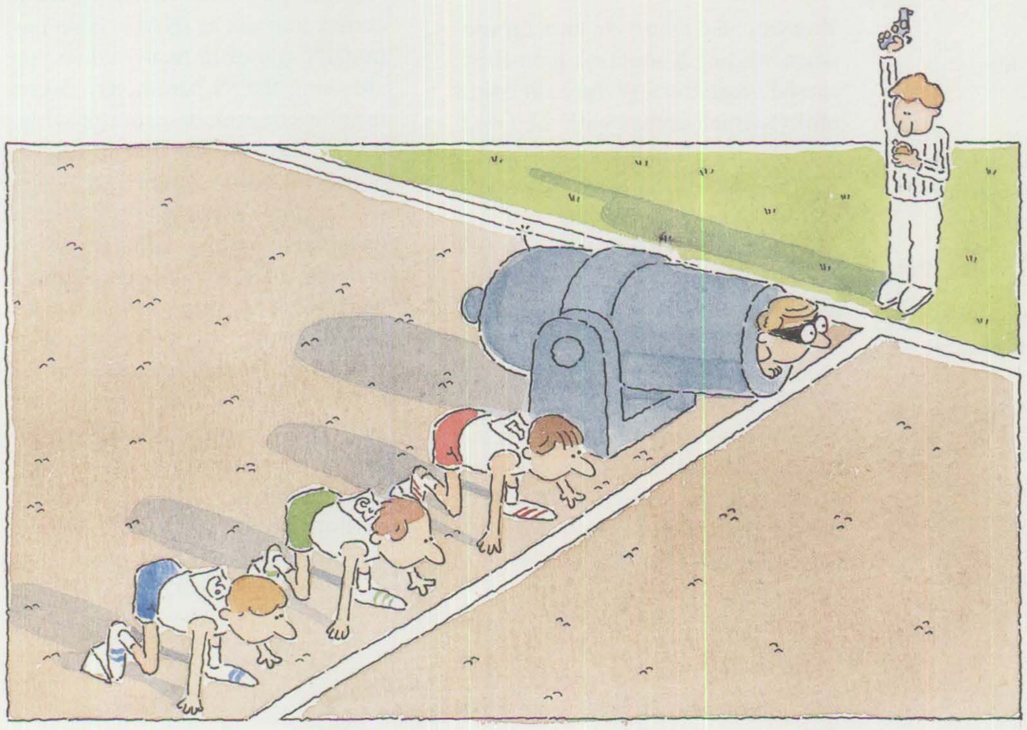
Have you ever wished that you could expand the flexibility of the OS/2 START command, giving it an even richer set of options than the robust set of parameters already available? For example, wouldn't it be nice if the OS/2 START command would let you associate a customized, default icon for specific programs? How about if you wanted to START an application in full-screen or windowed without having the session start in the foreground?

The OS/2 START command is an essential utility to start programs in separate OS/2 sessions. For instance, it allows the user to specify various options to control how the session is started, such as full-screen, windowed, or PM-based, to keep the OS/2 session after executing a command or exit immediately when the command is finished. You can provide all the functions given by the START command inside an OS/2 application by using the DosStartSession API call. If you compare the options given by the DosStartSession API call with those given by the OS/2 START command, you suddenly realize that the DosStartSession API call provides even more options and flexibility.

Take, for example, the case where you want to start a session explicitly in full-screen, windowed, or PM-based (that is, you specify /FS, /WIN, or /PM for the START command). The START command assumes that you want the session to begin in the foreground. There is not an option to specify /FS, /WIN, or /PM and have the session start in the background using the START command, though there is no such restriction when using the DosStartSession API call. How about starting a windowed session minimized or maximized? How about picking a distinctive icon for a full-screen or windowed application instead of the icon with the "OS/2" symbol?

START Command Options

Because the DosStartSession API call provides an even more abundant set of options than those provided by the OS/2 START command, I decided to write my own version of this command. I call it START2. I wrote this utility using IBM C/2™ Version 1.1 and the OS/2 Toolkit Version 1.1. Because I had some trouble setting up the various OS/2 API calls, I have included the source code to this utility program in hope that this code



will help you in your own ventures into OS/2 programming.

All the options of the START command are included in START2. However, two of the parameters are different: The program title and the /F parameter.

Because of a command line parsing problem (as far as I can determine, the OS/2 program loader strips off the double quote character from the command line, so no application can detect the double quote character in the command line), the program title enclosed in double quotes must include a space or tab. For example, if the program title is two words or more (for example, "Communications Manager"), then the space is already included. If the program title is a single word, then you can include a space preceding the actual title; if the first character in the program title is a space, then the START2 command line parsing routine will skip over the first space in the program title. For example, to display a program title of "Demonstration", you would specify

```
" DEMONSTRATION"
```

on the command line for the START2 program.

The /F parameter must be specified if you want the session to start in the foreground. This is different from the OS/2 START command, as specifying /FS, /WIN, or /PM with the START command will start the session in the foreground.

Additional Options Under START2.EXE

The help text is included in the START2 program. The help text is displayed either when the user asks for help or when a parameter is specified incorrectly. To ask for help from START2, type in any one of the following four:

```
START2
START2 ?
START2 /?
START2 /help
```

In the help text, I tried to implement all the options that apply to starting an independent session. I did not include any of the options that apply to a dependent session; if you need this level of control to start a session, you would probably want to code to the DosStartSession API call inside your own application instead of shelling out of your program to execute START2.

START2 gives the user two ways to specify the icon file.

You can also associate an icon file when starting a session. Icon files can be created using the icon editor utility, which comes with the OS/2 Version 1.1 Toolkit package. START2 gives the user two ways to specify the icon file. One way is for the user to specify the fully qualified filename of the icon file, by using the /ICON=filename parameter. The other way is for the user to specify the /ICON parameter, and START2 will look for the icon file. START2 will assume that the icon file has the .ICO file extension and is stored in the same directory as the program file (the program file has either .EXE or .CMD file extension).

The criteria that START2 uses to find the icon file are fairly close to the criteria that OS/2 uses to find the program file (.EXE or .CMD): If path information is included in the filename (for example, D:PROG1, E:\DIR1\PROG1, DIR1\PROG1, and so forth), START2 will look for the file in the specified drive and/or directory; if no path information is included on the filename, START2 will look for the file using the PATH= environment string. If no file extension is specified, START2 will try finding the file with .EXE extension first and, if not found, try finding the file with .CMD extension. Once the file is found, START2 replaces the file extension with .ICO.

Please be aware that only when you use the /ICON parameter will the icon file need to have a .ICO file extension and be stored in the same directory as the program file. If /ICON = filename is specified, then the icon file can reside on any disk in any directory with any file extension.

Another parameter that occasionally comes into use is /NAC. This parameter comes in handy when you are using the /N parameter. Sometimes a program does not provide a pause, allowing the user to read the output on the screen before closing the session. If the user starts such a program using the /N parameter, the program runs, displays its output to the screen, and exits (which by default automatically closes the session); in the meantime, the user gets but a glimpse of the output. Using the /NAC option will keep the session after the program has exited; the user has to manually close any session started with /NAC parameter, either through the system menu or the task manager.

Known Inconsistency

If you specify starting a windowed session in the foreground, minimized and with a specific icon file (the /F, /ICON or /ICON=filename, and /MIN all used together), the icon you selected will not display until you restore and then minimize the session again. As far as I can tell, this seems to be a characteristic of the DosStartSession API. If you specify starting a windowed session in the background, minimized with a specific icon file (the /ICON or /ICON=filename, with the /MIN used together), the icon is displayed correctly.

Examples For Using START2

If you want to start the program PROG1.CMD with its own default icon file, then type:

```
START2 /ICON PROG1
```

If you want to start the program PROG2.EXE in the directory C:\DIR1 as a windowed session without loading the command processor, using an icon file in the C:\DIR2 directory with the filename of MYICON2.ICO, then type:

```
START2 /N /WIN
/ICON=C:\DIR2\MYICON2.ICO
C:\DIR1\PROG2.EXE
```

If you want to start PROG2.EXE in the C:\DIR1 directory as a windowed session without loading the command processor, and position it on the lower left corner, covering approximately a fourth of the screen, then type:

```
START2 /N /WIN /POS=(0,0)
/SIZ=(300,300)
C:\DIR1\PROG2.EXE
```

If you would like, you can also specify the icon file to the previous example. To specify starting a session with the command processor and start PROG1.CMD with the default icon file and the program title of "My program", type:

```
START2 "My program" /ICON
PROG1
```

To start the same program as the previous example but with the program title of "Program1", type:

```
START2 " Program1" /ICON PROG1
```

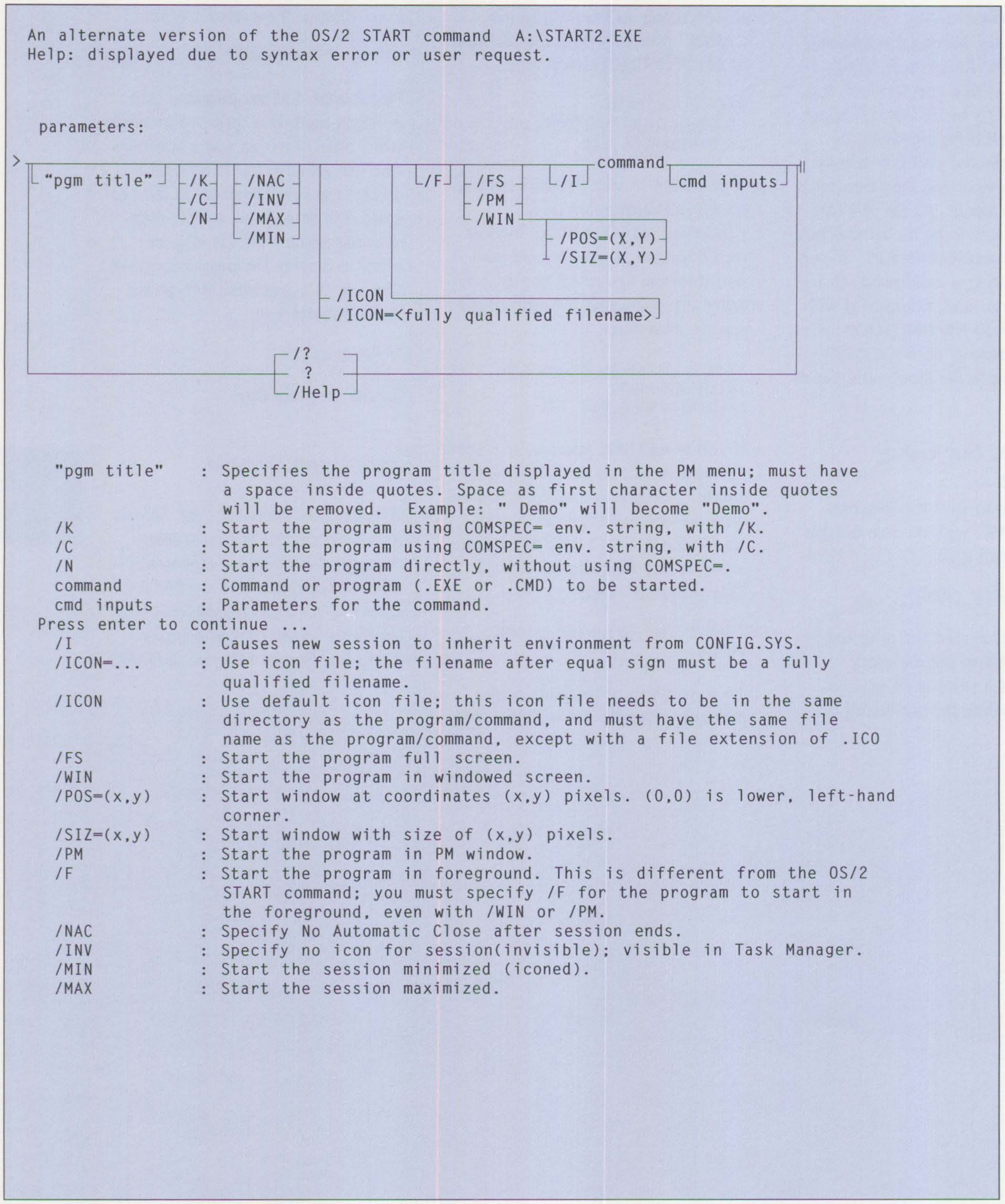
Please note that the program title needs to include a space. For single-word titles, you can add a space before the actual title. If START2 detects the first character in the program title as a space, it will skip over the space. This is why in order to display the program title of "Program1", you need to type on the command line:

```
" Program1"
```

for the program title.

ABOUT THE AUTHOR

William Wen received a bachelor's degree in electrical engineering from the University of Houston, Central Campus. Bill worked for six co-op terms as a supplemental employee in the IBM Marketing Technical Support Center in Dallas.



```
An alternate version of the OS/2 START command   A:\START2.EXE
Help: displayed due to syntax error or user request.


 parameters:

>─┬───────────────┬┬────┬┬───────┬┬──────────────────┬┬────┬┬───────┬┬────┬─command─┬────────────┬─┤├
  └─"pgm title"───┘├─/K─┤├─/NAC──┤│                  ││    │└─/F─┘├─/FS───┤└─/I─┘          └─cmd inputs─┘ │
  │               ├─/C─┤├─/INV──┤│                  ││    │      ├─/PM───┤                                │
  │               └─/N─┘├─/MAX──┤│                  ││    │      └─/WIN─┬──────────────┬─┘                │
  │                     └─/MIN──┘│                  ││    │             ├─/POS=(X,Y)─┤                   │
  │                              │                  ││    │             └─/SIZ=(X,Y)─┘                   │
  │                              ├─/ICON────────────┘│    │                                                │
  │                              └─/ICON=<fully qualified filename>─┘                                      │
  │                                                                                                        │
  │                         ┌─/?────┐                                                                      │
  └─────────────────────────┼─ ? ───┼──────────────────────────────────────────────────────────────────────┘
                            └─/Help─┘


   "pgm title"     : Specifies the program title displayed in the PM menu; must have
                     a space inside quotes. Space as first character inside quotes
                     will be removed.  Example: " Demo" will become "Demo".
   /K              : Start the program using COMSPEC= env. string, with /K.
   /C              : Start the program using COMSPEC= env. string, with /C.
   /N              : Start the program directly, without using COMSPEC=.
   command         : Command or program (.EXE or .CMD) to be started.
   cmd inputs      : Parameters for the command.
 Press enter to continue ...
   /I              : Causes new session to inherit environment from CONFIG.SYS.
   /ICON=...       : Use icon file; the filename after equal sign must be a fully
                     qualified filename.
   /ICON           : Use default icon file; this icon file needs to be in the same
                     directory as the program/command, and must have the same file
                     name as the program/command, except with a file extension of .ICO
   /FS             : Start the program full screen.
   /WIN            : Start the program in windowed screen.
   /POS=(x,y)      : Start window at coordinates (x,y) pixels. (0,0) is lower, left-hand
                     corner.
   /SIZ=(x,y)      : Start window with size of (x,y) pixels.
   /PM             : Start the program in PM window.
   /F              : Start the program in foreground. This is different from the OS/2
                     START command; you must specify /F for the program to start in
                     the foreground, even with /WIN or /PM.
   /NAC            : Specify No Automatic Close after session ends.
   /INV            : Specify no icon for session(invisible); visible in Task Manager.
   /MIN            : Start the session minimized (iconed).
   /MAX            : Start the session maximized.
```

Figure 1a. Help Text from the Start2 Program

```
#define LINT_ARGS
#define INCL_DOS
#include <stdio.h>
#include <stdlib.h>
#include <process.h>
#include <string.h>
#include <os2.h>
#include <dos.h>
#include <ctype.h>
#define NO_COMSPEC   1
#define DEF_ICON     2

extern char far *_pgmptr;

USHORT SessID;
USHORT ProcessID;
STARTDATA StartD;

char CurDir[80];
char PT[60] = "";
char PN[60] = "";
char parms[60] = "";
char IcF[60] = "";
char buffer[255], buffer2[80];
char *help_text[] = {

 "parameters:  ",

">─┬──────────────────┬┬──────┬┬────────┬┬──────────────────────────────┬┬──────┬┬───────┬┬─────┬─command ┬──────────────┬─\n",
" │└\"pgm title\"┘├─/K ─┤├─/NAC ─┤│                              │└─/F ─┘├─/FS ──┤└─/I ─┘          └cmd inputs ┘ \n",
" │                  ├─/C ─┤├─/INV ─┤│                              │       ├─/PM ──┤                                 \n",
" │                  └─/N ─┘├─/MAX ─┤│                              │       └─/WIN ─┴──────────┐                      \n",
" │                         └─/MIN ─┘│                              │              ├─/POS=(x,y) ─┤                      \n",
" │                                   │                              │              └─/SIZ=(x,y) ─┘                      \n",
" │                                   ├─/ICON ────────────────────────┘                                                \n",
" │                                   └─/ICON=<fully qualified filename>┘                                             \n",
" │                  ┌─ /?    ─┐                                                                                      \n",
" └──────────────────┼─ ?     ─┼──────────────────────────────────────────────────────────────────────────────────┘  \n",
"                    └─ /Help ─┘\n\n",


   "\"pgm title\"    : Specifies the program title displayed in the PM menu; MUST have\n",
                    " a space inside quotes. Space as first character inside quotes\n",
                    " will be removed.  EX: \" Demo\" will become \"Demo\".\n",
   " /K             : Start the program using COMSPEC= env. string, with /K.\n",
   " /C             : Start the program using COMSPEC= env. string, with /C.\n",
   " /N             : Start the program directly, without using COMSPEC=.\n",
   " command        : Command or program (.EXE or .CMD) to be started.\n",
   " cmd inputs     : Parameters for the command.",
   " /I             : Causes new session to inherit environment from CONFIG.SYS.\n",
   " /ICON=...      : Use icon file; the filename after equal sign must be a fully\n",
                      " qualified filename.\n",
   " /ICON          : Use default icon file; this icon file needs to be in the same\n",
                    " directory as the program/command, and must have the same file\n",
                    " name as the program/command, except with a file extension of .ICO",
```

Figure 1b. Start2 Program

```
   " /FS              : Start the program full screen.\n",
   " /WIN             : Start the program in windowed screen.\n",
   " /POS=(x,y)       : Start window at coordinates (x,y) pixels. (0,0) is lower, left -\n",
                        " hand corner.\n",
   " /SIZ=(x,y)       : Start window with size of (x,y) pixels.\n",
   " /PM              : Start the program in PM window.\n",
   " /F               : Start the program in foreground. This is different from the OS/2\n",
                        " START command; you MUST specify /F for the program to start in\n",
                      " the foreground, even with /WIN or /PM.\n",
   " /NAC             : Specify No Automatic Close after session ends.\n",
   " /INV             : Specify no icon for session(invisible); visible in Task Manager.\n",
   " /MIN             : Start the session minimized (iconed).\n",
   " /MAX             : Start the session maximized.\n",
   "",
   };

FileError(char *);
void parsename(char *, char *);

/*****************************************************************************************************
* START2.C                                                                                           *
* By William J. Wen                                                                                  *
*                                                                                                    *
* This utility program provides all the options under the DosStartSession API call, except for those options related *
* to starting dependent sessions. The help text, which explains all the options available, is shown by typing *
* one of the following:                                                                              *
*                                   START2/                                                          *
*                                   START2 /?                                                        *
*                                   START2 ?                                                         *
*                                   START2 /help                                                     *
*****************************************************************************************************/
   main(argc, argv)
      int argc;
      char *argv[];
   {
      char *ptemp;
      int count=1;
      USHORT Options = 0, size;
      HDIR dir1;
      FILEFINDBUF info;

      printf("\nAn alternate version of the OS/2 START command = ");
      while(*_pgmptr) putchar(*_pgmptr++);
      printf("\n    By William J. Wen\n\n");

/*****************************************************************************************************
* Display help when requested                                                                        *
*****************************************************************************************************/
      if (argc==1) help();
      else if (*argv[1]=='?') help();
      else if (!strcmp(argv[1], "/?")) help();
      else if (!strcmpi(argv[1], "/help")) help();

      init();

/*****************************************************************************************************
* Pick up program title, if it exists                                                                *
*****************************************************************************************************/
```

Figure 1c. Start2 Program

```
if (*argv[1]!='/') {
   if (strpbrk(argv[1], " \t")!=NULL) {                     /* program name */
      count++;
      StartD.PgmTitle = PT;
      if((*argv[1]==' ') || (*argv[1]=='\"')) strcpy(PT, (argv[1]+1));
      else strcpy(PT, argv[1]);
      if(*(PT+strlen(PT)-1)=='\"') *(PT+strlen(PT)-1) = '\0';
   }/* endif */
}/* endif */
if (count==argc) help();

/**************************************************************************************************
* Loop until there is no "/" parameter left                                                       *
**************************************************************************************************/
for(; COUNT<argc; count++) {
   if(*argv[count]!='/') break;
   strupr(argv[count]);
   ptemp = argv[count]+2;
   switch (*(ptemp-1)) {
   case 'N':
      if (*ptemp) {
         if(strcmp(ptemp, "AC")) help();
         else StartD.PgmControl |= 8;
   }  else {
         Options |= NO_COMSPEC;
         parms[0] = 0;
   }  /* endif */
      break;
   case 'K':
      if (*ptemp) help();
      else {
         strcpy(parms, "/K");
         Options &= ~NO_COMSPEC;
}
   break;
case 'C':
   if (*ptemp) help();
   else {
      strcpy(parms, "/C");
      Options &= ~NO_COMSPEC;
}
   break;
case 'F':
   if (*ptemp) {
      if(strcmp(ptemp, "S")) help();
      else StartD.SessionType = 1;
   } else StartD.FgBg = 0;
   break;
case 'P':
   if (*ptemp) {
      if (strcmp(ptemp, "M")) {
         if (ptemp==strstr(ptemp, "OS")) {
            StartD.PgmControl |= 0x8000;
            if((ptemp=strchr(ptemp, '('))==NULL) help();
            if(sscanf(++ptemp, "%d,%d", &StartD.InitXPos,
               &StartD.InitYPos) != 2) help();
      } else help();                                    /* not "/POS..." */
   }  else StartD.SessionType = 3;                      /* "/PM" */
   } else help();                                       /* "/P" not a valid parameter */
```

Figure 1d. Start2 Program

```
   break;
case 'W':
   if (*ptemp) {
      if(ptemp!=strstr(ptemp, "IN")) help();
      else StartD.SessionType = 2;
   } else help();
   break;
case 'I':
   if (*ptemp) {
      if (ptemp==strstr(ptemp, "CON")) {
         ptemp = ptemp + 3;
         StartD.IconFile = IcF;
         switch (*ptemp) {
         case '\0':
            Options |= DEF_ICON;
            IcF[0] = 0;
            break;
         case '=':
            strcpy(IcF, ++ptemp);
            Options &= ~DEF_ICON;
            break;
      default:
         help();
      }                                            /* endswitch */
      } else if (ptemp==strstr(ptemp, "NV")) StartD.PgmControl |= 1;
      else help();
} else StartD.InheritOpt = 0;
   break;
case 'M':
   if (*ptemp) {
      if(ptemp==strstr(ptemp, "IN")) StartD.PgmControl |= 4;
      else if(ptemp==strstr(ptemp, "AX")) StartD.PgmControl |= 2;
      else help();
   }else help();
   break;
   case 'S':
      if (*ptemp) {
         if (ptemp==strstr(ptemp, "IZ")) {
            StartD.PgmControl |= 0x8000;
            if((ptemp=strchr(ptemp, '('))==NULL) help();
            if(sscanf(++ptemp, "%d,%d", &StartD.InitXSize,
               &StartD.InitYSize)!=2) help();
   }     else help();                                /* not "/SIZ.."  */
      } else help();                                 /* "/S" invalid  */
      break;
   default:
      help();
   }                                                 /* endswitch */
   }                                                 /* endfor */

if (count==argc) help();
   ptemp = argv[count]; strupr(ptemp);

/*****************************************************************************************************
* Looks for icon file if /ICON is specified                                                          *
*****************************************************************************************************/
   if (Options & DEF_ICON) {
      if (Options & NO_COMSPEC) {
         strcpy(IcF, argv[count]);
```

Figure 1e. Start2 Program

```
         if(ptemp = strrchr(IcF, '.')) strcpy(ptemp, ".ICO");
         else FileError(argv[count]);
      } else {
         parsename(buffer, argv[count]);
         if( ((buffer+strlen(buffer)-4)==strstr(buffer, ".EXE")) ||
            ((buffer+strlen(buffer)-4)==strstr(buffer, ".CMD")) ) {
            dir1 = (HDIR) 1;
            size = 1;
            if(!DosFindFirst(buffer, &dir1, 0, &info, sizeof(info),
            &size, 0L)) {
               strcpy (IcF, buffer);
               ptemp = strrchr(IcF, '.');
               strcpy (ptemp, ".ICO");
            } else FileError (buffer);

}else if (strchr(buffer, '.')==NULL) {                  /* No file extension */
      strcat (buffer, ".EXE");                          /* try .EXE for extension */
         dir1 = (HDIR) 1;
         &size =1;
         if (DosFindFirst(buffer, &dir1, 0, &info, sizeof(info),
         &size, 0L)) {
            ptemp = strchr(buffer, '.');
            strcpy(++ptemp, "CMD");                     /* try .CMD for extension */
            dir1 = (HDIR) 1;
            size = 1;
            if(DosFindFirst(buffer, &dir1, 0, &info, sizeof(info),
            &size, 0L)) {
               printf("\nTry to find file via PATH= environment string.\n");
            } else {
               strcpy(IcF, buffer);
               ptemp = strrchr(IcF, '.');
               strcpy(ptemp, ".ICO");
   }
   } else {
      strcpy(IcF, buffer);
      ptemp = strrchr(IcF, '.');
      strcpy(ptemp, ".ICO");
      }
   }

   if (!IcF[0]) {                                       /* Still haven't found file */
      strcpy(buffer2, argv[count]);
      ptemp = buffer2;
      if( ((ptemp+strlen(ptemp)-4)==strstr(ptemp, ".EXE")) ||
         ((ptemp+strlen(ptemp)-4)==strstr(ptemp, ".CMD")) ) {
         if(!DosSearchPath(3, "PATH", ptemp, buffer, sizeof(buffer))){
               strcpy(IcF, buffer);
            ptemp = strrchr(IcF, '.');
            strcpy(ptemp, ".ICO");
         } else FileError(argv[count]);
      } else if (strchr(ptemp, '.')==NULL) {
      strcat(ptemp, ".EXE");
      if (DosSearchPath(3, "PATH", ptemp, buffer, sizeof(buffer))) {
         ptemp = strchr(argv[count], '.');
         strcpy(++ptemp, "CMD");
         if(DosSearchPath(3, "PATH", argv[count], buffer,
         sizeof(buffer))) FileError(argv[count]);
         else {
            strcpy(IcF, buffer);
            ptemp = strrchr(IcF, '.');
```

Figure 1f. Start2 Program

```
            strcpy(ptemp, ".ICO");
       }
       } else {
          strcpy(IcF, buffer);
          ptemp = strrchr(IcF, '.');
          strcpy(ptemp, ".ICO");
          }
       }
       }                                                /* endif */
    }                                                   /* endif */
 }                                                      /* endif */

/****************************************************************************************************
* Clean up for the /N parameter                                                                     *
****************************************************************************************************/
if (Options&NO_COMSPEC) {
   strcpy(PN, argv[count++]);
} else {
   if(!strlen(parms)) strcpy(parms, "/K");
} /* endif */

for (; count<argc; count++) {
   strcat(parms, " ");
   strcat(parms, argv[count]);
} /* endfor */

if (StartD.PgmTitle) printf("\nProgram title    = %s", PT);
if (StartD.PgmName)  printf("\nProgram name     = %s", PN);
                     printf("\nParameters       = %s", parms);
if (StartD.IconFile) printf("\nIcon file name   = %s", IcF);
printf("\nReturn code     %u\n", DosStartSession(&StartD, &SessID,
&ProcessID));
}

/****************************************************************************************************
* Function to initialize DosStartSession data structure to defaults. The defaults set in this function are the same *
* ones set in the OS/2 START command.                                                               *
****************************************************************************************************/
init()
{
StartD.Length = 50;                                     /* data structure length*/
StartD.Related = 0;                                     /* independent process */
StartD.TraceOpt = 0;                                    /* trace option off */
StartD.TermQ = NULL;
StartD.Environment = NULL;
StartD.PgmHandle = 0;
StartD.PgmTitle = NULL;                                 /* program title */
StartD.PgmName = PN;
strcpy(PN, getenv("COMSPEC"));
StartD.PgmInputs = parms;
StartD.IconFile = NULL;                                 /* icon file */
StartD.FgBg = 1;                                        /* background */
StartD.InheritOpt = 1;                                  /* inherit from shell proc */
StartD.SessionType = 1;                                 /* screen set by shell */
StartD.PgmControl = 0x0000;
StartD.InitXPos = 50;
StartD.InitYPos = 40;

StartD.InitXSize = 200;
```

Figure 1h. Start2 Program

```
StartD.InitYSize = 400;
}

/*****************************************************************************************************
* Generic file error function. Function does little more than display the error number.             *
*****************************************************************************************************/
FileError(s)
char *s;
{
   int i=0;
   printf("\nError looking for default icon file of: %s\nBuffer is: ", s);
   while( (i < sizeof(buffer)) && (buffer[i]!='\0') ) putchar(buffer[i++]);
   pause();
   exit(1);
}

/*****************************************************************************************************
* Pause function. Handy during troubleshooting                                                      *
*****************************************************************************************************/
pause()
{
   printf("\n    Press enter to continue ...");
   getchar();
}

/*****************************************************************************************************
* Function to display help                                                                          *
*****************************************************************************************************/
help()
{
int i;
printf("Help: displayed due to syntax error or user request.\n");
for (i=0; *help_text[i]; i++) {
   if(strstr(help_text[i], "/I ")!=NULL) pause();
   printf("%s", help_text[i]);
} /* endfor */
exit(1);
}

/*****************************************************************************************************
* Function to parse the file name from the command line. I could not find a function to return a fully qualified   *
* file name from different combinations of what the user might want to type on the command line, so I had to       *
* write the function. For example, assume that a system has two logical drives, C and D. The current directory is  *
* \CDIR1\CDIR2 for the C: drive, and \DDIR1\DDIR2 on D: drive. There is a CDIR3 subdirectory in the                *
* \CDIR1\CDIR2 directory, and a DDIR3 subdirectory in the \DDIR1\DDIR2 directory. Assume that the current          *
* drive is C:. Assume that each directory has programs that a user might want run; programs named as follows:      *
* PROGC1.EXE in C:\CDIR1 directory, PROGC2.EXE in C:\CDIR1\CDIR2 directory, PROGD3.EXE in                           *
* D:\DDIR1\DDIR2\DDIR3 directory, and so on.                                                                        *
*                                                                                                                   *
* If the user types in:          PROGC2                                                                             *
* the function returns:          C:\CDIR1\CDIR2\PROGC2                                                              *
*                                                                                                                   *
* If the user types in:          CDIR3\PROGC3                                                                       *
* the function returns:          C:\CDIR1\CDIR2\CDIR3\PROGC3                                                        *
*                                                                                                                   *
* If the user types in:          \CDIR1\PROGC1                                                                      *
```

Figure 1i. Start2 Program

```
* the function returns:          C:\CDIR1\PROGC1                                    *
*                                                                                   *
* If the user types in:          ..\PROGC1                                          *
* the function returns:          C:\CDIR1\PROGC1                                    *
*                                                                                   *
* If the user types in:          D:DDIR3\..\..\PROGD1                               *
* the function returns:          D:\DDIR1\PROGD1                                    *
*************************************************************************************/
void parsename(out, in)
char *in, *out;
{
   char *ptr1;
   USHORT drv, path_len;
   ULONG drv_map;
if (*(in+1)==':' && isalpha(*in)) {
   *out = toupper(*in++);
   drv = *out - 'A' + 1;
   in++;
} else {
      DosQCurDisk(&drv, &drv_map);
      *out = drv + 'A' - 1;
   } /* endif */
   strcpy(out+1, ":");
   if (*in=='\\') in++;
   else {
      DosQCurDir(drv, CurDir, &path_len);
      *(CurDir+path_len) = '\0';
      strcat(out, "\\");
      strcat(out, CurDir);
      }                                              /* endif */
   while (*in) {
      if (*in == '.') {
         if (*++in == '.') {
            if (ptr1=strrchr(out, '\\')) {
               *ptr1 = '\0';
            } else {
               printf("\nCannot chdir back past the root.\n");
               exit(1);
            }                                        /* endif */
            in++;
         }                                           /* endif */
         in++;
      } else {
         ptr1 = out + strlen(out);
         *ptr1++ = '\\';
         while (*in && *in!='\\') *ptr1++ = *in++;
         *ptr1 = '\0';
         if (*in=='\\') in++;
      }                                              /* endif */
   }                                                  /* endwhile */
}
```

Figure 1j. Start2 Program

CUA: A Consistent Interface

Jennifer L. Wilson
IBM Corporation
Boca Raton, Florida

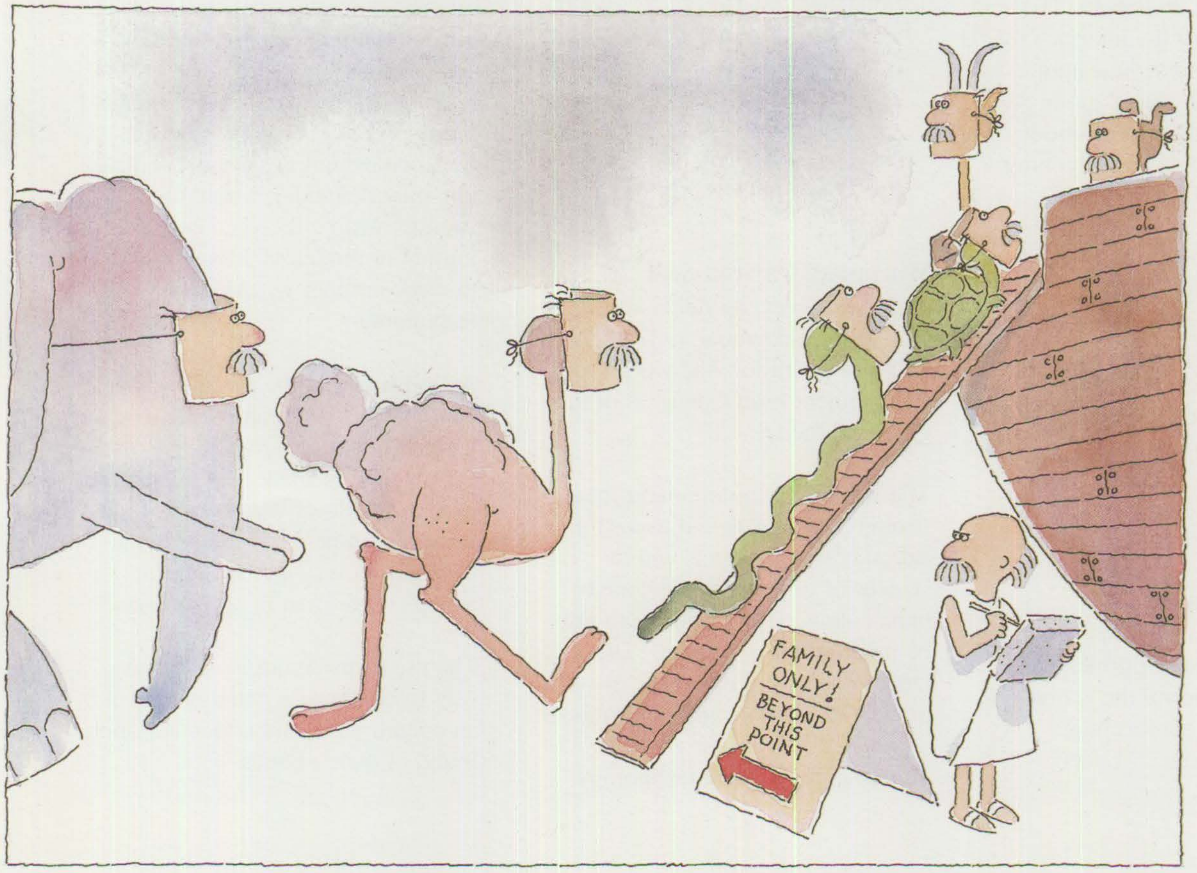
Common User Access (CUA), which is one section of Systems Application Architecture™ (SAA™), provides a consistent user interface for application developers. The current version of CUA defines models and design principles for high-level design. For detailed design, individual components are defined. When designing applications, basic CUA principles should be followed, which will produce an application interface that is easy to use and learn.

With the large number of applications available today, application developers must find as many ways as possible to get the competitive edge for their products. One way is to design applications using the Common User Access (CUA) interface. With CUA, applications are easier to use and provide a consistent interface with other applications and the operating system. Tools for developing CUA applications are available to shorten your development time.

SAA and CUA

Systems Application Architecture (SAA) is a set of guidelines that allows applications to be more consistent. CUA is the part of SAA that describes the panel display and the user interaction of a product. SAA and CUA apply to both the programmable workstation environment and to nonprogrammable terminal interfaces.

By designing applications using CUA, applications are easier to learn. CUA let application users focus on the actual work they are trying to do, rather than on the interface being used. Rather than focusing on which keys to press to use an



editor, for example, the focus is on the document being edited.

It is also easier for users to switch from one CUA application to another without having to learn a new interface. This is very useful, especially for OS/2 applications, because applications can be windowed on the same screen. Consistency within the product itself is also a benefit of CUA.

CUA Evolution

The original definition of CUA was published in December 1987 as *SAA, CUA: Panel Design and User Interaction* (SC26-4351). Two sets of rules were defined depending on the application type: one for graphics and the other for text.

This manual was replaced in June 1989 by the *SAA, CUA: Basic Interface Design Guide* (SC26-4583) and the *SAA, CUA: Advanced Interface Design Guide* (SC26-4582). The basic guide defines the interface for nonprogrammable terminal applications, while the advanced guide defines the interface for workstation applications (such as those running on OS/2).

The guides define CUA at three levels of detail. First, the basic design principles are defined. Next, models for high-level guidelines are described; and finally, detailed descriptions of the use of CUA components are given.

The principles, models, and rules are for generic rather than specific application types, such as word processing or accounting.

The basic CUA design principles are to let users control the action and to encourage a conceptual model of the interface. Users should be able to continue when they run into problems, and should find that the interface is similar to others they use.

CUA Models

Entry and graphical are the two basic CUA models. The entry model can be used for nonprogrammable terminals, while the graphical model is for programmable workstations.

Both models define the display and interaction of applications. An object-action philosophy is used, where, for example, files (objects) are selected, and the delete function (action) is chosen and performed.

System pull-downs include options to size, restore, move, minimize, and maximize the window.

Workplace Environment

As the graphical model is an extension of the December 1987 CUA graphics definition, the workplace environment is an extension of the graphical model.

The workplace environment focuses strongly on objects and lists. Interaction between objects and instances of objects can be done by mouse drag. Connections can also be made between objects. The OfficeVision™ product has many of the features of the workplace model.

As computer use and technology continue to change, CUA will evolve to meet the future needs.

CUA Components

The various components of a programmable terminal application interface are defined by the graphical model CUA rules. The components include the basic display (Figure 1) with windows and dialog boxes, and the contents of the display used to implement the interface. Display contents can include the action bar and pull-downs, pushbuttons, help, scroll bars, and many types of lists.

Figure 1 shows the location and appearance of some CUA components. For more details, use the CUA design guides.

A window, and its ability to be sized and moved, is defined by CUA. Borders, system pull-downs, and window-sizing icons are defined. System pull-downs include options to size, restore, move, minimize, and maximize the window.

A dialog box (also sometimes called a pop-up box) is used for interactions requiring a response. A dialog box may be modal, where the response must be received before continuing or modeless, allowing users to work in other windows before responding.

Action bars and the pull-downs associated with the action bar items contain the operations to perform on the objects selected. These bars are used in windows where various operations can be performed. Standard pull-downs are defined for actions used often in applications.

The use of pushbuttons is an easy way to complete a panel or cancel an operation. Pushbuttons are often found in dialog boxes.

Scroll bars are often used when long lists of items are displayed. The scroll bar represents the entire list. The slider box represents the proportional size and position within the list of currently displayed items.

Help is an important part of an application. Having help information available for each item on the screen is called "contextual help." Other help options are available from the help pull-down, which can also be selected from the help panel itself.

Many controls are defined to make selection easier. Controls used to make selections from lists are radio buttons, check boxes, list boxes, combination boxes, drop-down lists, drop-down combination boxes, spin buttons, and value sets.

Depending on the control, users may select one or any number of items, choose from selections, or turn items off or on. Radio buttons are used when only one selection can be made. Check boxes are used if choices are on or off. The list, combination, drop-down list, drop-down combination boxes and the spin button, are all controls used to select an item from a list of items.

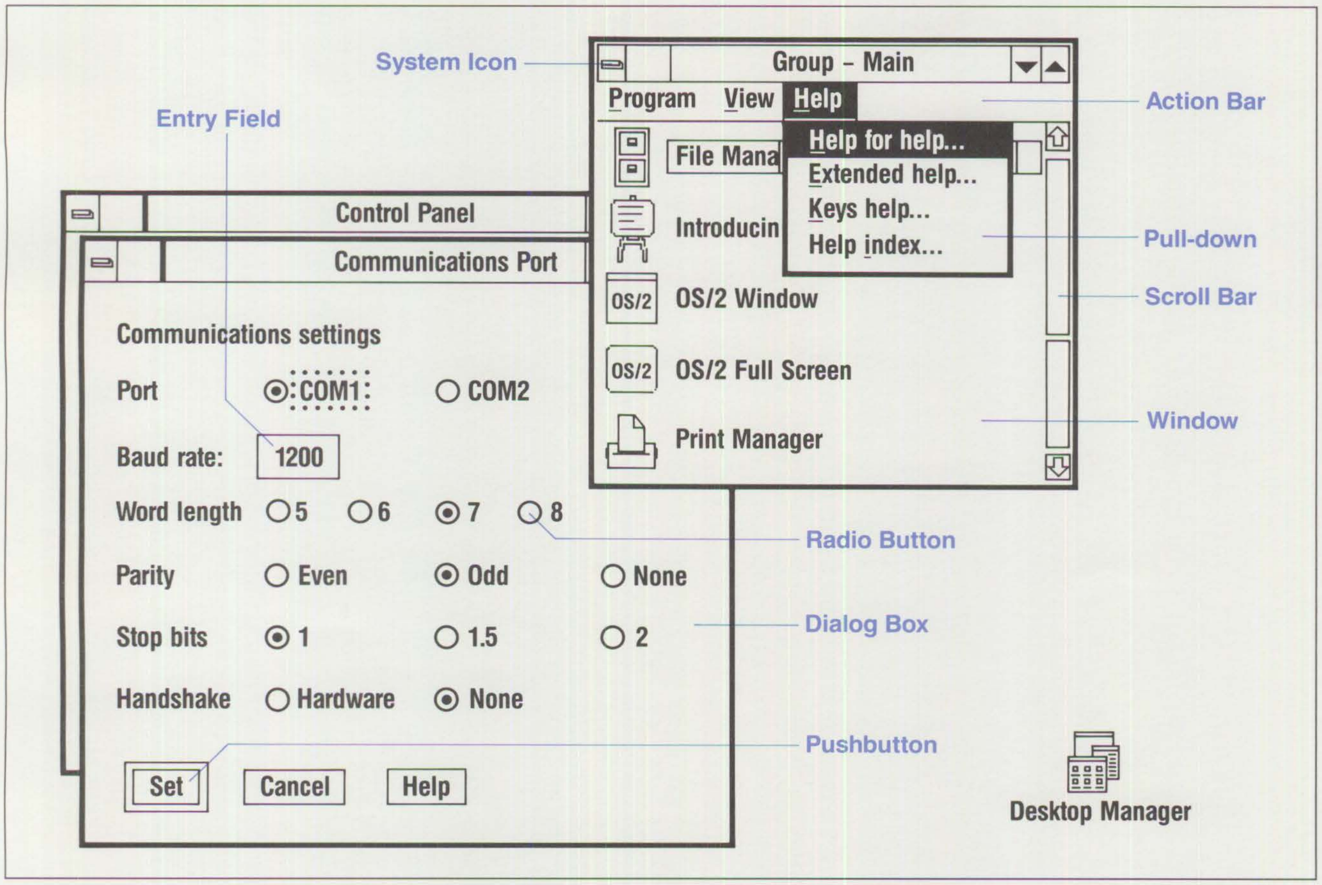
Single- and multiple-line entry fields are fill-in controls. Entry fields may have a default value or may be filled in by users.

Designing CUA Applications

CUA can easily be incorporated into a new application if it is included in the early design process. Some tools to help you to include CUA-compliant components are available, such as OS/2 Presentation Manager™ and Dialog Manager.

To increase application usability, your design should follow CUA guidelines, along with some common sense. Some basic guidelines to follow are to:

- Keep it simple
- Allow recovery from mistakes
- Limit the amount of information the user must remember
- Group common objects close together



Communications settings

Port	[✓ COM1]	COM2		
Baud rate:	1200			
Word length	5	6	[✓ 7]	8
Parity	Even	[✓ Odd]	None	
Stop bits	[✓ 1]	1.5	2	
Handshake	Hardware	[✓ None]		

Figure 1. CUA Components

- Offer a default when there is a common answer
- List the most frequently requested items first
- Allow the interface to be tailored where possible for transition from new to experienced users

Where possible, pertinent information in help and information panels should be on one page.

Summary

CUA sets a standard of providing application users easy learning and transition. By using the tools available, your applications can be made easier by designing CUA into them. CUA will continue to evolve as interface technology improves, giving users improved access to their work.

References

- *SAA, Common User Access: Basic Interface Design Guide* (SC26-4583)
- *SAA, Common User Access: Advanced Interface Design Guide* (SC26-4582)
- *SAA, Writing Applications: A Design Guide* (SC26-4362)
- *SAA, A Guide for Evaluating Applications* (G320-9803)

ABOUT THE AUTHOR

Jennifer Wilson is a senior associate programmer in IBM's Entry Systems Division (ESD) in Boca Raton, Florida, working in OS/2 Systems Assurance. Since joining IBM in 1985, she has worked in ESD development for educational software and interface tools, as well as on the design and development of the DOS 4.00 shell. Jennifer has a dual degree in computer science and mathematics from Pennsylvania State University, and recently received an M.B.A. from Nova University.

Little Solutions

*Welcome to **Little Solutions**. Starting with this issue, this column will be a regular feature. It will become a repository for those tidbits of information that are just lurking in the back of the mind...waiting to get out and help the readers with a problem they may come across...one whose solution may cause a deep sigh of relief and, just maybe, elicit a "Why didn't I think of that?" response. Then again, it could be a short-and-sweet solution that took its author weeks to stumble across.*

*We hope you enjoy the column, and welcome your ideas and submissions that can make a workday task easier for other readers. Read on, and send us your own **Little Solutions** in care of the Editor.*

WinSetPresParam(...

WinSetPresParam is a new API in OS/2 Version 1.2. Its purpose is to set (or change) a presentation parameter of a window. This change can be the background and foreground, hilite or normal, or disabled colors of a window. It can also be used to set or change the font name and size or the font handle.

WinSetPresParam is an easy way to set the presentation parameters of controls. A Listbox was the only control that the programmer had direct control over in OS/2 Version 1.1. Now that same capability and more can be used for all controls.

One of the idiosyncrasies of the API is how you pass the size of the presentation parameter. When setting a font in a control, you may have the following:

```
fontname = "8.helv"
WinSetPresParam(hwnd,
PP_FONTNAMESIZE,
sizeof(fontname)+1, fontname)
```

Be sure that the "sizeof" parameter is increased by one to accommodate the NULL terminator that C requires. If this is not done, the results of the call will be unpredictable. – *Larry Pollis, IBM, Dallas*

Enhanced 80386 Memory Adapter Needs Installation Program

The Enhanced 80386 Memory Expansion Option (referred to hereafter as the Memory Adapter) requires an installation program when used in a PS/2. These programs are described in more detail in the *PS/2 Hardware Interface Technical Reference* (S68X-2330).

When any adapter is installed in a PS/2, the Automatic Configuration Utility must be used to tell the PS/2 that the adapter is there. This utility requires an Adapter Description File (ADF file), which describes the adapter being installed. Most adapters on the market today only require this file. However, some memory adapters may also require an Adapter Description Program (ADP). If required, ADP is identified by the EXEC_ADP entry in the ADF file. The SETUP program uses the adapter description program to initialize these adapters before the Power On Self Test (POST) is run. POST checks to see if the hardware is working properly.

Complicated adapters (like the Enhanced 80386 Memory Expansion Option) require an initialization program that runs after POST. Before a PS/2 can use the adapter, this program must have information stored on track 0 of the hard file or have a device driver on the boot diskette.

Using an initialization program means that POST, and therefore the system reference diskette, won't recognize the memory installed, because it has not yet been initialized. This means that displaying the system configuration, using the system reference diskette, won't show the adapter memory even though it's correctly installed. However, operating systems (DOS, OS/2, and AIX) will recognize and use this memory as long as the track 0 program is installed (when booting from hard disk) or the device driver is installed (when booting from diskette or server image).

The Memory Adapter uses an initialization program to support 1 MB, 2 MB, and 4 MB Single In-Line Memory Modules (SIMMs), installed in any combination in any of the four SIMM sockets. These programs are installed on the fixed disk by the Set Configuration Option of the main menu on the systems reference diskette and must be used to install an initialization program correctly. The programs are *not* installed by the automatic configuration program.

The Set Configuration Option examines the adapter's ADF file. If this file contains an INITPROG statement, the required code is installed on track 0 of the fixed disk. This program will have a name like ICardID.ADF, where the CardID part is the adapter's POS identification in hex. For the Memory Adapter the ADF file is IFDDF.ADF.

To understand how these programs work, see what happens at IPL time. After powering on, POST is executed. If the hardware is working, POST passes control to the operating system. It expects to find the boot record on track 0. This boot record loads the operating system.

At installation time, based on information in the Memory Adapter's ADF file on the reference diskette, setup moves the master boot record on track 0 and replaces it with the initialization program for the adapter. When POST passes control to what it thinks is the master boot record, it's really passing control to the adapter's initialization program. When the adapter setup is complete, the initialization program passes control to the master boot record, and the operating system is loaded.

When power is turned on, POST checks the hardware and counts any memory that it knows about. Because the Memory Adapter has not yet been initialized, POST doesn't see this memory and won't count it. This doesn't mean the memory won't be used by the operating system. When POST is complete, it passes control to the Memory Adapter's initialization program, which counts the memory installed on the adapter. It then turns control over to the boot record, and the operating system is loaded.

Assuming a PS/2 Model 70 with 4 MB installed on the system board, 6 MB installed on the Memory Adapter, and a power-on password installed, POST will count to 4096, prompt for the power-on password, then turn control over to the initialization program, which will count from 4097 to 10240. Control is then passed to the operating system, and it's loaded. *Note*: Both OS/2 and AIX require at least 3 MB of memory for installation. If only 2 MB of memory is installed on the system board, the Enhanced 80386 Memory Expansion Option Diskette, Version 1.03, must be used, or the installation cannot be completed. – *Chuck Hanford, IBM, Dallas*

Installing an i486 Power Platform

The key is to follow the installation instructions; otherwise, problems will ensue. First read the IBM PS/2 Model 70 486 Software Support Package. Next, read the installation instructions prior to installing both the power platform kit and the communications software. Each of the following have their own steps:

3270 Connections: With the IBM Token-Ring Network Adapter/A or the Local Area Network Program, some files in the 486 support diskette must replace some files on your fixed disk. If these files are not replaced, problems may occur while bringing up communications on the power platform.

LAN 1.32: The NETWORK1.CMD file, usually found in the \pclpxs\net1_30 directory on the fixed disk, must be updated. Use the DOS COPY command and the support diskette to replace the old file with the new one. This is a read-only file, and the read-only attribute must be turned off before replacing the old file.

Network Bridge Program 1.0: ECCC0MOD.SYS, usually found in the root directory, must be replaced.

IBM Token-Ring Network Bridge: ECCC0MOD.SYS and ECCSBMOD.SYS files must be replaced. They are usually found in the R_BRIDGE directory.

LAN Support Program 1.1: The DXMC0MOD.SYS and DXMT0MOS.SYS files, usually found in the root directory, must be replaced.

The token-ring diagnostics on the PS/2 Model 70 486 reference diskette must also be updated. Use the DOS COPY command to copy the files @E000.DGS and DIFPDPM.EXE from the support diskette to a backup copy of the Model 70 486 reference diskette. – *Chuck Hanford, IBM, Dallas*

New Products

Hardware

IBM PS/2 Model 80 (8580-A21, -A31)

The PS/2 Model 80 family is expanded with two new models, the 8580-A21 and 8580-A31. The Model 80 A21 and A31 are high-function, Micro Channel floor-standing systems utilizing the Intel® 80386 processor running at 25 MHz with 64 KB memory cache. With these systems, IBM introduces Small Computer System Interface (SCSI) as a standard feature to the PS/2 product line. In addition, this system contains 4 MB of 80-nanosecond, high-speed memory, and a Direct Memory Access (DMA) parallel printer port. The 8580-A21 contains a 120 MB SCSI fixed-disk drive with 23 millisecond average access time, and the 8580-A31 contains a technologically advanced 320 MB SCSI fixed-disk drive with a 64 KB look-ahead buffer and 12.5 millisecond average access time.

Design enhancements to these systems offer significant advantages in configuration flexibility and expansion. In addition to providing seven available adapter slots, the standard configuration will support up to five internal Direct Access Storage Devices (DASD), such as diskette drives, fixed-disk drives, tape backup and CD-ROM. An optional feature allows support of up to six internal DASD devices.

The IBM PS/2 4 MB System Board Memory Expansion Kit allows greater system board memory expansion. This feature is for the Model 80 A21 and A31 only and provides an additional 4 MB of 80-nanosecond system board memory.

The IBM PS/2 1.44 MB One-inch High Diskette Drive Kit C allows the installation of the second 1.44 MB diskette drive.

The Models 8580-A21 and 8580-A31 are supported by OS/2 Standard Edition 1.1 and 1.2, OS/2 Extended Edition 1.1 and 1.2, and DOS Version 3.30 and 4.00, and Advanced Interactive Executive PS/2 (AIX PS/2) Version 1.2.

Highlights:

- PS/2 Micro Channel SCSI Adapter – as standard DASD controller
- 64 KB memory cache
- 4 MB standard system board memory – expandable to 8 MB
- 120 MB SCSI 23 millisecond 320 MB SCSI 12.5 millisecond fixed-disk drive models
- Micro Channel architecture with eight I/O slots: the 8580-A21and -A31 contain four 32-bit slots and four 16-bit slots
- Five internal DASD bays – optionally expandable to six bays
- Support for 5.25-inch internal DASD device

IBM PS/2 Model 80-061, -121, -321

The PS/2 Model 80 family is expanded with two new models, the IBM PS/2 Model 8580-121 and the IBM PS/2 Model 8580-321. The 8580-121 and -321 are high-function Micro Channel floor-standing systems utilizing the Intel 80386 processor running at 20 MHz.

Standard features include 1.44 MB, 3.5-inch, half-high diskette drive, VGA graphics and 2 MB of 80-nanosecond memory on the system board. The fixed-disk controller is the newly introduced Small Computer System Interface, the PS/2 Micro Channel SCSI Adapter. This bus master adapter has additional expansion capability and an interface for the new 3.5-inch, half-high SCSI fixed-disk drives of either 120 MB (8580-121) or 320 MB (8580-321). Both models can be expanded to 4 MB of memory on the system board and support up to 16 MB of system memory.

Design enhancements to these systems offer significant advantages in configuration flexibility and expansion. In addition to providing several available adapter slots, the standard configuration will support up to five internal Direct Access Storage Devices (DASD), such as diskette drives, fixed-disk drives, tape back-up, and CD-ROM. An optional feature allows support of up to six internal DASD devices.

The Models 8580-121 and 8580-321 are supported by OS/2 Standard Edition 1.1 and 1.2, OS/2 Extended Edition 1.1 and 1.2, the DOS Version 3.30 and 4.00, and Advanced Interactive Executive PS/2 (AIX PS/2) Version 1.2.

Highlights:

- PS/2 Micro Channel SCSI Adapter – as standard DASD controller
- 2 MB SCSI 23 millisecond or 320 millisecond SCSI 12.5 millisecond fixed-disk drive models
- 120 MB SCSI 23 millisecond or 320 MB SCSI 12.5 fixed-disk drive models
- Micro Channel architecture with eight I/O slots: 8580-121, -321 contain three 32-bit slots and five 16-bit slots (SCSI adapter requires one 16-bit slot)
- Five internal DASD bays – optionally expandable to six bays
- Support for 5.25-inch internal DASD bay

IBM PS/2 Model 65 SX -061, -121, -321

The PS/2 Model 65 SX 8565-061 and the 8565-121, two new Micro Channel architecture models based on the popular Intel 80386 SX® processor, utilize the 80386 SX processor running at 16 MHz. Standard features include a 1.44 MB, 3.5-inch, half-high diskette drive, VGA graphics and 2 MB of memory on the system board. The fixed-disk drive controller is the recently introduced Small Computer System Interface (SCSI), the PS/2 Micro Channel SCSI Adapter. This bus master adapter has additional expansion capability while providing an interface for the new 3.5-inch, half-high SCSI fixed-disk drives of either 60 MB (8565-061) or 120 MB (8565-121). Both models can be expanded to 8 MB of memory on the system board and support up to 16 MB of system memory.

Design enhancements to these systems offer significant advantages in configuration flexibility and expansion. In addition to providing seven available adapter slots, the standard configuration will support up to five internal Direct Access Storage Devices (DASD), such as diskette drives, fixed-disk drives, tape backup and CD-ROM. An optional feature allows support of up to six internal DASD devices.

The model 8565-061 and 8565-121 are supported by OS/2 Standard Edition 1.1 and 1.2, OS/2 Extended Edition 1.1 and 1.2, the DOS Version 3.30 and 4.00.

Highlights:

- PS/2 Micro Channel SCSI Adapter – as standard DASD controller
- 60 MB or 120 MB SCSI 23 millisecond fixed-disk models
- 2 MB memory standard on system board – expandable to 8 MB
- Micro Channel architecture with eight 16-bit I/O slots (SCSI adapter requires one I/O slot)
- Five internal DASD Bays – optionally expandable to six bays
- Support for 5.25-inch internal DASD device

IBM PS/2 Model 25 286 (8525-006, G06, 026, G36)

The PS/2 Model 25 286 system unit is an enhanced version of the PS/2 Model 25 and features an 80286 microprocessor, expanded system board memory (.5 to 4 MB), new 3.5-inch DASD options, and an integrated 12-inch VGA color monitor. The PS/2 Model 25 286 utilizes the 80286 processor operating at 10 MHz with one wait state to system memory and has the following integrated functions: parallel port, serial port, pointing device port, keyboard port, 1.44 MB diskette drive support, and VGA graphics. The PS/2 Model 25 286 is offered in diskette only and 30 MB fixed-disk models with the IBM Enhanced (101-key) or Space Saving (84-key) Keyboard.

The following new options are also offered:

- IBM PS/2 1.44 MB One-inch High Diskette Drive I
- IBM PS/2 20 MB Fixed Disk Drive I
- IBM PS/2 30 MB Fixed Disk Drive I
- IBM PS/2 Fixed Disk Drive Kit A

Highlights:

- .5 MB (diskette only) or 1 MB memory standard on the system board
- Capability for increased memory to 4 MB on the system board
- 30 MB fixed-disk capacity
- Audio earphone connector
- 12-inch VGA color monitor

PS/2 Micro Channel SCSI Adapter

The IBM PS/2 Micro Channel SCSI Adapter is a 16-bit Micro Channel bus master adapter that is expandable for all PS/2 Micro Channel architecture systems. This adapter connects multiple internal and/or external SCSI devices and has a 8.3 MB per second Micro Channel burst transfer rate. The adapter card may be installed in a 16- or 32-bit card slot and connects SCSI high-performance fixed-disk drives, scanners, printers, CD-ROMs and other SCSI peripherals.

The IBM PS/2 Card to Option Cable attaches the first external SCSI device to the Micro Channel SCSI adapter. The PS/2 Micro Channel SCSI Adapter Option Interface Terminator, included with the PS/2 Card to Option Cable, is a terminator installed on the last SCSI device to terminate the SCSI bus.

The IBM PS/2 Option to Option Cable is a standard SCSI cable that attaches an external SCSI device to another external SCSI device.

Highlights:

- Industry-standard interface
- PS/2 16-bit intelligent bus master adapter
- Supports up to seven physical SCSI devices
- Supports internal and external SCSI devices (single-ended)
- Micro Channel data transfer rate of up to 8.3 MB per second
- Supports asynchronous or synchronous SCSI devices

IBM PS/2 Micro Channel SCSI Adapter with Cache

The PS/2 Micro Channel SCSI Adapter with Cache is a 32-bit bus master adapter with expandability for all PS/2 Micro Channel systems. This Small Computer System Interface (SCSI) adapter can be installed in either a 16- or 32-bit card slot, has a 16.6 MB burst transfer rate, and a 521 KB Cache buffer. It adheres to the Small Computer System Interface (SCSI), which connects for multiple internal and/or external SCSI devices. These devices include SCSI high-performance fixed-disk drives, scanners, CD-ROM drives, and other SCSI peripherals.

The PS/2 Micro Channel SCSI Adapter External Terminator is a plug required to terminate the SCSI bus at the external port on the SCSI adapter. The external terminator is installed at the SCSI adapter port when no external SCSI devices are attached.

Highlights:

- Industry-standard interface
- PS/2 32-bit intelligent bus master adapter
- Supports a maximum of seven physical SCSI devices
- Supports internal and external SCSI devices (single-ended)
- Installable in any full-length micro channel 16- or 32-bit slot

- Micro Channel data transfer rate of up to 16.6 MB per second
- Supports asynchronous or synchronous SCSI devices

IBM PS/2 320 MB, 120 MB, 60 MB SCSI Fixed-Disk Drive and Kit A

IBM introduces a new family of storage devices, the PS/2 60 MB SCSI Fixed Disk Drive, 120 MB SCSI Fixed Disk Drive, and 320 MB SCSI Fixed Disk Drive. These Small Computer System Interface (SCSI) fixed-disk drives, with either a PS/2 Micro Channel SCSI Adapter or a PS/2 Micro Channel SCSI Adapter with Cache installed, enhance the storage expandability of all PS/2 Models 8560, 8565, and 8580.

The PS/2 SCSI Fixed Disk Drive Kit A is specifically designed and required to support the installation of two SCSI fixed-disk drives in the front bay of PS/2 Models 8560, 8565, and 8580 with a PS/2 Micro Channel SCSI Adapter or a PS/2 Micro Channel SCSI Adapter with Cache installed.

Highlights:

- Industry-standard SCSI interface (single-ended)
- High-performance storage devices
  - 60 MB formatted capacity, 23 millisecond average access time, 32 KB buffer
  - 120 MB formatted capacity, 23 millisecond average time, 32 KB buffer
  - 320 MB formatted capacity, 12.5 millisecond average access time, 64 KB look-ahead buffer
- Improved system configuration flexibility

PS/2 CD-ROM Drive and Kit A

The PS/2 CD-ROM Drive is a Compact Disc-Read Only Memory drive. This internal device expands application solutions of all PS/2 Models 8560, 8565, and 8580 with Micro Channel by enabling various applications such as catalogs, libraries, and archives that are distributed on compact discs to be accessed by a computer. The PS/2 CD-ROM Drive adheres to the Small Computer System Interface (SCSI) and supports compact discs with 600 MB typical capacity.

The IBM PS/2 CD-ROM Drive Kit A is specifically designed and required to support the installation of a PS/2 CD-ROM Drive in PS/2 Models 8580, 8565, and 8560 with a PS/2 Micro Channel SCSI Adapter or the PS/2 Micro Channel SCSI Adapter with Cache installed.

Highlights:

- Industry-standard SCSI interface (single ended)
- Industry-standard compact disc (ISO 9660)
- 380 millisecond average access time
- Analog stereo headphone output jack
- Improved flexibility

PS/2 SCSI External CD-ROM and Cables

The PS/2 External CD-ROM Drive Model 001 (3510-001) is an external Compact Disc-Read Only Memory drive. This external device expands applications solutions of all PS/2 Micro Channel systems by enabling various applications such as catalogs, libraries, and archives, which are distributed on Compact Discs to be accessed by a computer. The PS/2 External CD-ROM Drive utilizes the Small Computer System Interface (SCSI) and supports compact discs with 600 MB typical capacity.

The PS/2 External CD-ROM Drive can be attached to any PS/2 Micro Channel system with a PS/2 Micro Channel SCSI Adapter or the PS/2 Micro Channel SCSI Adapter with Cache installed. The first PS/2 External CD-ROM Drive is attached to the SCSI adapter through a PS/2 Card to Option Cable. Additional PS/2 External CD-ROM Drives or external SCSI devices are attached through the PS/2 Option to Option Cable.

Highlights:

- Industry-standard SCSI interface (single-ended)
- Industry-standard compact disc (ISO 9660)
- 600 MB typical formatted capacity (disc dependent)
- 380 millisecond average access time
- Analog stereo output jack
- Universal 32-watt power supply
- External SCSI ID selector
- Pushbutton power switch with power-on indicator

IBM PS/2 Internal Tape Backup Unit Installation Kit A

The IBM PS/2 Internal Tape Backup Unit Installation Kit A allows the IBM PS/2 Internal Tape Backup Unit, which is sold separately (Feature 5279, part number 30F5279), to connect and be easily installed in the new PS/2 Model 80 - A21, A31, 121, 321 and the new PS/2 Model 65 SX - 061, 121 system units.

Highlights:

- Allows installation of PS/2 Internal Tape Backup Unit in new system units of the PS/2 Model 80s and Model 65 SX
- Easy to install
- Flat cable, 1.7 inches long, with edge connector on one end and 34-pin connector on other end

IBM PS/2 Model P70 386

The PS/2 Model P70 386 (8573-031) is a new model of the P70 family, providing high-function at a lower entry point than the previously announced models. The 8573-031 is a 16 MHz version, with a 30 MB fixed-disk drive. It includes Micro Channel architecture, a VGA, 16-gray scale plasma display and a fully compatible PS/2 enhanced keyboard, all integrated into a single package. The new model features the 80386

32-bit microprocessor, high-density memory technology, and a wide range of integrated features. With the capability of supporting up to 16 MB of high-speed real memory, 30 MB of disk storage, full VGA graphics, and an optional 80387 math co-processor, the 8573-031 provides significant performance for portable computer operations. All P70 models maintain compatibility with most existing software products for IBM personal systems.

The 8573-031 provides 16 MHz 80386 performance with 2 MB of high-speed (85 ns) memory and a 30 MB fixed disk.

Highlights:

- 30 MB portable version of 16 MHz PS/2 Model 70 (8570)
- 2 MB memory standard, expandable to 8 MB on the system board
- High-quality, 16-gray scale-integrated gas plasma display
- 80386 16 MHz 32-bit microprocessor and optional co-processor
- 30 MB fixed disk with integrated controller
- Micro Channel architecture with a 16- and 32-bit bus
- One full- and one half-length expansion slot
- Ergonomic briefcase, portable design
- External storage device port

IBM PS/2 Color Display 8515

The IBM PS/2 Color Display 8515 is a multimode, analog, color display featuring high contrast and clarity for alphanumeric, graphics, and image applications. The 14-inch screen has a data area of 9.8 x 7.4 inches. It supports all VGA modes and has a maximum of 1024 x 768 pixel addressability, and 256 colors at a time may be selected when using the IBM 8514/A Display Adapter.

In response to emerging customer requests, the PS/2 Color Display 8515 has lower VLMF (Very Low Magnetic Field), meeting field emission requirements of Scandinavia. The tilt/swivel base is standard.

Highlights:

- 14-inch display with four programmable selected modes up to 1024 x 768 pixel resolution.
- 0.28 mm phosphorus-dot pitch permitting sharp, clear viewing
- Data area 42 percent larger than the 8513
- High-contrast screen with reduced glare and minimum color washout in bright lighting conditions
- Tilt/swivel pedestal with 300 degrees rotation

IBM LaserPrinter E (4019-E01)

The IBM LaserPrinter E is a lower-speed version of the IBM LaserPrinter. It can produce letter-quality text at up to five pages per minute, and graphics at up to 300 x 300 dots per inch (DPI). Datastream, graphics, and paper-handling capability of the IBM LaserPrinter E is identical to that of the IBM LaserPrinter.

The IBM LaserPrinter E uses the same single-element print cartridge, and supports all the optional paper-handling features, memory cards, font cards, downloadable font packages, PostScript® options, and printer-sharing option currently available for the IBM LaserPrinter.

The IBM LaserPrinter E can be converted to the 10-page-per-minute IBM LaserPrinter via the upgrade option. The IBM LaserPrinter E attaches to IBM PS/2 products, IBM Personal Computers, and selected IBM displays and IBM systems.

Highlights:

- Prints up to five pages per minute, up to 300 DPI for text and image on a variety of stationery supplies, in either portrait or landscape orientation
- Compact size: 360 mm X 469 mm (14.2 X 18.5 in) footprint
- Single-element high-yield cartridge with up to 10,000-page yield in typical text-printing applications
- Two hundred-sheet auto feeder standard, 500-sheet second drawer and auto envelope feeders optional
- Support for IBM Personal Printer Data Stream (PPDS), Hewlett-Packard® (HP®) LaserJet® Series II emulation, and plotter emulation (IBM 7372 and HP 7475A color plotters)
- Extensive font support: resident fonts, downloadable fonts, and font cards
- Commonality of features and supplies with the IBM LaserPrinter (4019-001)
- 512 KB memory standard, expandable to 1.5 MB, 2.5 MB, or 4 MB
- Can be shared by up to six PCs via the printer sharing option

PostScript Options and Download Font Card for the IBM LaserPrinter

The PostScript Options provide the IBM LaserPrinter with PostScript processing capability. They consist of the PostScript Option, which has 17 resident outline fonts; the PostScript Option (Premium Font Package) has 17 resident, plus 22 outline fonts (located on the Outline Font Card). The Outline Font Card is available separately.

Screen fonts in the Microsoft Windows and the Adobe® formats are provided, and allow the document to be displayed as it will be printed.

Highlights:

- Full PostScript page description language capability
  - — Choice of either 17 or 39 outline fonts
  - — Screen fonts to display the documents as it will be printed
- 3.5 MB memory card supports more complex jobs in all IBM LaserPrinter operating modes

— Can increase performance in PostScript mode

— Provides storage for additional fonts

Bell and Howell Copiscan II Document Scanners

The Bell and Howell Copiscan®II Document Scanner, Models 2135 and 3338, allows users to convert documents to digital records that can be managed, transmitted, displayed, printed, and stored. The scanners are part of IBM's ImagePlus ™ support and are attachable to an IBM PS/2 via an IBM Image Adapter/A Card.

The Copiscan II Document Scanner, Models 2135 and 3338, replaces the Copiscan I, Model 2115I, and has improved scanning capabilities.

Models 2135 and 3338 have been verified for compliance with FCC rules as FCC, Class A devices for use only in industrial, commercial, or business environments.

The Model 2135 has an average scanning speed of 1.2 seconds per 8- 1/2-by-11-inch page and 200 dot-per-inch resolution. The transport speed is 9.0 inches per second with a data rate of 5.0 Mbps. It handles document sizes from 2-1/2 to 8-1/2 inches wide and 14 inches long.

The Model 3338's average scanning speed per 8-1/2-by-11-inch page depends on the resolution selection (300 or 200 dots per inch, respectively). It handles documents sizes from four to 11 inches wide and 17 inches long.

Highlights:

- High-quality resolution; selectable resolution with Model 3338
- Accepts various document sizes
- High transport speed and data rate
- More reliable due to less mechanical parts
- Accepts a wide range of document thickness ranging from .002" to 008"

Software

IBM OfficeVision/2 Release 1.1 Availability

OfficeVision/2 Release 1.1 offers increased flexibility in the areas of migration and communications configuration. The OS/2 Office Feature now suggests OS/2 Extended Edition Version 1.2 and OS/2 LAN Server Program Version 1.2, providing a logical migration path to future OfficeVision/2 offerings. The OS/2 Office DOS Requester feature download option allows the user to execute the requester code from the IBM Disk Operating System (DOS) workstation that may result in improved performance.

A key enhancement to the OS/2 Office Feature is the application launch facility that allows a customer to add OS/2 applications to the Office window. The user provides the application name and the desired icon (a personalized icon, using the OS/2 icon editor or the default icon) in a single-step enrollment process. Once enrolled, each OS/2 application can be invoked via an icon on the office window. A new "Forward-With-A-Reply" mail function enables a user to add text to an existing note and forward the updated note to other users.

Availability of U.S. English is planned for May 25, 1990.

This product features an application launch facility, which allows the user to SNAP-ON OS/2 application as icons onto the main OS/2 Office window.

IBM PS/2 Internal Tape Backup Program and Upgrade (OS/2 Version), and Convenience Kit.

The PS/2 Internal Tape Backup Program (OS/2 Version) enhances the PS/2 Models 50, 50Z, 60, 70, and 80 by allowing users to transfer up to 120 MB of OS/2 formatted data from a disk storage device to a removable .25-inch mini tape cartridge for later recall.

For the customer's convenience, the IBM PS/2 Internal Tape Backup Convenience Kit (OS/2 Version) has in one package the IBM PS/2 Internal Tape Backup Program (OS/2 Version), the IBM PS/2 Internal Tape Backup Unit and formatted mini tape cartridge. The IBM PS/2 Internal Tape Backup Program provides the customer with an easy-to-use, easy-to-install, and technologically advanced internal tape backup system.

Customers can upgrade from the DOS version of the Internal Tape Backup Program to the OS/2 version by ordering the upgrade.

This product gives users the ability to back up and restore data internally from the fixed disk while the systems management process is enhanced.

IBM DisplayWrite 5 Composer

IBM DisplayWrite™ 5 Composer is a PC DOS-based application product, containing all the functions of the IBM Personal Computer DisplayWrite 5/2 Composer licensed program, including a user-specified option to select between two different user levels, the ability to view and print text, images and graphics; and a command line function that gives an expert user the ability to bypass menus while in text. IBM and non-IBM printer support are featured. Direct customer support for end users via the IBM Support Center, 1 800 237-5511, will be available at general availability.

Linguistics functions such as spell aid, spell verify, hyphenation, and synonyms (in selected languages) are supported by means of the IBM DisplayWrite dictionaries that include 16 additional languages, plus legal and medical terminologies. Up to three of these dictionaries can be used concurrently with DisplayWrite 5.

Highlights:

- List processing
- Table of Contents/Index/Table of Authorities (reference lists)
- LAN server code included

- Host integration using the GCI Intel I860 Developer Toolkits for C and Fortran

IBM Current, Version 1.1

The IBM Current, Version 1.1 licensed program is a flexible and easy-to-use information manager that enables users to organize and retrieve personal data. Version 1.1 supports Microsoft® Windows Version 2.0 and 3.0.

A program upgrade option is offered, at no charge, that allows IBM Current, Version 1.0 users to obtain the new functions.

Highlights:

- Categories increased to 100; connections increased to 200; reports increased to 100; views increased to 100; items increased to 4,000 and fields increased to 50
- Dynamic Data Exchange (DDE) support
- Tagged windows can be saved; more fonts supported including fractional fonts
- Alarms can be set to signal appointments
- Apply auto-assignment rules during import
- Multiple data directories can be created for applications use
- Allows reversal of first and last names on reports
- Existing information in dBASE III®, dBASE IV™, DIF and ASCII files can be easily incorporated into IBM Current using the versatile import feature

Intel I860 Developer Toolkits for C and Fortran

The Intel i860™ Software Development Tools for FORTRAN vendor-logo programs that contain the software tools for development of FORTRAN applications for the IBM PS/2 Wizard Adapter (3462) feature. The software developer can develop numeric-intensive FORTRAN applications running under OS/2 that can take advantage of Intel i860 RISC processor. These tools are native compiler, consisting of an assembler, linker, debugger, FORTRAN compiler, libraries and vectorizers that that provide access to the capabilities of the i860 microprocessor.

Highlights:

- The combination of software development tools with the PS/2 Wizard Adapter (#7321) feature and the PS/2 Model 70 and 80 provides and excellent new platform for C an FORTRAN applications running under AIX and OS/2
- User productivity gains are realized through a flexible, desktop solution for a broad range of applications that typically run on workstation or mainframe platforms
- The customers investment is protected when an upgrade path migrates from a general-purpose PS/2 system configuration to a technical computing platform
- The application developers can expand the applications base to include both C and FORTRAN applications, running under OS/2 or AIX

AIX PS/2 Xstation Manager

The AIX PS/2 Xstation Manager licensed program supports the IBM Xstation 120, a low-priced, advanced-function X server station that is local area network (LAN)-attached. The combination of the AIX PS/2 Xstation Manager and Xstation 120 allows for X client graphics and character-based applications. Windows can be opened on the Xstation 120 by an application running on an AIX host, non-AIX hosts or non-IBM host that supports X Window System™ Version 11, Release 3. The X Window separated from user interaction and displayed (at the X server station) Xstation 120 users may control and view multiple applications in multiple windows on a single screen.

Highlights:

Supports for attachment of one or more IBM Xstation 120 systems

- Allows Ethernet® or optional IBM Token Ring attachment
- Is designed to be compatible with X Window System Version 11, Release 3
- Provides functions that are equivalent to the IBM AIX Xstation Manager/6000

IBM AIXwindows Environment for PS/2

The AIXwindows™ Environment is a state-of-the-art graphical user environment for PS/2 models using the AIX® PS/2 Operating System. The AIXwindows Environment lets users develop and run AIXwindows application and AIX PS/2 X-Windows-based applications. The AIXwindows Environment for PS/2 consists of the following components:

- AIXwindows – A graphical user interface and tool kit based on the OSF/Motif™ user environment component
- AIXwindows Desktop – A graphical desktop that allows users to browse the file system and start-up applications
- AIX PS/2 X-Windows Version 1.2 based on the X Window System Version 11 Release 3

Highlights:

- AIXwindows graphical user interface and on OSF/Motif
- Desktop function for end users
- AIX PS/2 X-Windows support

AIX PS/2 NextStep Environment Version 1.1

AIX PS/2 NextStep™ Environment is a state-of-the-art graphical user interface and programming environment for IBM AIX workstations, and compatible with the same application programming interface (API) as the NextStep product

(Software Release 1.0, provided by NeXT® Inc.). (Note: AIX PS/2 NextStep Environment Version 1.1 is marketed as AIX PS/2 Graphic User Environment Version 1.1 in some other countries).

AIX PS/2 NextStep Environment Version 1.1 has icons and menus for easy access to system utilities and applications. The AIX NextStep Interface Builder® has a rich set of well-defined objects and graphical cut-and-paste capabilities for designing and implementing application user interfaces. The Objective-C™ Complier provides the benefits for object-oriented programming for developers who choose to design additional objects for the application development environment. AIX PS/2 NextStep Environment is designed to increase the productivity of programmers and end users.

Highlights:

- A Window Server
- Display PostScript interpreter
- Workspace management tool
- Full use of screen and control of multiple applications
- Objective-C capability to design additional objects for the application development environment
- Cut-and-paste capabilities for designing and implementing application user interfaces
- Client/Server model for distributed windows using PostScript protocol

IBM PS/2 ImagePlus Workstation Program Version 1.1

IBM PS/2 ImagePlus Workstation Program Version 1.1 is the latest release of the ImagePlus Workstation Program designed to capture, view, print, and manipulate image documents on an IBM PS/2 Micro Channel architecture computer in a DOS Version 4.0 environment. This program is a component of the IBM ImagePlus MVS/Enterprise Systems Architecture (MVS/ESA™), Application System/400® (AS/400™), and System/36 Systems, which allow the conversion of paper-intensive applications to computer-based electronic systems. It is being made available with a number of new functions designed to further enhance high-volume image document handling in high-performance environments.

Highlights:

- New functions (table of contents, movable horizontal ruler line, page marking, adjustable zoom box, and enhanced document / working set manipulation) and additional scanner support, to improve workstation productivity
- New features (configurable display options and co-resident functions) to augment image-processing applications
- Statements of directions to prepare customers for high-speed batch scanning audio capture and playback, and future OS/2 versions of the IBM ImagePlus Workstation Program.

IBM SAA EASEL/2 Version 1.1 and Statement of Direction

IBM is announcing enhancements to the two currently available OS/2 Extended Edition Licensed Programs EASEL® for OS/2 EE Development System and EASEL for OS/2 EE Runtime Facility. These IBM programs are used for developing and executing graphical applications on OS/2 programmable workstations. With this announcement, EASEL now supports:

- OS/2 EE Database Manager
- Cooperative processing via an Advanced Program-to-Program Communication (APPC) mapped conversation interface
- Access to AS/400 applications through the OS/2 EE 5250 Workstation Feature
- The Common User Access (CUA) graphical model user interface enhancements of OS/2 Version 1.2

The Layout/CUA facility, which is part of the EASEL for OS/2 Development System, provides a highly productive CUA user interface definition facility that prompts application developers for conformance with Systems Application Architecture (SAA) and CUA guidelines. The EASEL for OS/2 EE Development System also contains the EASEL for OS/2 EE Runtime Facility to help developers debug newly developed EASEL for OS/2 EE applications.

IBM EASEL for OS/2 EE is in the Analysis/Design phase of the AD/CYCLE™ framework. It is a design tool allowing the developer to create the end-user interface part of a cooperative processing application.

EASEL can support cooperative processing applications, stand-alone applications and applications that are front ends to existing host terminal-based applications. Front ends can be written for host applications that use 5250 terminals, in addition to the existing EASEL support for front ends to host applications that use 3270 an asynchronous terminals.

EASEL for OS/2 EE is developed by Interactive Images Inc. Woburn, Mass.

Highlights:

- Growth can be enabled by writing new EASEL for OS/2 EE applications using new easily learned, powerful programming facilities that support cooperative processing and audio capture and playback
- User productivity gains can be achieved by implementing consistent graphical CUA and SQL-based applications
- Investments are protected by implementing EASEL for OS/2 applications that build on existing AS/400-based applications and existing data

IBM OS/2 Extended Edition Version 1.2 Available

OS/2 Extended Edition (OS/2 EE) Version 1.2 is available with the functions announced May 16, 1989, and the enhancements announced November 14, 1989.

IBM OS/2 Local Area Network Server Version 1.2 Availability

The OS/2 LAN Server Version 1.2 offers significant new functions in LAN resource and information sharing for IBM DOS and OS/2 Requesters on IBM Token Ring and personal computer networks. Enhancements include a DOS LAN Requester and LAN Support Program Version 1.1 included in the product, full-function server support for the OS/2 Extended Edition (EE) Version 1.2 Requester, a new file replication service for automatically copying selected files, enhanced Requester access to resources controlled by LAN Servers, a Server Migration Utility that will assist in the move from IBM Personal Computer LAN Program (PCLP) Version 1.3 Server and IBM OS/2 LAN Server Version 1.0 and additional LAN enhancements derived from the the IBM OS/2 Extended Edition Version 1.2 features, including support for Ethernet LANs.

IBM TCP/IP Version 1.1 For OS/2 Extended Edition

The Transmission Control Protocol/Internet Protocol (TCP/IP) Version 1.1 for OS/2 EE completely replaces the TCP/IP Version 1.0 offering. TCP/IP Version 1.1 for OS/2 Extended Edition offers all the function of the Version 1.0 product and makes available a new feature, Network File System™ (NFS™) Client. The NFS Client feature is based on Request for Comments (RFC), 1094, Network File System Protocol Specification.

Additionally, client support of the Serial Line Internet Protocol (SLIP) is provided. Support is also included for a File Transfer Protocol Application Programming Interface (FTP API) and PC Network. Support of the IBM Industrial Computer, GEARBOX™ Model 800 is announced for both TCP/IP Version 1.0 for OS/2 and for TCP/IP Version 1.1 for OS/2 Extended Edition.

The requirement for both the IBM C Language Complier Version 1.1 and the Microsoft C™ Compiler Version 5.1 has been removed. The only compiler requirement is the IBM C Language Compiler Version 1.1.

Highlights

- Business solutions enabled by the NFS Client function
- Growth enablement and investment protection facilitated by the NFS Client capability and support of the PC Network

IBM GDDM-OS Link Version 1.0 Available

GDDM™ - OS/2 Link provides host graphics support for the GDDM Series under OS/2 Extended Edition. The user can run typical GDDM applications in the OS/2 Presentation Manager windowing environment, save a picture locally, and print and plot via the OS/2 Presentation Manager.

IBM Gearbox Language Extensions Version 1.2 Available

GEARBOX Language Extensions Version 1.2 is now available.

GEARBOX Language Extensions for PC DOS and GEARBOX Language Extensions for OS/2 programs have been enhanced to support the GEARBOX Model 800. GEARBOX Language Extensions for PC DOS and OS/2 are designed to allow GEARBOX Model 800, 7552 users to write programs that take advantage of the unique features of the processors, such as power-fail notification, that are not a part of other IBM industrial computers.

IBM Spelling Series: Levels I-III Version 1.10

The IBM Spelling Series is an educational program that combines practices, tests, games, and activities making learning to spell enjoyable. The series develops and improves spelling skills for Grades 1-6. Spelling patterns and traditional "trouble words" are included. More than 700 words are presented. Product features include the ability to alter the lesson sequence, add spelling word lists, delete and hide lessons, and with the the use of speech adapter, provide teacher recorded and digitized speech. The Spanish alphabet can also be supported. The series may be used on IBM Personal Computers or PS/2 system units in a network or stand-alone environment.

Highlights:

- Creates a highly interactive learning environment for spelling instruction in grades 1-6
- Uses human speech to provide audible instructions, demonstrations, hints, and feedback, eliminating reading dependencies
- Provides a combination of structured and exploratory activities
- Permits teacher control of activities and lesson sequence
- Allows additional word lists to be added

IBM SciBench

IBM SciBench is a general-purpose laboratory data acquisition and processing system that allows a user to write simple data acquisition programs for displaying and plotting the data on any OS/2 display and OS/2 supported printer or plotter. IBM SciBench is designed to be used to a stand-alone plotting package or as a data acquisition package to meet the needs of the physical scientist.

Highlights:

- Uniquely designed to meet the needs of the physical scientist
- Stand-alone plotting package or data acquisition package
- Device-independent graphical hardcopy
- Prototype device drivers
- Display and plotting functions for graphical representation of scientific data

Personal Science Laboratory (PSL) Explorer Software and Curriculum

Personal Science Laboratory® (PSL®) Explorer Version 1.00 is the interactive software that gives teachers and students the ability to control microcomputer-based laboratory experiments.

Personal Science Laboratory (PSL) Curriculum Version 1.00 is the instructional material designed for students in grade levels 6 through 12 who are questioning the world around them and who are in the midst of developing important science and math skills. This curriculum materials are also applicable for college-for collage and university-level science and math students. By using the curriculum materials with the PSL Temperature, Light, pH, and Motion modules and sensors the student and teacher are guided through a variety of interesting experiments.

Highlights:

- Allows students to analyze, explore, ask "what if" questions, and use the computer to make measurements and to graph data
- Allows students and teachers to solve experimental problems by doing real-time data collection, graphing, and analyses with curricular emphasis on development of process skills, problem solving, and higher-order thinking skills
- Creates a self-paced, highly interactive learning environment of personal discovery and involvement, which helps stimulate students' curiosity
- Teacher guides and lesson plans designed to help teacher and student experiment in a guided format based on student ability and varying grade levels
- Provides experiments in various science subjects from grades 6-12
- Instruction is based on research methodologies
- Complements other IBM math and science offerings

Reading for Meaning; Levels II-IV Version 2.0

The IBM Reading for Meaning Series is an educational reading program that draws from current theory and technology. The series is designed to lead students in grades three through eight through a variety of instructions that increases reading awareness and comprehension while making the reader think about what is read. The levels indicate complexity so students are able to use reading materials that correspond to their achievement level. Features include colored illustrations, interesting stories, and editing capabilities, which allow teachers to create lessons of their own or edit existing ones.

Highlights:

- Allows the adding of teacher-oriented materials
- Permits the teacher to edit existing lessons
- Provides a 485-picture library to illustrate teacher-authored stories. A portion of these graphics support adult themes in support of adult literacy requirements
- 256 color mode graphics are used to create exciting, motivational pictures
- Online keyboard guide helps are included

Statement of Direction

Image Plus System PS/2 - Statement of Direction

IBM intends to provide an IBM ImagePlus System PS/2, an OS/2 LAN-based addition to the ImagePlus Family. This ImagePlus system will provide image capability for customers' OS/2 line-of business applications. It will provide image storage and retrieval, with document/folder/case and workflow management capabilities similar to current ImagePlus family members (ImagePlus System MVSS/ESA, AS400, and System/36). The system will utilize scanners, facsimile I/O (FAX), magnetic and optical disk(ettes), optical libraries, image displays, and page printers to capture, index, store, retrieve, and distribute image documents electronically. This will permit customers to convert large volumes of paper-based documents to electronic image processing. The system is being designed using IBM's SAA and ImagePlus object architectures and will be common user access (CUA) compliant. This system is being developed jointly by IBM and Eastman Kodak Company.

“Depending on the application, EMS memory can be used to contain code, data, and heap. (page 12)

“FASTOPEN can provide better performance that may not be available from system buffers. (page 23)

“Good grief! What is this error, and how can we fix it? (page 28)

“DOS LAN Requester provides many functional enhancements over PC LAN Program 1.3. (page 34)

“In some situations, the particular controls on a dialog box may be data dependent. (page 43)

“START2 gives the user two ways to specify the icon file. (page 54)

“System pull-downs include options to size, restore, move, minimize, and maximize the window. (page 66)

“Welcome to Little Solutions. (page 69)

G325-5007-00