

Issue 4, 1990

# Personal Systems



IBM Personal Systems Technical Solutions





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

Editor	Libby Boyd
Consulting Editor	Ed Bamberger
Automation Consultant	Andy Frankford
<i>Little Solutions</i>	Larry Pollis
Communications	Elisa Davis
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.

Permission to reprint material from this magazine is granted to publications that are produced for non-commercial use and do not charge a fee. When republishing, include the names and companies of authors, and add the words "Reprinted by permission of *IBM Personal Systems Technical Solutions*." Other

publications may reprint only with permission of the editor.

Titles and abstracts, but no other portions, of information in this publication may be copied and distributed by computer-based and other information-service systems.

IBM believes the statements contained herein are accurate as of the date of publication of this document. However, IBM hereby disclaims all warranties as to materials and workmanship, either expressed or implied, including without limitation any implied warranty of merchantability or fitness for a particular purpose. In no event will IBM be liable to you for any damages, including any lost profits, lost savings or other incidental or consequential damage arising out of the use or inability to use any information provided through this service even if IBM has been advised of the possibility of such damages, or for any claim by any other party.

Some states do not allow the limitation or exclusion of liability for incidental or consequential damages so the above limitation or exclusion may not apply to you.

This publication could contain technical inaccuracies or typographical errors. Also, illustrations contained herein may show prototype equipment. Your system configuration may differ slightly.

IBM has tested the programs contained in this publication. However, IBM does not guarantee that the programs contain no errors.

This information is not intended to be a statement of direction or an assertion of future action. IBM expressly reserves the right to change or withdraw current products that may or may not have the same characteristics or codes listed in this publication. Should IBM modify its products in a way that may affect the information contained in this publication, IBM assumes no obligation whatever to inform any user of the modifications.

It is possible that this material may contain reference to, or information about, IBM products (machines and programs), programming or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such products, programming or services in your country.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever.

All specifications are subject to change without notice.

© Copyright 1990 by International Business Machines Corporation

#### ABOUT THE COVER

*A mirror image? Look again. Closer examination will reveal a basic difference in the two images. Likewise, operating systems may appear similar at first glance, but have fundamental differences. The article, "Operating Systems Platforms: A Business Perspective," explains some of the differences in today's operating systems.*



# Contents

## Hardware

- 1** First Look at the New IBM PS/1™ Computer
- 8** Using the IBM 4019 LaserPrinter Effectively
- 18** Micro Channel® Interface Chip Sets
- 23** Token Ring Bus Master LAN Adapters
- 27** Extension of Wiring Rules for 4-Mbit/s Token Ring Using UTP Lobes
- 32** SCSI and DISK386.SYS

## Software

- 33** Operating System Platforms: A Business Perspective
- 39** Minimum OS/2® 1.2 DASD Requirements
- 45** User Profile Management
- 53** Understanding OS/2 1.2 LAN Server Performance
- 60** PM: An Object-Oriented Approach
- 75** DOS 4.00 SHARE
- 78** A "C" Programming Model for DOS Device Drivers

## Random Data

- 91** An Electronic Bulletin Board for PC Users
- 93** Little Solutions
- 95** New Products



## Trademarks

IBM, AIX, Micro Channel, Operating System/2, OS/2, Personal Computer AT, AT, DisplayWrite, Personal System/2, PS/2, PROFS, Quietwriter, Quickwriter, RT, Wheelwriter and Writing to Read are registered trademarks of International Business Machines Corp.

AD/Cycle, AIXwindows, Application System/400, AS/400, Assistant Series, Audio Visual Connection, AVC, BookManager, C/2, COBOL/2, Communications Manager, Database Manager, DisplayWrite 4, FORTRAN/2, GDDM, Independence Series, KnowledgeTool, LinkWay, MVS/ESA, MVS/SP, MVS/XA, NetView, NetView/PC, OfficeVision, OS/400, Pascal/2, Personal Computer XT, PC XT, Presentation Manager, PS/1, Storyboard, Systems Application Architecture, SAA, System/360, VM/XA, Writing to Write, and 3090 are trademarks of International Business Machines Corp.

ARCnet is a trademark of Datapoint Corp.

DESQview is a trademark of Quarterdeck Office Systems.

Ethernet is a registered trademark of Xerox Inc.

Excelsior is a registered trademark of Excelsior Software Inc.

Express Publisher is a trademark of Power Up Software Corp.

Hewlett-Packard, HP, HPGL, and LaserJet are registered trademarks of Hewlett-Packard Corp.

Intel is a registered trademark, and i860, 386, i486, 486, 80286, 80386, 80386SX, and 80486 are trademarks of Intel Corp.

LANSchool is a trademark of LanFan Technologies Inc.

Laserwriter NT is a trademark of Apple Computer Inc.

Lotus is a registered trademark, and Lotus 1-2-3 is a trademark of Lotus Development Corp.

Microsoft, MS, and Works are registered trademarks, and Microsoft C and Windows are trademarks of Microsoft Corp.

Novell is a trademark of Novell Inc.

Omnilaser is a trademark of Texas Instruments.

Postscript is a trademark of Adobe Systems Inc.

VM/386 is a trademark of IGC.



## First Look at the New IBM PS/1 Computer

*E. Gene Barlow  
IBM Corporation  
Dallas, Texas*

**IBM recently announced a new PS/1 personal computer system and a wide selection of software and services to go with it. Because of all the hardware options, software applications, and service offerings available with the PS/1 system, many small businesses, students, and home computer users will find it ideal for their personal computing needs.**

*Editor's note: To more effectively reach home users, the PS/1 is sold by leading department stores and by IBM authorized dealers. It is not sold through IBM branch offices. Service and technical support are described in this article. Support for the PS/1 is not available through IBMLink.*

Let's take a look at the new IBM PS/1, its technical features and capabilities. We'll look first at the PS/1 hardware systems announced, then examine the software features that come with the system, and finally investigate some service and support programs offered only to PS/1 users.

### IBM PS/1 Hardware

The PS/1 system comes in four models – two diskette-only models, and two hard-disk models, both with

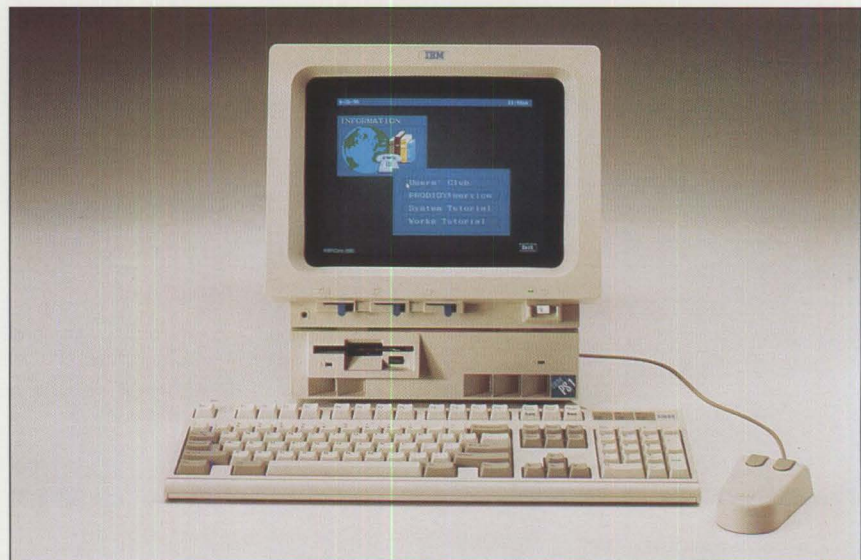
black-and-white or color displays. This section explores the standard and optional hardware features available with these PS/1 systems.

**Packaging:** The PS/1 comes with everything you need to use your PS/1 productively and quickly. Upon opening the box, the first thing you see is the small parts, cords, and documentation placed in special pockets in the styrofoam packing that protects the top of the computer. The booklet containing setup instructions is already opened to the page needed to help you set up the system. Additional manuals for the system and application software are included elsewhere in the box. A packet of diskettes containing software is included with diskette-based models. Also included is an IBM Mouse, and a cord to connect the PS/1 to a telephone line. You also will find a small phone-line splitter, so both the telephone

and new PS/1 computer can be connected to a single phone jack in the wall.

Under the styrofoam cover containing the small pieces are the three main components of the PS/1 computer system. The largest component is the video display. Next to it is the PS/1 computer system unit, standing on its side. Also standing on its side is a long thin box containing the keyboard. Once the components are removed, you're ready for setup.

**Setting up the PS/1:** You can set up your PS/1 system in just minutes. Setting up a PS/1 is extremely easy for even the newest of computer users. The display can sit on top of the small computer system unit and has two connecting cables and a power cord extending out the back. The two cables connect to the computer system unit, and the





power cord plugs into the wall. The cables and plug are of different sizes and no tools are required to attach them, so connection is easy and foolproof.

Plug the small keyboard cable into the clearly-marked socket in the back of the computer; then do the same with the mouse cable. Finally, snap the telephone cord into the back of the computer system unit, and attach the other end to either the telephone wall jack directly or to the telephone splitter to keep your phone connected.

Then, if you're an experienced computer user, you're probably thinking, "now comes the hard part — formatting the hard disk, installing DOS, and loading some application software." Before the PS/1, these steps were indeed necessary. But not now! Your PS/1 system comes with its hard disk formatted, DOS installed, and a selection of applications already installed. So, once you've set up the four connecting cables, power cord, and telephone cord, the PS/1 is ready to use.

Press the power switch at the base of the display. The system comes to life with an attractive screen containing four graphical blocks (icons) representing the four functional areas of the system. You are ready to use your PS/1 system productively just minutes after unpacking.

**PS/1 System Unit Facts:** For most new users, the unpacking and setup are all they need to know to get started. For users who are more technically inclined, the following information is provided.

The first thing you notice about the PS/1 system unit is its compact size. It takes up only 10.8 by 13.8 inches of your desktop, and is only 3.3

inches high. It's not much bigger than a college dictionary, and not much heavier. The unit weighs in at only 9.25 pounds without a hard disk, and at only 10.5 pounds with one.

Even though the PS/1 system unit is small and lightweight, the case is made of sturdy metal and is very durable. The unit is painted an off-white (cream) color, and has a matte, textured finish similar to other personal computer systems from IBM. The overall impression of the unit is one of quality.



*The first thing you notice about the PS/1 is its compact size.*

The front of the unit contains a slot for the 3.5-inch diskette drive and a panel covering the hard disk drive. Bevels for ventilation give the unit a distinctive appearance.

All cables attach to the back of the PS/1. From left to right, the attachment plugs are the video display connector from the display, the power connector also from the display, a parallel connector for a printer, the keyboard connector, and the mouse connector. Just above the mouse connector is a place for a standard telephone cord to plug in.

The top, bottom, and sides of the system unit are all nicely painted and contain nothing to distract from the clean lines of the unit. Rubber feet are attached to the bottom to prevent the unit from scratching the desktop.

All PS/1 models use an 80286 microprocessor running at a reliable 10 Mhz with one wait state. The basic system unit without a hard disk includes 512 KB of random access memory (RAM), upgradable to 1 MB. The hard disk model of the PS/1 comes standard with 1 MB of RAM installed. Additional RAM can be added to the PS/1 system using Memory Expansion Adapter cards.

All PS/1 models contain a 1.44 MB, 3.5-inch diskette drive on the front left side. The basic PS/1 system unit has an empty slot available next to the diskette drive to accommodate either a second 1.44 MB, 3.5-inch diskette drive or a 30 MB hard disk drive. The hard disk system unit has a 30 MB hard disk drive already installed in the right disk slot. To help the novice user, the diskette drive is clearly marked A: and the hard disk drive is clearly marked C:.

All U.S. models come standard with a 2400-baud internal modem installed. Using this modem, every PS/1 user can access a wide selection of telecommunications services available in the industry.

**Inside the PS/1 System Unit:**

Getting inside the PS/1 system unit is extremely simple, although most users will never have to open it. The front panel snaps forward and can then be removed. Removing the top cover from the PS/1 is equally simple. Just depress a spring catch located at the top front and slide the cover forward before lifting it off. No tools are required.

With the top cover removed, you can see that the main system board covers the bottom of the unit. This board contains most of the electronic components needed by the system, including the microproces-



sor, 512 KB of RAM, 256 KB of read only memory (ROM), and the electronics for attaching the input/output (I/O) devices. The I/O devices supported as standard on the main system board are the disk drives, a Video Graphics Array (VGA) connector for the display, a parallel port commonly used to attach a printer, and an internal serial port that is occupied by the internal modem. Also supported are the keyboard and mouse devices.

Above the main system board and to the front of the unit are two disk drive slots. Removal and installation of disk drives are very straightforward. To remove a disk drive, detach the connector cable from the back of the drive and press the spring catch at the bottom of the drive while sliding the drive out of its slot. Installing a new disk drive is just as easy – simply slide the disk drive into its slot until the spring catch snaps into place, then attach the connector cable to the back of the drive.

The PS/1's main system board comes standard with 512 KB of RAM. An additional 512 KB of RAM, contained on a small memory card, is optional on the basic unit, and standard on the hard disk unit. This card is located at the front of the unit, just below the disk drives, and is easily removed or installed. To help remove the tight-fitting card, two small arms are attached to each side of the memory card. By pulling outward on these arms, you can carefully pull the card out of its socket. To install the optional additional RAM in the basic unit, simply plug the small memory card into its slot.

Above the main system board and behind the diskette drive (A:) is the internal modem card. There is a

small space above this card for attaching an optional audio and joystick adapter card. In the middle back of the system unit is a power attachment card that connects to the display unit, which contains the main power supply.

Everything inside the system unit shows the careful engineering and design that went into the PS/1. This completes our tour of the inside of the PS/1 system unit.

You can replace the top cover and front panel as quickly as you removed them. Simply slide the top cover to the back of the unit until the spring catch at the front snaps into place. Then set the bottom edge of the front panel in place and snap the top into place by pressing both sides.

### *The color display has 256 different colors in 320 x 200 mode.*

**The PS/1 Display Unit:** A PS/1 system comes with either a black-and-white or color display. Either display has an attractive wedge-like cover with a rectangular base that fits squarely on top of the PS/1 system unit to give both units the appearance of one combined unit. When set on a desk in front of the operator, the screen tilts slightly upward to provide the best viewing angle. If the display is set on a shelf above the level of the keyboard, a prop can be moved into place on the bottom of the display to give the display screen an almost-vertical angle.

Both displays have 12-inch screens with 36-mm dot separation to give the viewer a sharp and crisp display. The displays are driven by the VGA circuitry on the main system board. Having VGA capability has become very popular in the personal computer industry, and most software packages today can take advantage of this superior display standard. In displaying graphics on the screen using the VGA standard, 640 x 480 different pixel elements can be individually addressed. The color display has 256 different colors in 320 x 200 mode, and the black-and-white display has 64 different shades of gray.

The display unit contains parts that are used by the entire PS/1 system. The power supply for the entire system is contained within the covers of the display unit. The main on/off switch for the system is a white push button located at the lower right side of the display. Pressing this button powers up both the display and the system unit. This design means that only one cord and power socket are needed for the PS/1 system.

The display unit also contains a small audio speaker. The speaker is located at the top rear portion of the display unit, and produces clean and clear audio signals. For individuals who prefer to hear the audio in private, a headphone jack is provided on the front of the display.

Three sliding blue levers are conveniently located at the lower front portion of the display head. These are clearly marked and permit the operator to adjust the speaker sound volume, and the brightness and contrast of the display.

**The PS/1 Keyboard:** The keyboard is one of the most important



parts of any computer system. IBM made sure that the PS/1 was designed with a high-quality, full-function keyboard that will make communicating with the system a dream. The PS/1 keyboard uses the same 101-key layout that has become the standard on most IBM personal systems. The size and placement of the keys and indicator lights are identical with other IBM enhanced keyboards. The bottom of the keyboard has rubber pads to protect the surface of the desk and two adjustable feet to vary the tilt of the keyboard.

The PS/1 designers made some improvements on the standard keyboard for the home user. The overall size of the keyboard has been reduced to better fit on a small desk or table. This change was engineered without reducing the key size or placement by simply trimming the excess border that surrounds other larger keyboards. This careful trimming also reduced the weight of the keyboard to just over two pounds, making it lighter to hold or use on your lap.


Another improvement with the PS/1 keyboard is the cord connecting it to the system unit. Instead of a tightly coiled cord that seems to have a mind of its own, the cord is small, light, and is easy to hide from view.

The feel of the PS/1 keyboard is crisp and positive like other IBM personal system keyboards. I've used my PS/1 for several weeks now and sense little difference between the touch of its keyboard and the keyboard of my PS/2 system at work.

Every PS/1 system comes standard with an IBM two-button mouse. The software included with the PS/1

takes advantage of the mouse as an input device although keyboard alternatives are available for those individuals not comfortable using a mouse.

**Optional 5.25-Inch Diskettes:** If you frequently need to access 5.25-inch diskettes, an optional diskette drive may be attached to the main system unit. With the optional diskette drive in place, either 3.5-inch diskettes or 5.25-inch diskettes may be used. Either a 360 KB or 1.2 MB 5.25-inch diskette drive can be attached to the PS/1.



*Every PS/1 system comes standard with an IBM two-button mouse.*

The optional 5.25-inch diskette drive can be attached quickly without any tools. The optional diskette drive is approximately the same size and shape as the system unit and attaches directly below it. To attach the optional diskette drive, remove the covers from the system unit and a small panel in the base of the system unit. The system unit snaps into place on top of the 5.25-inch diskette drive. A small power cable feeds up through the bottom of the main system unit and attaches to the internal modem card. The flat ribbon cable that attaches the 3.5-inch diskette drive to the system board is replaced with a three-way cable that connects both diskettes to the system board. Simply replace the various covers and you have a flexible unit that supports both 3.5-inch and 5.25-inch diskettes.

**Optional Expansion Unit:** For most PS/1 users, the standard system, perhaps with a printer attached, will satisfy most of their needs for years to come. But for users who want to attach additional adapter cards to the PS/1, an optional expansion unit is available.

The expansion unit is as compact as the main system unit. It has three empty slots for the addition of AT-standard adapter cards. Because the system unit and the matching expansion unit are compact, two adapter cards can be 11 inches long and the third card can be 9.5 inches long. These are common sizes for adapter cards; dozens of different cards currently available from various vendors have been tested and claimed compatible.

Attaching the optional expansion unit to the system unit is simple and can be done quickly without tools. First, remove the front panel and top cover from the main system unit. A new cover for the system unit is furnished with the expansion unit. Slide the new cover over the system unit until it snaps into place, and replace the front panel of the system unit. Next, place the expansion unit on top of the newly covered system unit. A connector card extends down through one opening in the top of the system unit and firmly anchors the expansion unit in place. After adding adapter cards to the expansion unit, the original top cover of the system unit now slides onto the expansion unit and covers it. Finally, snap the front panel for the expansion unit in place, and the PS/1 is ready to be used.

The overall footprint of the PS/1 remains the same with either or both optional units installed. What changes is the height of the system unit and its connected optional



unit(s). If you have either the 5.25-inch diskette unit or the expansion unit attached to the main system unit, the combined height is slightly over five inches. With both the 5.25-inch diskette drive and the expansion unit attached to the main system unit, the total height is about eight inches. This design permits your PS/1 system to grow without taking up more of your limited desktop space.

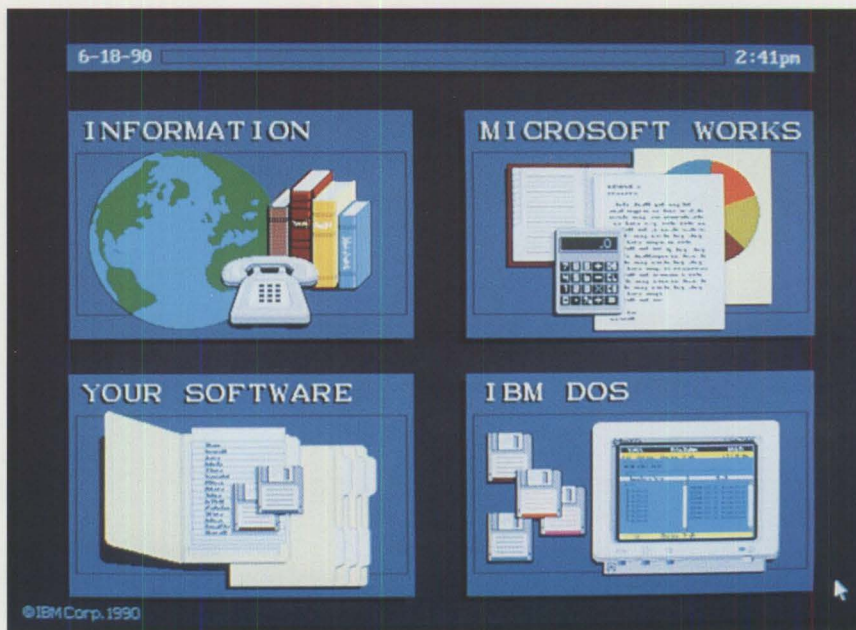
### IBM PS/1 Software

The PS/1 hardware system is an attractive personal computer for home use. But hardware alone does not constitute a complete computer system. Only when well-chosen software is combined with the PS/1 hardware, does the total system benefit the home user.

The PS/1 comes with many useful software packages already loaded and ready to use. Here's a description of that software and how it can benefit you.

**Main System Menu:** When you power-on your PS/1 system, the main system menu is displayed. The current date and time are shown across the top of the screen. Below this line are four icons representing the main menu items available:

- **INFORMATION** – This icon shows a telephone and books superimposed on the world. Select it to access the world of information available over the telephone and from the tutorials.
- **MICROSOFT® WORKS** – This icon shows a calculator, a printed letter, an arithmetical pie-chart, and an address book. Select it to use this popular software package for a variety of record-keeping functions in the home.
- **YOUR SOFTWARE** – Here are file folders and diskettes. Select



PS/1 Main System Menu

- this icon to run your favorite software packages, which you've added to your PS/1 system.
- **IBM DOS** – This icon shows diskettes and a computer display. Select it to use the DOS operating system to organize your diskette files and modify your PS/1 system.

To select any main system menu item, simply point to it with the mouse, then press and release the left mouse button. When you finish using that function, you will return again to the main menu. The menu serves as the anchor point of your software systems and ties the entire system together in one user-friendly package.

Let's look at each of these four areas and see what they provide.

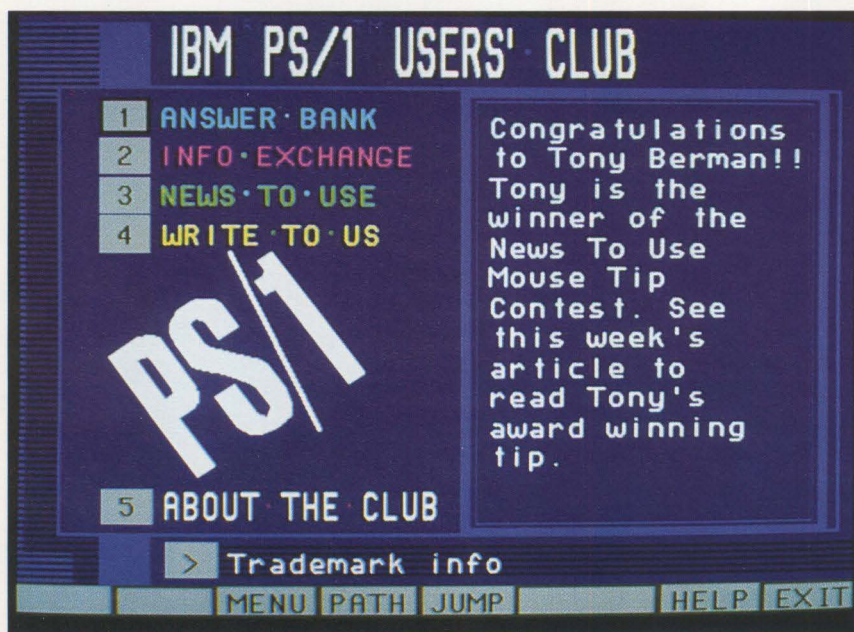
**Information:** The Information area gives you access to these information sources outside the home and within your PS/1 system:

- Users' Club
- PRODIGY
- System Tutorial
- Works Tutorial

The Users' Club is a service through which all PS/1 users can find and share PS/1 information via telephone line. Select this option and you will have access to a database of PS/1 questions and answers. You can exchange information about the PS/1 with other users, review the latest news and tips about your system, or send your questions to IBM experts for personal answers.

PRODIGY is a nationwide information service offered jointly by IBM and Sears. Gain access to this service, also, through a telephone connection. PRODIGY offers a method for sending messages to other PRODIGY users across the country. With it you can plan your vacations, manage your investments, and shop for many products at home. It offers





PS/1 Users' Club

you the latest news, sports, and weather and lets you read the advice of experts in a wide variety of areas.

The System Tutorial lets you spend time learning PS/1 hardware, software, and system care. The tutorial is broken down into small, easily understood pieces of information that can be reviewed at your convenience. If you are new to personal computers, this is an excellent way to start using the PS/1.

The Works Tutorial helps you understand and use Microsoft Works. It is broken into six separate topics, each having four to six training sessions. The sessions can be completed sequentially or at random.

**Microsoft Works:** The Microsoft Works software package contains four widely-used, integrated applications. Information may be shared among all four applications or used separately. All four applications are designed with similarities, making it

easy to use all four packages once you are familiar with one of them.

The most widely used PC application is word processing. Microsoft Works has an excellent word processing system for writing letters and reports. Students can use it for written homework assignments and term papers. It can be used to print envelopes and mailing labels, too. It can check spelling and offer synonyms from its built-in thesaurus.

Another application often used on personal computers is the spreadsheet. Microsoft Works contains an excellent spreadsheet for setting up family budgets, estimating taxes, balancing the checkbook, and tracking expenses. It is often said that a picture is worth a thousand words. With the Works spreadsheet, endless lists of numbers can be displayed in meaningful charts and graphs.

The third important application contained in Microsoft Works is a

database manager. You can use the database to keep track of addresses, birthdays, insurance records, greeting card lists, and other information. A query feature is provided to quickly access information stored in your database. A powerful reporting feature lets you prepare printed reports from the database.

The final function of Microsoft Works is a communications system that lets you link to other online information services and bulletin board systems. This facility can also be used to access your computer at work.

**Your Software:** The PS/1 is not limited by the software provided with the system, but can accommodate an extensive selection of software written for IBM personal computer systems. This includes special software for personal use in the home, or business software needed when bringing work home. Software written for IBM personal computers is the most widely available software in the world and most of it will run on your PS/1.

When adding software packages to your PS/1, follow the installation instructions that come with your software. Most software will set aside a subdirectory on your hard disk and load the software into that subdirectory. To use the software package, you must access the subdirectory and run one or more of the programs stored there. As you add more software to your PS/1 system, the number of subdirectories increases and the method of using the software varies. This all adds confusion for the new computer user.

The area called "Your Software" is designed to make it easier to access the subdirectories on your hard disk and to run the programs stored in



them. When you select "Your Software," the DOS root directory and the executable programs contained in that directory are displayed. Near the bottom of the screen are a number of labeled, closed folders, which contain the subdirectories stored on your hard disk. Selecting one of these folders will display the names of the executable software contained in that subdirectory. You can select a software program to run or open another folder to gain access to its software. In effect, This area of the PS/1 manages the directories and programs on your system and gives quick, easy access to both.

**IBM DOS:** IBM DOS is the last major section of your PS/1 software system. Here you have access to all the power and functions of the IBM DOS operating system. This version of DOS includes an excellent DOS Shell that makes using DOS easier and more powerful. When you select IBM DOS, you can access this DOS Shell with some special adaptations for the PS/1.

Use the DOS Shell to set the date and time, manage files, format diskettes, copy disks and files, backup and restore the system, and change the color of the Shell screens. Two special tasks are provided for the PS/1 system. The first lets you customize how your PS/1 system starts up when powering on. Also, if the hardware configuration is modified, the system may need to be updated to reflect these changes. Individuals comfortable with the many DOS commands can exit the DOS Shell and enter the commands directly into the system. For those who would like to try their hand at

programming, the PS/1 contains the BASIC programming language.

The PS/1 system comes with a wealth of ready-to-use software. A graphical main system menu pulls all of these software applications together in an easy-to-use format. Still, this software system is open-ended and expandable. It can run most of the thousands of software packages written to IBM personal computer standards.

*Send your question to  
IBM's PS/1 experts.*

### **IBM PS/1 Marketing, Service and Support**

The IBM PS/1 is distributed in the continental U.S. and Canada. To assure that the PS/1 is easily available, it will be sold by leading department stores and through IBM authorized dealers.

The PS/1 comes with a one-year warranty on all hardware and software. If it should fail within that period, here's how you correct the problem. A toll-free phone number is provided to assist you. If a part needs to be replaced, you may either take it to an authorized PS/1 remarketer for repair or use the PS/1 Express Maintenance service. Using the express maintenance service, a replacement part will be sent to you, arriving within 48 hours. Re-

turn the defective part to IBM and keep the new one.

Technical support for the PS/1 couldn't be easier. Simply connect your PS/1 to the phone line and contact the IBM PS/1 Users' Club. Look first to see if the question has been answered before, or send your question to IBM's PS/1 experts using the "Write to Us" option of the Users' Club. You will most likely have an answer within 24 hours.

### **Conclusion**

With the PS/1 system, IBM has introduced a new computer for the whole family. It includes reliable hardware that is expandable, and a software system designed to make the system easy to use, yet still powerful enough for experienced, technical users. And behind the PS/1 stands IBM's determination to provide the PS/1 with outstanding support and service programs.

### **ABOUT THE AUTHOR**

*Gene Barlow is the program manager within IBM responsible for interfacing with all IBM PC user groups. He was also the manager responsible for the creation of this publication, "IBM Personal Systems Technical Solutions," and its predecessor "Exchange." He has held numerous technical and management positions during the 28 years he has been with IBM. Gene received his B.A. degree in mathematics from Brigham Young University.*



# Using the IBM 4019 LaserPrinter Effectively

*This article was condensed from material supplied by the IBM printer development laboratory in Lexington, Kentucky.*

**There continues to be confusion about attaching the 4019 LaserPrinter to a PC or Personal System/2® (PS/2®) and how to effectively use the 4019 family of IBM LaserPrinters with various applications. This article attempts to dispel some of the confusion about the different emulation modes and data streams supported by the 4019 and details the function available with each emulation mode.**

To understand the capabilities of the 4019 LaserPrinter and what features and functions are available with each of its four modes of operation, you need to understand how applications communicate with and control the 4019. Communicating with, passing data to, and controlling the 4019 LaserPrinter, as with most personal computer printers, are done through a combination of commands and printable characters called the printer data stream. The data stream is passed from the application via a print driver to the printer.

The simplest way to think of a printer data stream is as a language that the application uses to talk to different printers. The precise language used, and the printer's ability to understand that language, determine exactly how the output will be printed.

## Printer Data Streams

Personal computer printer data streams consist of eight-bit bytes that contain a character to be printed, a segment of a graphic image, or an element of a printer command.

Most data streams used today are derived from the data stream developed for the IBM Graphics and Epson® FX-80 printers. This data stream was fairly simple because these printers offered only basic printing functions. But as printers with new capabilities were developed, companies found they needed to add commands that supported the new functions in the original IBM Graphics and Epson FX-80 data streams.

The original PC printers used control codes and escape sequences to control printer functions. Control codes are single-byte commands that specify printer functions, such as carriage return (X'0D'), line feed (X'0A'), and form feed (X'0C'). Escape sequences are multibyte commands that begin with an escape character (X'1B') followed by a character that defines the escape command, followed by optional data fields, if required. Escape F (X'1B46'), the command to end emphasized printing, is an example of an escape sequence.

Some newer printers have added control sequences to the existing control codes and escape sequences. Control sequences are more com-

plex and often carry multiple commands within one sequence. They consist of a three-byte mnemonic (Escape followed by the [ symbol, followed by a control sequence identifier byte), a two-byte length count – low byte, high byte – (allows the 4019 to determine the exact length of the sequence), followed by the bytes containing the commands. The Page Presentation Media command shown in Figure 1 is a good example of a control sequence.

## IBM Personal Printer Data Stream – (Extended IBM ASCII):

The data stream developed for the IBM Graphics Printer (originally called IBM ASCII) has evolved into the Personal Printer Data Stream (PPDS) used by the 4019 LaserPrinter. The evolution is fairly well defined, starting with the IBM Graphics Printer (Epson FX-80), adding the Proprinters™ I through III, followed by the Proprinter X24s, the Quietwriter® III (or Quickwriter®), and finally the 4019 and 4019-E LaserPrinter.

The PPDS data stream has three levels of function. Each level was designed to support newer printers as they were developed. Each offers enhanced capabilities over the previous level while maintaining most of the existing level's capabilities.

Level 1 is the basic level of PPDS. It supports the 9-pin and 24-pin Proprinter family of printers. Level 1 supports basic paper handling, limited font selection, and both

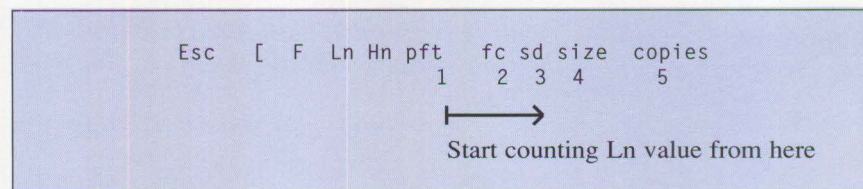


Figure 1. Example of the Page Presentation Media Command



9-wire and 24-wire bit-image graphics with multiple resolutions.

PPDS level 2 enhances the data stream to support the Quietwriter and Quickwriter III printers. This level of the PPDS data stream adds better font selection and control, enhanced paper handling with cut-sheet support, and better text justification and formatting.

PPDS level 3 supplies support for page printers such as the 4019. This level of PPDS allows random placement of text and graphic images, limited drawing by the printer, selection of paper orientation, and the use of typographic fonts.

**HP® Page Control Language Data Stream – PCL:** Hewlett-Packard® went through a similar evolution with its PCL printer data stream. The portion of the evolution that is of interest to IBM 4019 LaserPrinter users begins with the Hewlett-Packard LaserJet®. The LaserJet used a version of PCL called PCL-3, developed to be a language for office word processing. An enhancement to PCL-3, called PCL-4, includes all the function of PCL-3 and adds a set of page formatting commands. PCL-4 is used in the LaserJet+, LaserJet 500+, LaserJet II, LaserJet IID, and LaserJet IIP printers. Several versions of PCL-4 were developed as the data stream matured, and functions were added or removed to support different printer models. The recently announced LaserJet III uses an enhancement of PCL-4, called PCL-5. The PCL-5 data stream includes all the functions of PCL-3 and PCL-4, and adds new commands to support office publishing.

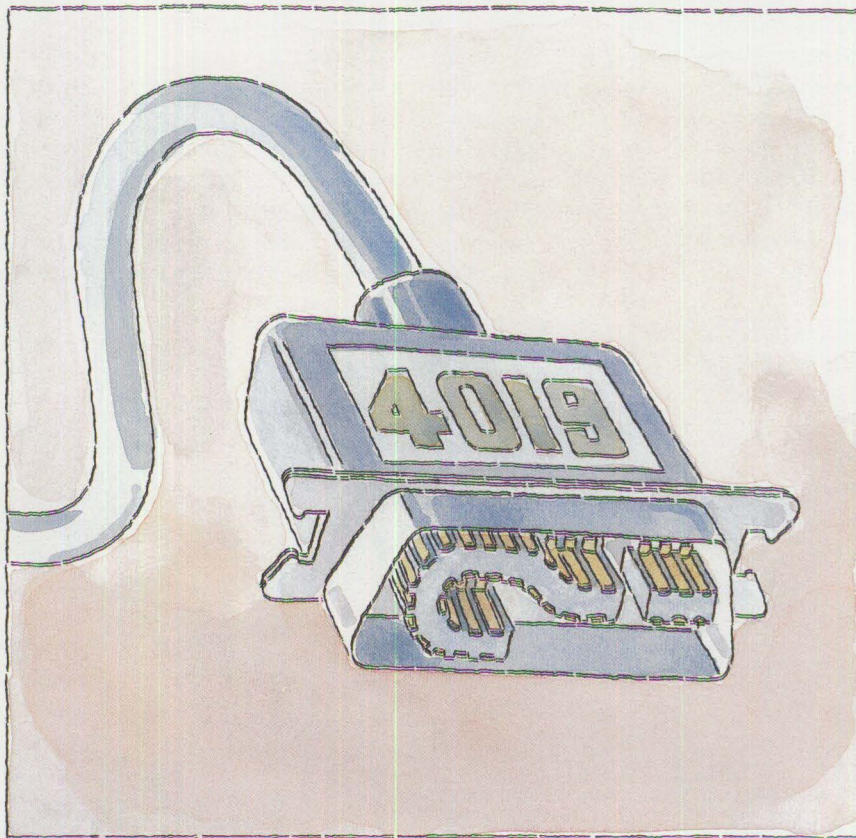
IBM selected PCL-4, the data stream developed for use in the HP LaserJet II, for use in the 4019

LaserPrinter. To the standard PCL-4 command set, IBM added a command that is not present in the LaserJet II. IBM incorporated the command used in the LaserJet 500+ and LaserJet IID for accessing its secondary sheet feed to allow 4019 users to take advantage of the optional second sheet feeder when running HP emulation. A second sheet feeder is not available on the HP LaserJet II, and therefore is not part of the LaserJet II command set.

**IBM and HP Graphics Language Data Stream – GL:** The IBM Graphics Language (or HPGL) is a data stream for creating plotter drawings. The IBM LaserPrinter accepts many IBM-GL commands and IBM-GL device-control instructions when running in plotter mode. The resulting images are similar or identical to those produced by many

IBM and Hewlett-Packard multipen graphic plotters. Most differences in output are due to the physical differences between the multipen plotters and the LaserPrinter.

The IBM 4019 LaserPrinter was designed to emulate the IBM 7372 and HP 7475A multipen plotters. The full set of 56 commands is included in the emulation. The IBM 6128 and HP 7550A commands that were added are: automatic paper loading and unloading to support the 4019's sheet feeding capability, and Download Character and Extra Space, to give additional label command capability to the 4019. The Download Character command allows the 4019 to download any character set available for the various supported plotters. The Extra Space command produces more attractive labels by adjusting the spacing be-





courier	courier-oblique
courier-bold	courier-boldoblique
helvetica	helvetica-oblique
helvetica-bold	helvetica-boldoblique
helvetica-narrow	helvetica-narrow-oblique
helvetica-narrow-bold	helvetica-narrow-boldoblique
times-roman	times-italic
times-bold	times-bolditalic
symbol	

Figure 2. PostScript Scalable Fonts

tween characters or words. The various character sets used in the plotter mode cannot be selected from the control panel and must be selected in the data stream.

Error-reporting features in the plotter data stream are available only when using the serial interface. When the 4019 is running via the parallel interface, responses normally sent to commands are suppressed.

**PostScript® Data Stream:** The newly announced PostScript Option for the 4019 and 4019-E is the

fourth data stream type supported by the family of LaserPrinters.

The PostScript data stream is unlike any of the three other data streams. This data stream is composed almost entirely of drawing commands, called the PostScript Page Description Language. These commands are analyzed and executed by the PostScript interpreter resident within a 4019 printer equipped with one of the 4019 PostScript Options. The 4019 PostScript Option supports the entire PostScript language as specified in the *PostScript Language Reference Manual* (Addison-Wesley Publishing Company,

Reading, Mass., 1985), as well as extensions to the language to support unique features of the 4019 LaserPrinter. These additional PostScript operators are detailed in the *IBM PostScript Option for the IBM LaserPrinter Technical Reference* (S544-4265), and apply specifically to Version 52.1 of the PostScript interpreter.

Because the PostScript interpreter attempts to recreate the output based entirely upon drawing commands rather than actual output, the PostScript interpreter is very strict about how drawing commands are structured. Failure to follow the PostScript Page Description Language specifications exactly when constructing the data stream, will result in an interpreter error and a failure to print the remaining pages. Attempts to send non-PostScript data streams to the 4019 while it is in PostScript emulation will result in the data stream being flushed from the printer and no output being printed. (This occurs with no error indication at either the 4019 or the application interface.

The PostScript Option for the IBM 4019 and 4019-E is available in two versions. The entry level has 17 scalable fonts, which are listed in Figure 2. (The text for this magazine shows examples of Times-Roman and Helvetica typefaces.)

The premium version has a font card that brings the total number of fonts to 39 scalable typefaces. The additional 22 fonts are also available as an upgrade to the basic PostScript Option. The additional fonts are listed in Figure 3.

There are several PostScript manuals that detail the PostScript Page Description Language. Those wishing to become knowledgeable in the

avantgarde-book	avantgarde-bookoblique
avantgarde-demi	avantgarde-demioblique
bookman-light	bookman-lightitalic
bookman-demi	bookman-demiitalic
helvetica-light	helvetica-lightoblique
helvetica-black	helvetica-blackoblique
new centuryschlbk-roman	new centuryschlbk-italic
new centuryschlbk-bold	new centuryschlbk-bolditalic
palatino-roman	palatino-italic
palatino-bold	palatino-bolditalic
zapfdingbats	zapfchancery-mediumitalic

Figure 3. Additional PostScript Fonts



PostScript language are encouraged to read one of the following manuals:

*PostScript Language Reference Manual*, Addison-Wesley Publishing Company, Reading, Mass., 1985

*PostScript Language Tutorial and Cookbook*, Addison-Wesley Publishing Company, Reading, Mass., 1985

*PostScript Language Program Design*, Addison-Wesley Publishing Company, Reading, Mass., 1988

### Selecting a Driver to Use for the 4019 in PPDS

Obviously, the best printer driver to use is the one written specifically to support the attached printer and use as many of its features as possible. Because many applications were written before the 4019 was announced, you are often required to select printer support other than 4019 when attaching a 4019 LaserPrinter to your system. To help you select the correct printer support for those applications, Figure 4 lists five additional printers that use the PPDS data stream and shows which features are available with each printer and its related driver.

Select the printer that most closely matches the functions required by your application. For word processing applications, the Quietwriter III supports almost all 4019 functions. For graphics, however, the choice is more difficult because none of the printers supports the Raster Image Graphics Command (300-pel All Points Addressable Graphics for IBM PPDS) and Rule and Fill. Using other printer drivers will result in lower resolution that may slightly distort the figure and take longer to print.

Other considerations must be understood when using 4019 line printer emulation. For the 4019, the maximum page length for a standard 8.5 x 11-inch page at six LPI is 64 lines in portrait, and 48 lines in landscape mode. Documents of other sizes are reduced by similar amounts. A number 10, legal-size envelope, for example, can have up to 21 lines. Generally the printable size is reduced by 1/2 inch from left to right, and by 1/3 inch from top to bottom when using the 4019 printer. As a result, your application must set the printer page length and right margin to the corrected values. For example, when using a line printer, a standard 8.5 x 11-inch page normally requires 65 line feeds and a form feed. Sending this data stream to a 4019 would cause an extra page to be fed, because the 4019 would form feed after the 64th line feed, whereas the application form feeds after the 66th line.

Failure to account for these differences can result in extra pages if the application includes a page eject command, or "page creeping" if the application does not include a page eject after the 66th line. With "page creeping," two lines from a 66-line-per-page document are moved to each subsequent page. The first page has the first 64 lines of page one printed on it. The second page has the last two lines from page one and the first 62 lines for page two — and this process continues for all following pages. The lines moved to the following page may be blank and can result in "extra blank pages." Envelopes that are set to a length greater than the printable area for the size being used will also show "creeping" or extra blank pages. The length that should be used for a standard U.S. number 10 envelope is 20 lines, but by using 19 lines the problem would defi-

nately be avoided. Figure 5 shows the PPDS printable areas.

Another common problem is "line wrapping," which occurs when lines are too long and extend into the unprintable area of the 4019. When the machine is set in the normal "Print Page Format," any characters that extend into the right unprintable area are moved to the left margin of the next line. This usually results in more lines on the page than expected and either extra blank pages or creeping text. It can be corrected by reformatting the output to the correct margins.

Poor print quality or blurred print can occur on the last printable line of a page, especially when using small typefaces. In portrait mode this is usually not a problem because few jobs print on the last line or use a small typeface. However, when printing a rotated job, such as a spreadsheet, the bottom of the page becomes the left side of the page and smaller typefaces, such as 17.1 Courier, are commonly used. To prevent this problem, a left margin of at least .33 inch should be used for landscape printing. This is three to four characters in 10 pitch, four characters in 12 pitch or proportional, and five to six characters in 17.1 pitch.

### Selecting a Driver to Use for the 4019 in HP-PCL

Selecting the best driver for use when running the 4019 with the HP-PCL data stream is similar to selecting a driver when running with the PPDS data stream. Choose the driver that closely matches the functions and features of the attached printer. However, when running in an emulation mode, there is a slight twist. Instead of matching the driver to the printer, match the driver to



	IBM 5152 and Epson FX-80	Proprinter	Proprinter I and II	Proprinter X24	Quick/Quiet- writer III	4019 Laser- Printer
10 pitch	X	X	X	X	X	X
10 pitch bold						X
12 pitch		X	X	X	X	X
17 pitch	X	X	X	X	X	X
Proportional printing			X	X	X	X
Variable point size fonts						X
Select fonts					X	X
Select font global					X	X
Character font image download		Note 1	Note 1	Note 1	X	X
Underscore		X	X	X	X	X
Overscore	X	X	X	X	X	X
Emphasized print	X	X	X	X	X	X
Double-strike print	X	X	X	X	X	X
Double-wide print	X	X	X	X	X	X
Set presentation highlights			X	X		X
Super/sub script	X	X	X	X	X	X
Select print mode	X	X	X	X	X	X
Set horizontal margins			X	X	X	X
Set vertical margins					X	X
Backspace	X	X	X	X	X	X
Portrait/landscape with rotated fonts						X
Page presentation media					X	X
Normal graphics 60 DPI	X	X	X	X	X	X
Dual-density 120 DPI	X	X	X	X	X	X
High-density 240 DPI	X	X	X	X	X	X
High-resolution 360 horizontal DPI				X		
Raster image 300 DPI						X
1/8, 1/6, 7/72 line space	X	X	X	X	X	X
Variable line spacing	X	X	X	X	X	X
Cursor movement						X
Rule and Fill						X
Include plot commands						X
Macros						X
Set initial conditions			Note 2	Note 2		X
<p>Note 1: This function is present, but may require a FontSet Driver to be compatible with the 4019 function, or may be incompatible.</p> <p>Note 2: This function is present in a limited form on the Proprinter III and the Proprinter X24E. It can be used to initialize the printer, but not to select data stream.</p>						

Figure 4. Support for PPDS Functions on IBM Printers



Paper Number–Type	PPDS Portrait			PPDS Landscape		
	10 CPI	12 CPI	17.1 CPI	10 CPI	12 CPI	17.1 CPI
Letter	80chr x 64lns	96chr x 64lns	136chr x 64lns	106chr x 48lns	128chr x 48lns	182chr x 48lns
Legal	80chr x 82lns	96chr x 82lns	136chr x 82lns	136chr x 48lns	164chr x 48lns	233chr x 48lns
B5	66chr x 58lns	80chr x 58lns	114chr x 58lns	97chr x 40lns	117chr x 40lns	167chr x 40lns
A4*	80chr x 68lns	96chr x 68lns	136chr x 68lns	114chr x 48lns	136chr x 48lns	194chr x 48lns
Executive	67chr x 61lns	81chr x 61lns	115chr x 61lns	101chr x 40lns	122chr x 40lns	173chr x 40lns
<b>Envelope Number–Type</b>						
7 3/4 (Executive)				71chr x 20lns	86chr x 20lns	122chr x 20lns
9 (Enclosure)				85chr x 20lns	102chr x 20lns	146chr x 20lns
10 (Business)				91chr x 21lns	110chr x 21lns	156chr x 22lns
DL				83chr x 22lns	99chr x 22lns	142chr x 22lns
C5				86chr x 35lns	104chr x 35lns	148chr x 35lns
B5				95chr x 38lns	114chr x 38lns	162chr x 38lns
Other (Maximum)				136chr x 48lns	164chr x 48lns	233chr x 48lns
<p>* A recent change has been made in machines manufactured for European markets that allows 80-character lines to be printed on A4-size paper in IBM PPDS mode. Some U.S. manufactured machines still have a 77-character writing line.</p>						

Figure 5. PPDS Printable Area Definitions in Characters at Six Lines per Inch

the features supported by the emulation mode of the printer. For the IBM 4019, this usually means selecting the HP LaserJet Series II driver.

There are, however, some exceptions. If you need hardware options on the 4019 that are not available on the LaserJet Series II, you may need to use a driver for a LaserJet printer that supports the desired hardware. If, for example, you need the automatic envelope feeder, use the HP LaserJet Series IID driver rather than the Series II driver. Figure 6 shows the functions supported by the various models of the LaserJet printer and the 4019 LaserPrinter. From this table, select the HP driver that supports the most features you need. Remember, regardless of the functions checked for the HP printer chosen, every function that you want to use must be checked under the 4019 column as well. Remember too, selecting an HP driver for one function may

cause you to lose another. The HP LaserJet 500+ supports secondary sheetfeed, but not resident 10 pitch bold. The following 4019 functions are *not* available in PCL mode.

- Resident proportional fonts
- Resident 12-pitch font
- Overscore
- Emphasized print
- Double-strike print
- Double-wide print
- Set presentation highlights
- Rotation of soft fonts
- Support plot commands

In addition to those functions, there are some other differences between the HP LaserJet II and the 4019 LaserPrinter. These differences can affect printed output from applications that think they are printing to an HP LaserJet printer.

- The 4019 will feed and format:
  - B5 size international paper (size Parameter 12)
  - B5 size international envelopes (Parameter 99)
  - Size 9 U.S. enclosure envelopes (Parameter 89)
- When manually feeding odd-size papers and envelopes, the location of print on the paper may require different compensation than on the LaserJet II. (This is because of the 4019's edge-fed paper system.)
- The 4019 uses Courier 16.7 instead of Lineprinter 16.7, which is used on the LaserJet II.
- The 4019 does not accept HP plug-in font cartridges.
- The 4019 allows use of up to 99 different fonts per page. The Series II allows only 16 fonts on a page.



	LaserJet	LaserJet+	LaserJet 500+	LaserJet II	4019	LaserJet IID	LaserJet IIP
10 pitch	X	X	X	X	X	X	X
10 pitch bold				X	X	X	X
12 pitch					Note 3	X	X
16.7 pitch		X	X	X	X	X	X
Assorted italics/bold					Note 3	X	X
ASCII, Roman Extended System set		X	X	X	Note 4	X	X
Variable point size fonts		X	X	X	X	X	X
Select font ID#	X	X	X	X	X	X	X
Soft fonts		X	X	X	X	X	X
Auto rotate soft fonts							X
Underscore char+machine		X	X	X	X	X	X
Underscore char+machine+font				X	X	X	X
Executive-size paper				X	X	X	X
Half-line feed	X	X	X	X	X	X	X
Set margins	X	X	X	X	X	X	X
Set page length	X	X	X	X	Note 4	X	X
Set paper size				X	X	X	X
Select number of copies	X	X	X	X	Note 4	X	X
Select 7-bit symbol set				X	Note 4	X	X
Backspace		X	X	X	X	X	X
Portrait/landscape	X	X	X	X	X	X	X
Duplex print						X	
Secondary sheetfeed			X		X	X	X
Envelope feed option				Note 5	X	X	
Manual feed envelope				X	X	X	X
Graphics 75 to 300 DPI		X	X	X	X	X	X
Set lines per inch	X	X	X	X	X	X	X
Vertical/horizontal motion index	X	X	X	X	X	X	X
Cursor movement	Note 6	X	X	X	X	X	X
Rule and Fill		X	X	X	X	X	X
Macros		X	X	X	X	X	X
Set initial conditions					X		
<p>Note 3: This function is available in the IBM 4019 with the addition of a font card (P/N 1255846) to match the function of the HP IID, except for duplex capability.</p> <p>Note 4: The IBM 4019 honors this data stream function, but does not allow selection from the control panel as a default selection. The data stream does not allow permanent storage.</p> <p>Note 5: The LaserJet II has a 15-envelope tray that replaces the primary paper tray. A different driver may be needed to use the LaserJet II envelope feeder.</p> <p>Note 6: Reduced function on the LaserJet I.</p>							

Figure 6. PCL Functions Supported by the IBM 4019 LaserPrinter



Paper Number-Type	PCL Portrait			PCL Landscape		
	10 CPI	12 CPI	16.7 CPI	10 CPI	12 CPI	16.7 CPI
Letter	80chr x 60lns	96chr x 60lns	133chr x 60lns	100chr x 48lns	120chr x 48lns	167chr x 48lns
Legal	80chr x 78lns	96chr x 78lns	133chr x 78lns	130chr x 48lns	156chr x 48lns	217chr x 48lns
B5	66chr x 54lns	80chr x 54lns	111chr x 54lns	91chr x 40lns	109chr x 40lns	151chr x 40lns
A4	77chr x 64lns	93chr x 64lns	129chr x 64lns	107chr x 46lns	128chr x 46lns	178chr x 46lns
Executive	67chr x 57lns	81chr x 57lns	112chr x 57lns	95chr x 410lns	114chr x 40lns	158chr x 40lns
<b>Envelope Number-Type</b>						
7 3/4 (Executive)				65chr x 20lns	78chr x 20lns	108chr x 20lns
9 (Enclosure)				79chr x 20lns	94chr x 20lns	131chr x 20lns
10 (Business)				85chr x 21lns	102chr x 21lns	141chr x 21lns
DL				76chr x 22lns	91chr x 22lns	127chr x 22lns
C5				80chr x 35lns	96chr x 35lns	133chr x 35lns
B5				88chr x 38lns	106chr x 38lns	147chr x 38lns

Figure 7. PCL Default Margin Area Definitions in Characters at Six Lines perInch

- The 4019 can be reset with an INIT\* signal on the parallel interface cable.
- IBM serial support is limited to RS-232 signal interface. It does not support RS-422. IBM requires use of the DSR signal; HP does not.

*Note:* This could result in the computer or other hardware not supplying the DTR signal on pin 20 of its RS-232 serial output port (which becomes the DSR signal at the printer). It is imperative that this condition be corrected for proper operation of the IBM 4019 LaserPrinter.

- XON/XOFF is returned with two stop bits in all cases.
- ROBUST-XON is not supported.
- XOFF is transmitted only once.
- DTR inversion is not supported.

There are other hardware differences to consider when using PCL on the 4019. The default margins for an 8.5 x 11-inch page are .5 inch top and bottom, .17 inch left, and .33 inch right. To use these HP default margins, set the line

length to 80 characters in 10 pitch, 96 in 12 or proportional, and 132 in 16.7 pitch. Set the page length at 60 lines for six lines per inch or 80 for eight lines per inch. Reduce other size pages by comparable amounts.

Figure 7 lists the PCL default margin areas.

The unprintable areas of the page are actually slightly smaller than the margins. By sending the command to change margins, the page length can be increased to 10.6 inches or 63 lines for a standard U.S. sheet of paper. When printing in landscape, the default margins are .2 inch left and right, .17 inch top, and .33 inch bottom. The correct line settings become 106 characters in 10 pitch, 127 in 12 pitch or proportional, and 177 in 16.7 pitch.

PCL also has the problem of leaving unprinted information in printer memory if a form feed isn't issued at the end of a partial page. This can usually be corrected by setting "eject at end of job" to "ON" in the software. "Wait between pages" should be turned to "OFF."

The PCL data stream has the same limitation printing on the bottom line of a page as the PPDS data stream: blurring can occur with smaller fonts. This generally has less impact in PCL mode because the default margins are larger than the unprintable area for PPDS. If, however, you are printing a rotated spreadsheet, the bottom of the page becomes the left side of the page, and the default left margin is .2 inch. This is slightly less than the .33-inch margin recommended to avoid blurring, so this problem may occur less frequently than when using PPDS in landscape mode. To prevent the problem, use a left margin of at least .33 inch for landscape printing. This would be three to four characters in 10 pitch, four characters in 12 pitch or proportional, and five to six characters in 17.1 pitch.

Fonts designed for use in the PPDS mode are generally not usable in PCL mode, and vice-versa. Exceptions are specifically noted in the font descriptions. IBM 4019s will accept download or soft fonts available from companies other than



IBM. Those fonts, however, will not be rotated automatically when the machine is in an orientation different than the one for which the font was originally designed. For landscape printing, use only landscape soft fonts, and for portrait mode, only portrait soft fonts.

### PPDS Versus PCL: Which Data Stream to Use?

If you run only word processing and spreadsheet applications, PPDS is recommended. The PPDS data stream provides more resident typefaces (five) than are available in the PCL mode (three) and gives access to 12 pitch and proportional printing. This is true even when the actual 4019 driver is not in use. The Quietwriter III, Quickwriter, and Proprinters support 12 pitch. Quietwriter III, Quickwriter, or any Proprinter except the Proprinter I support proportional printing. Easy access to both envelope feed and secondary sheet feed is available when using Quickwriter or Quietwriter III drivers. This generally makes PPDS the better selection for this category.

If landscape printing is a requirement, the only PPDS driver you can use is the 4019 driver. An alternative is available in PCL mode using the HP LaserJet IID driver. However, the LaserJet IID has a larger selection of resident fonts than the 4019's LaserJet II emulation; if your application requires these fonts, your output may be affected. The same is true for the IID's duplex function. The font difference can be overcome by purchasing the HP Mode Font Card (P/N 1255846), which brings the font selection up to the level of the IID.

For graphics applications, the deciding factor is 4019 support. If the ap-

plications do not support the 4019, you must select from among the Quickwriter, Quietwriter III, Proprinter, or LaserJet Family. None of the IBM printers can print 300 x 300 pel-per-inch graphics. They are limited to the coarser 240 x 240 density. While the LaserJet family of printers does support 300 x 300 pel-per-inch graphics, the 4019 PPDS driver, if available, usually performs faster than the equivalent function in PCL.

The decision of which data stream to use depends on whether 4019 support is available in the application. If the 4019 is supported for most or all of your applications, use the 4019 in PPDS mode. Substitute the Quickwriter or Quietwriter III drivers when the 4019 driver is not supported but enhanced paper-handling options are required. If the 4019 is not supported, and graphics or advanced functions are required, the LaserJet II or IID is the better selection. Use the IID when the paper-handling options are a factor. Finally, where PostScript Page Description Language is required, the PostScript data stream and related PostScript driver are the only options.

### Plotter Emulation Using the IBM and HPGL Data Stream

When running the 4019 in plotter emulation using the IBM-GL data stream, required functions may vary between the plotter and the 4019. Some of these differences are based upon the differences in the hardware and some upon the emulation. Here are some of the differences to consider before using 4019 plotter emulation.

When there is insufficient memory available to process a plot in

300 x 300 pel mode, the 4019 will automatically switch to the coarser 150 x 150 pel mode.

In setting up the page for plotter emulation, it is important to honor the non-print areas. The margins used for an 8.5 x 11-inch page should be no greater than 7.573 x 10.240 inches. The difference between page size and margins allows for the width of the widest pen line outside the edge of the defined page. The actual values vary depending on whether the plot is in merged or stand-alone mode. In stand-alone mode the limits are 7,910 by 10,619 plotter units, and in merged mode with autoscaling, they are 7,962 by 10,365 plotter units. In stand-alone mode these values are equivalent to being 64 pels in from the no-print borders in each direction. This value would apply to any paper size. When using the merged mode, the value must be increased to .22 inch to accommodate the command to change to merge mode in printer memory.

If automatic paper handling is not a requirement, select the emulations in the following order:

- IBM 7372/HP 7475A
- IBM 6128/HP 7550A
- IBM 6180/ HP Color Pro Plotter

Emulation of the IBM 6180 Pen Plotter and the HP Color Pro Plotter should work adequately, but may result in some 4019 functions not being used. Use of other plotter emulations may result in unacceptable results, insufficient memory, or slow performance.

The 4019 Plotter Emulation offers color emulation. Color emulation is supported by printing each color line with a different broken line pattern. The 4019 uses a unique pattern



for type 2 lines. All other lines conform to those produced by the line-type command. The use of broken lines includes text and legends. This may make the plot difficult to read. If the application supports the differentiation of colors, it may perform color plotting in a more pleasing manner than the 4019. For example, using the Lotus® single-color plot usually produces a better-looking plot than setting the 4019 into color emulation.

None of the drivers will take advantage of the 4019's envelope feeder or secondary sheet feeder. However, if the plotter mode is accessed from the merged mode of the printer, the PPDS commands for accessing these options can be used before entering plotter emulation. Note that as yet no applications have been written to take advantage of this paper feeding capability.

When in size B emulation, the plotter automatically scales the plot for the currently active page size.

### PostScript

When running the 4019 with the PostScript Page Description Language, the choice of drivers is more limited because of the strict requirements of the PostScript interpreter. As with plotter emulation, here are some thoughts when using PostScript on the 4019 LaserPrinter.

If a driver for the 4019 PostScript option is not available, consider the drivers for the Personal Page Printer models 030 and 031. These drivers may have additional fonts listed that are not available on the 4019, and you must take care not to specify any unsupported fonts. If drivers for the 4019 and IBM Personal Page

Printer Models 030 and 031 are not available, you may use drivers for the Texas Instruments 2108 Omnilaser or the Apple Laserwriter NT. The 2108 Omnilaser provides access to 13 of the 17 fonts offered in the entry PostScript option, and the Laserwriter NT provides access to 35 of the 39 fonts available in the Premium PostScript option.

*If a driver for the 4019 PostScript option is not available, consider the drivers for the Personal Page Printer models 030 and 031.*

Problems can be encountered when the application or driver sets the timeout parameters outside the valid range for the 4019 Postscript Option. For each type of printer, the job timeout, manual feed timeout, and wait timeout parameters of PostScript's **setdefaulttimeout** operator are valid only within certain ranges. The 4019's valid range for the job timeout parameter is zero (no timeout) or from 15 to 65535. The manual feed timeout valid range is zero to 255. The wait timeout range is zero (no wait) or from 15 and 255.

Additional memory can impact printer performance when using the PostScript data stream. With 1 MB memory option installed, the 4019 will approach six 8.5 x 11-inch pages per minute for less complex output, and output is restricted to a

paper size no larger than 8.5 x 11 inches. When the 2 MB memory option is installed, output will more consistently approach six pages per minute and more complex output can be built. The maximum page size increases to 8.5 x 14 inches, and there is more capacity for storing downloaded fonts. This eliminates the need to rescale the character and improves processing speed. With 3.5 MB memory installed, the 4019 can approach a speed of 10 pages per minute, build more complex output, and maintain the ability to print up to 8.5 x 14-inch pages. The font cache is also enlarged, which contributes to the improved performance. The effects of the memory on the performance for the 4019-E are similar for the 1 and 2 MB memory expansions; however, the lower throughput capability of the 4019-E limits the increased performance obtained when the 3.5 MB memory expansion is installed.

To access the envelope feeder and sheet feeder on the 4019, the specialty drivers written for the Personal Pageprinter sheet feeder may be used. However, the Personal Pageprinter uses a different numbering scheme than the 4019 LaserPrinter for the various paper feeders, which may result in feeding from the wrong paper source. If you are willing to experiment, and there is an IBM Personal Pageprinter driver available, accessing the 4019's various paper sources can be accomplished while in PostScript. You will have to try the various combinations available with the driver and determine which source on the 4019 will actually be used.



# Micro Channel Interface Chip Sets

*Jim Bringley  
IBM Corporation  
Boca Raton, Florida*

**IBM, Intel®, and Chips and Technologies are developing a family of chip sets that integrate the logic required to attach to a Micro Channel bus. This article discusses IBM's objectives in developing these chip sets and gives a brief explanation of one bus master design point. This is a high-level, functional overview of the design's external interfaces and a description of advanced internal features.**

**The Intel-marketed, 32-bit bus master design implements many advanced architectural features of the Micro Channel, such as 32-bit Streaming Data transfers, address and data parity, and a register set supporting the Subsystem Control Block architecture. The article shows the generic bus master design is applicable to a wide range of intelligent subsystem applications.**

Several groups within IBM have taken on the mission to assist independent option vendors in designing Micro Channel adapter cards. One strategy has been to supply to the industry general-purpose Micro Channel attachment chip sets for use on feature cards for Micro Channel computers. The goal was to integrate common Micro Channel functions in a generic set of components that could be used across many designs. With these components available from companies like Intel, and Chips and Technologies, adapter card builders can expect a lower learning curve for engineering de-

sign and verification, and can bring new feature cards to market faster.

One implementation of this strategy is an on-going project between IBM and Intel. IBM groups at Boca Raton, Florida, and Burlington, Vermont, have designed a family of Micro Channel attachment chip sets that address different design points for interfacing to the Micro Channel. These chip sets will be available as off-the-shelf components from Intel. For many applications, independent card vendors can use these chip sets in products for the rapidly growing market for Micro Channel adapter cards.

This discussion is about the first offerings in this family of Micro Channel chip sets, Intel's 82325 Micro Channel bus master interface chip set.

## Design Alternatives

Micro Channel architecture presents new design challenges for developers who are building feature and option cards for IBM PS/2 computers. The architecture offers different levels of interfaces, making possible a variety of adapter design points beyond programmed or memory-mapped I/O. Adapters can be DMA slave devices that take advantage of the system's DMA facilities, or full bus master designs for applications requiring higher performance and greater function. Another design option exists for Micro Channel data bus widths of 8, 16 or 32 bits. The deciding criteria for these design options are usually based on the desired price and performance targets for adapters and the applications that use them.

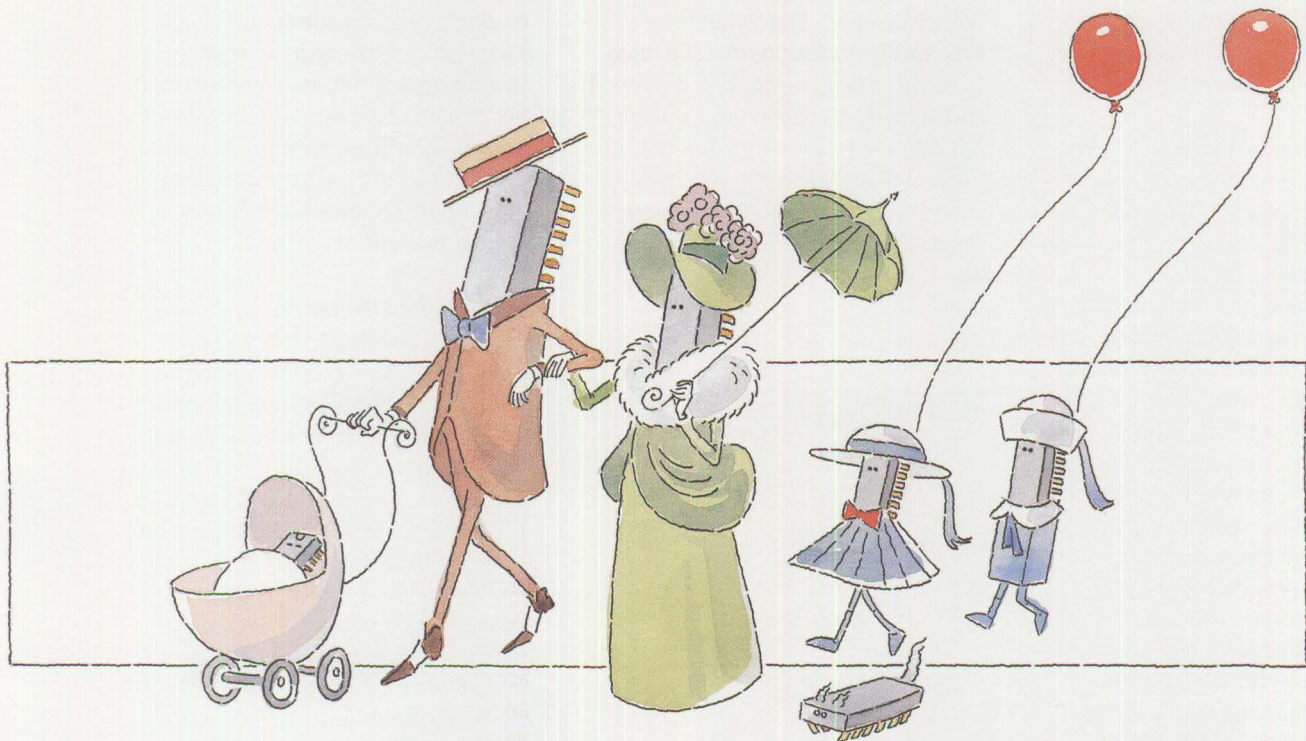
In addition to design-point alternatives, today's adapter card builders are faced with greater levels of com-

plexity. Personal systems are becoming faster and more reliable machines. Overall cycle times have shortened, resulting in an increase in system bandwidth. There are also now many more types of bus cycles, such as SET UP, ARBITRATION, DEFAULT, and SYNC EXTENDED. Programmable Option Select (POS) registers have replaced hardwired jumpers and switches for configuration and reliability purposes. These new complexities have raised the bar for experienced designers implementing new advanced Micro Channel functions.

Bus master designs offering advanced functions often require increased levels of integration in order to fit the function on one adapter card. Higher logic speeds are required for faster data transfers and arbitration cycles. Bus master designs require intelligent control over address, data, and control lines. Interfaces require the ability to address both system memory and I/O ports located throughout the system. At times, the logic required no longer lends itself to conventional programmable logic array (PLA) designs.

Designers are finding that they need standardized components for the bus interface sections of adapter designs. Use of such components ensures tested designs that meet all required interface standards. Availability of standardized components can reduce development costs and the learning curve required for implementing new adapter designs. At the same time, standard interface components can make it easier to take advantage of new functions available in Micro Channel architecture.





### A Family of Chips

A family of Micro Channel interface chip sets has been designed to provide standard front-end features for Micro Channel adapters. Three levels of interface design points exist: DMA slave, shallow bus master, and full bus master. These designs are required to be compatible with current and future implementations of IBM's Micro Channel and PS/2 systems.

The availability of these chips will help standardize Micro Channel compliance and reduce the verification efforts required for each new adapter design. The result will be more lower-cost, higher-function adapters available to provide advanced application solutions to end users. This discussion focuses on the most complex of the three design points: the full bus master. Fea-

tures of Intel's two-chip 82325 implementation are also discussed.

The 82325 bus master chip set was designed to meet three main objectives. The first objective was to design an intelligent interface for adapter cards containing outboard microprocessors in a wide range of applications. Next, the design minimizes the amount of available card "real estate" required to implement the bus-interface functions, leaving the greatest amount of card space available for card-level functional components. The last objective was to minimize the overall cost of components associated with Micro Channel interfacing. By meeting these objectives, the 82325 chip set enables developers to design low-cost, intelligent Micro Channel adapters for high-volume applications.

### 82325 Full Bus Master: A Two-Chip Set

The 82325 bus master interface consists of two chips that require one additional 245 transistor-transistor logic (TTL) driver for direct attachment to the Micro Channel. The 82325-MD chip is used for buffering and bus matching between the Micro Channel and the internal card interface. The 82325-MC chip contains most of the state machine control functions for the design. The Micro Channel address and data lines can be driven directly. The chip set meets the 24-mA current level required by the Micro Channel without external buffering. The chips are built in 1-micron complementary metal-oxide semiconductor (CMOS) technology and packaged in 124-pin and 144-pin plastic flat packs.




The 82325 chip set presents four external interfaces: the Micro Channel interface, the local processor interface, the local bypass interface, and the interchip bus interface.

The Micro Channel interface consists of the required signals for compliance in several configurations. Card designs can configure the components to have a data bus of 16 or 32 bits with two or four data-parity lines and one data-parity-enable control line. There are 32 address lines for full address space access with optional address parity support. Address and data lines are driven with 24-mA CMOS drivers for direct connection to the Micro Channel. The design can drive and receive control lines required for Micro Channel arbitration (ARB/GRANT, ARB0-3, PREEMPT) and data-transfer signals (SBHE, DS16RTN, DS32RTN, Streaming Data Rate SDR0 and SDR1, and BURST). One external 245 transceiver is required for control signals (CMD, ADL, S0,S1) with direction control provided by the 82325. Direct connection to the system set-up lines such as RESET, SFDBK, and CD\_SETUP is supported also.

The 82325 chip set supports the 32-bit Streaming Data Procedure. Thirty-two-bit bus master operations performed in 100 nanosecond (ns) cycles can significantly increase the data transfer rate of adapter cards using this design.

The local microprocessor interface is designed for direct attachment to one of two popular Intel microprocessor buses. Strapping the 186/376 pin determines the I/O pin definitions to form an 80186 or 80386SX-type local bus. (Intel's 80376 High Performance 32-Bit Embedded Processor offers the same external bus structure as the 80386SX proces-

sor.) The 186 bus offers a multiplexed address and data bus with a 16-bit data path and 21 address bits. The SX bus offers 16 data bits and 24 address lines, including the low- and high-byte enables. In both configurations the data transfer control lines such as MEM/IO, ALE(ADS) and \_WR\_RD (R/W) are supported, as are local interrupts and arbitration logic for local bus master control.



### *The 82325 chip set supports the 32-bit Streaming Data Procedure.*

In addition to the local microprocessor interface, the bus master design offers an option of adding a local bypass bus to adapter designs. The bypass bus isolation is done with the addition of two 245-bus transceivers on the subsystem side. Control of the local bypass bus is also provided by lines driven by the 82325. The bypass bus can be used in high bandwidth applications where a local subsystem needs to simultaneously transfer data elsewhere in the system while executing instructions locally. This feature can be used in most high-performance, intelligent controller applications such as DASD or LAN controllers.

The remaining pins on the bus master chip set are used to form interchip signals between the two modules. This interface consists of a 9-bit bidirectional command/data bus that is used for communications between chip set modules. Com-

mand and data exchanges are controlled by two data flow control clocks, a D clock for data, and a C clock for commands. The remainder of the signals are used for either Micro Channel bus requests, 245 direction control, or chip test signals. The test signals are for manufacturing purposes and are normally not used in operational mode.

### **Internal Functions**

Internally, the 82325 bus master chip set contains many required functions for Micro Channel interfacing. These functions include full POS register support, arbitration, and the address decoding logic required for memory and I/O slave operations. In addition, the chip contains a programmable DMA channel and port, and a set of special-function registers for the exchange of Subsystem Control Block (SCB) information. The DMA and SCB functions allow the 82325 chip set to act as the interface logic between the PS/2 host system and the intelligent subsystem implemented on the adapter card.

### **DMA Port and Channel**

Two features for direct memory access between system and I/O subsystem address spaces have been designed into the 82325. The first is called the DMA port. It allows the local microprocessor subsystem to transfer up to four bytes of data to or from system memory. By programming the Port Control, Address, Data, and Start registers, read and write operations to and from system memory or I/O space are performed. The Control register contains programmable bits for counts of one through four bytes in auto-increment or non-increment addressing modes. Completion status for a DMA port operation can be detected by the local microprocessor



via local interrupt or polling status bits in the Port Status register. Conditions such as bus timeouts, write checks, and data-size errors can be detected from the Port Status register bits.

DMA port operations can be programmed to operate on system data found in either system memory or I/O ports. This feature is intended as an independent port to the system for control blocks or parameter fetching across the Micro Channel. The local processor would act as either source or sink for the data of all port operations.

For transferring larger blocks of data, the 82325 contains a fully programmable DMA channel (DMAC). This channel can be run in one of two operating modes: direct or indirect. Direct program control mode of the DMA channel involves the use of two address registers: a 32-bit Channel System Address register (CSAR) and a 24-bit Channel Local Address register (CLAR). The Channel Byte Count register (CBCR) size is 24 bits, allowing up to 16 MB data transfers to be performed with one DMA channel sequence. A status register of the same format as the DMA port is provided, along with an expanded control register.

The DMA Channel Control register contains a field for the transfer mode of the current channel operation. Sixteen different modes are provided for transfers between various different source/destination pairs. Channel operations can move source data from system memory or I/O space, local memory, or both implicitly or explicitly addressed I/O ports on the local or bypass bus. The destination for these operations can be any of the same list. Completion status for the DMA channel is available via local interrupt or poll-

ing as in the DMA port. The large number of data-transfer modes supported provides extreme flexibility for card designers to optimize over all data flows in a wide range of applications.

The second mode of operation of the DMA channel is called indirect program control mode. This mode allows linked lists of control block information to be used to control the channel operations. Control blocks are constructed in local memory and can be chained together. The content of the control blocks is simple in form and resembles the data that would be programmed in the Channel registers. Linked-list DMAC operations are enabled by programming the Byte Count register to zero, enabling linked-list operations in the Control register and writing the local address of the first linked-list control block in the 24-bit Link List Pointer register.

*Once programmed, DMA channel operations are performed by the chip set independent of the local microprocessor.*

Once programmed, DMA channel operations are performed by the chip set independent of the local microprocessor. Channel operations are started via a local write to the DMAC START address. The optional linked-list structure allows the channel to fetch the next command block for multiple DMAC operations without processor

intervention. DMAC operations can be halted by writing the DMAC HALT address or programming the next linked-list control (LLC) bit to zero while the channel is running. Programmers familiar with linked-list DMA operations recognize the power and flexibility of such a DMA channel structure for ring buffer and frame-based subsystem applications. Through local driver code using the linked-list approach, a limitless number of logical DMA channels can be implemented.

### Special Register Set

Another powerful feature of the 82325 bus master design is a set of special-purpose registers designed for subsystem control block and command interchange. These registers allow an intelligent subsystem to communicate in a controlled manner with the host processor or other intelligent subsystems residing on the same Micro Channel. This set of registers is collectively called the SCB registers. The registers' base I/O addresses (IOAs) are set by the using adapter's I/O address and a 4-bit IOA field value is set in POS 2. Some fields and bits are specified for function, while others are left as subsystem-dependent. The implementation of some commands specified in the register definitions will differ between subsystems.

Six registers comprise the SCB hardware register support provided in the bus master chip set. These are the 32-bit Command Interface Port (CIP), the 8-bit Attention Port (AP), Subsystem Control Port (SCP), Interrupt Status Port (ISP), Command Busy/Status Port (CBSP), and a 16-bit Direct Interrupt Identification Port (DIIP).

Together the SCB registers form the basic hardware support required to



implement subsystem control between intelligent bus controllers. Driver code written to use these registers can implement a fast and efficient means of command and data communications for virtually any type of intelligent subsystem. A detailed discussion of SCB architecture can be found in the article "SCB - An Architecture for Micro Channel Bus Masters" in Issue 4, 1989 (G325-5004) of this magazine.

### Working Together

In summary, the 82325 bus master chip set described here offers designers an integrated solution for high-function, intelligent interfaces to the Micro Channel. Intelligent subsystems can be built using the bus master front end to directly attach to the Micro Channel. Local-side processing power can vary based on the local processor selection of a 186, 376, or 386SX. Bypass bus support allows high-throughput subsystems to continue outboard processing while transferring data to and from the system or channel memory. These functions are now integrated into two plastic flat packs that require less than four square inches on an adapter card.

The full bus master components contain the required standard Micro Channel features such as POS registers and arbitration state machines for using bus masters. In addition, they contain powerful state machine

designs for data transfers using the 32-bit Streaming Data protocol. The internal features of the DMA port and DMA channel allow flexible subsystem designs. Efficient small block data and command transfers can be accomplished via the DMA port, whereas larger data transfers are handled by the programmable DMA channel with linked-list command chaining. Data can be moved from local memory or I/O space to system or channel memory and I/O space depending on the card implementation.



*The full bus master chip set will reduce the design complexity, card real estate, and cost of implementing high-function adapters for Micro Channel systems.*

Intelligent subsystems can use the SCB registers to implement interlocked command interfaces. System and subsystem processors can communicate command and data information for specific subsystem tasks. General control fields are available

to implement subsystem specific commands.

The 82325 bus master chip set is one way to connect intelligent bus master subsystems to the Micro Channel and presents an integration of functions that will be commonly used in Micro Channel designs. The 82325 design will set a base example for other similar Micro Channel designs with different back-end interfaces for different applications. The full bus master chip set will reduce the design complexity, card real estate, and cost of implementing high-function adapters for Micro Channel systems. With the availability of such integrated designs from Intel, bus master designs for the Micro Channel will be much easier to implement.

### ABOUT THE AUTHOR

*Jim Bringley joined IBM in 1984 and currently manages the I/O VLSI Development department for IBM's Entry Systems Division. His previous experience includes work on the development of I/O processors and local area network (LAN) adapters for the AS/400™ and 9370 mid-range products. Jim earned his B. S. and M. S. in electrical engineering at the State University of New York at Buffalo.*



# Token Ring Bus Master LAN Adapters

Ariel Gomez-Ortigoza  
Lantana Technology Inc.  
San Diego, California

**This article discusses constraints to accurately appraising LAN adapter performance under different system architectures. Properly configured benchmarks demonstrate that Micro Channel-based bus master adapters comprise the best architecture for environments demanding high data transfer rates.**

Bus master adapters improve the throughput of the input/output subsystem of a microcomputer by as much as 40 percent, and are increasingly critical to local area networks that demand high rates of data transfer. Yet bus master LAN controllers are mostly underutilized in installed microcomputer networks, and "traditional" benchmark tools fail to reflect how bus masters affect performance.

Consider a typical example of what will be the normal operating environment of corporate America during the 1990s:

- An extensive and populated token ring network consists of several servers with hundreds of users sharing local and remote resources.
- This internetwork includes file and print servers with gigabytes of storage space and multiple printing capabilities. Communications servers support local and remote bridging and filtering functions, as well as electronic-mail services between different rings and gateway services to

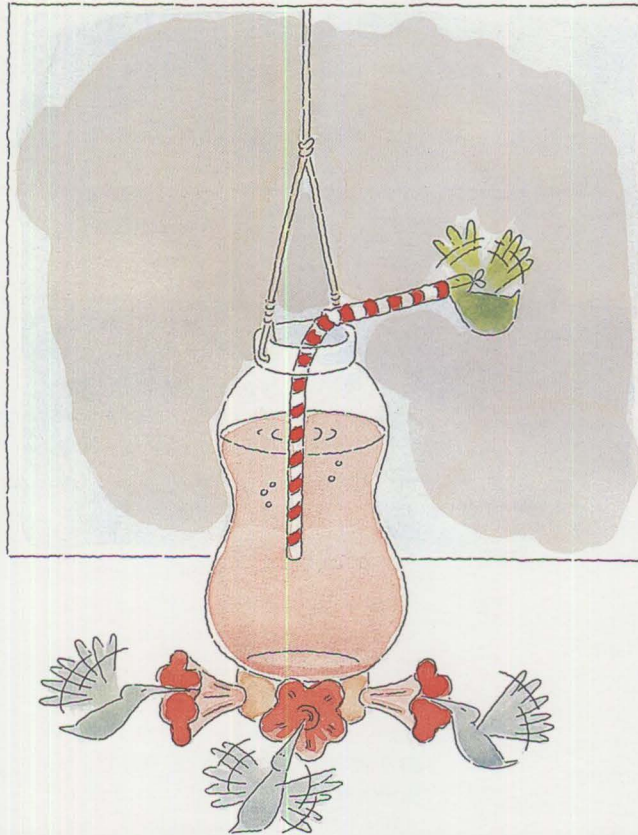
minicomputers and mainframes throughout the country. Also part of the network are data base servers with on-line data, which users, company-wide, continually access, update, and share.

- Each workstation is based on 32-bit microprocessors. At some sites, RISC technology is also used. Advanced bus architectures – such as the PS/2 Micro Channel, or comparable systems that allow multiprocessing – are supported on all nodes.
- The servers and workstations both run operating systems capable of delivering multitasking and multiuser services, including OS/2, LAN Manager, and AIX®. Each demands user-friendly, window-based applications and advanced graphics from Presentation Manager™ and CAD Graphic tools, and each

concurrently performs multiple applications while maintaining a session to download and access the mainframe.

Increasingly prevalent, this environment challenges corporate users with complex questions. What type of network and system should the user install in anticipation of this environment? How might the user prevent obsolescence and bottlenecks as technology advances? What working benchmarks might be established to effectively appraise adapter performance?

Lantana Technology Inc. is the developer and marketer of several bus master LAN adapters for ISA and Micro Channel computers. It is Lantana's experience that a bus master token ring adapter is the best architecture for environments that demand high data transfer rates.





## What is a Token Ring Bus Master?

A bus master adapter is a peripheral card that users add to a microcomputer system. It features a microprocessor that is dedicated to executing fast I/O data transfers. That microprocessor complements the central processor installed on the system motherboard. It enables the adapter to take control of the data transfer and operate as an independent peer of the central processor. Simply, this is a peer-to-peer configuration that improves adapter throughput – especially in environments that demand high data rates – by freeing the central processor for other tasks.

Further, a bus arbitration scheme enables the bus master adapter to share control of the bus with the host and several other bus masters running in the same host. Bus arbitration supports a preemption method and a fairness algorithm for equitable sharing of the bus.

In addition, bus arbitration schemes enable distributed multiprocessing, such as a communications protocol between bus masters.

The current 16 megabits-per-second (Mbps) token ring platform is the fastest LAN topology of the three most popular standards, transferring more data in less time than networks configured around the 10 Mbps Ethernet®, 2.5 Mbps ARC-net™, and 4 Mbps token ring standards.

In addition, token ring's advanced management functions equip the network administrator with tools that facilitate network management, especially when complemented with bus master capabilities.

The bus channel and the LAN adapter I/O throughput capacity are the two most critical architectural constraints limiting network performance.

Figure 1 demonstrates how the bus architecture selected for servers and workstations acting as nodes determines network performance. The bus frequency in MHz, and the maximum bandwidth in bits generate the theoretical maximum throughput of different competing architectures, as measured in megabytes per second.

Figure 1 presents ideal values that are actually far from “real world” results, which are subject to such factors as network loading and the number of users. However, this figure enables a comparison of the different system architectures. Similarly, the limitations and range of operation of each bus architecture may also be analyzed.

Working with these ideal values, the ISA non-bus-master architecture is severely limited in performance expansion.

Similarly, history demonstrates that the I/O subsystem on the LAN adapter is also a significant bottleneck.

Also limited is the performance of an ISA bus master adapter. Figure 1 reports a deceptively high theoretical throughput that is unattainable in the real world, because an ISA bus master is not a “true” bus master; it cannot preempt and must instead request the bus. The more realistic measure of the 16-bit ISA adapter is the DMA value.

Maximum Throughput (Megabytes per second)							
Architecture	ISA	ISA <sup>2</sup>	MC	MC	AMC <sup>4</sup>	EISA	NuBus
Clock frequency (MHz) <sup>1</sup>	4.77	8.33	10	10	10	8	10
Bandwidth (in bits)	8	16	16	32	32	32	32
DMA (MB/sec)	0.954	1.6	5	10		5-8 <sup>5</sup>	
DMS bus master (MB/sec)	N/A	16.7 <sup>3</sup>	10	20	160	33	37.5

1. MC is an asynchronous bus, and clock frequency is for reference only.  
 2. Data assumes an 80286 processor.  
 3. This value is attainable only on a bus master adapter. Otherwise, the result is the same as for DMA.  
 4. AMC = Advanced Micro Channel architecture.  
 5. Depends on EISA type A, B, or C.  
 N/A = not available in this platform.

Figure 1. Maximum Throughput



Data Transfer							
Time required to transfer 100 bytes (adjusted value)							
Processor	286	286	286	386	386	386	68030
Architecture	ISA	ISA	MC	MC	AMC	EISA	NuBus
Clock frequency (MHz)	4.77	8.33	10	10	10	8	10
Bandwidth (in bits)	8	16	16	32	32	32	32
I/O-mapped	463.31	25.6	20.48	13.7	13.7	N/S	N/S
Memory-mapped	192.24	25.6	20.48	10.7	10.7	N/S	N/S
DMA	104.82	62.5	20	10	N/S	12.5	N/S
DMA bus master	N/A	5.98 <sup>1</sup>	10	5	0.625	3	2.67

1. This value is attainable only on a bus master adapter. Otherwise, the result is the same as for DMA.  
 N/A = not available in this platform.  
 N/S = not specified, as data depends on specific implementation.

Figure 2. Data Transfer

Bus master functions were first included in the ISA IBM AT® architecture, which was introduced in 1984. However, this implementation was not a “true” bus master. It supported the operation of a single adapter with a dedicated Direct Memory Address (DMA) channel, but lacked a bus arbitration scheme. Multiple bus master adapters could not operate concurrently, arbitrate, or preempt for equitable sharing. The implementation did not have the intelligence to take control of the bus or transfer information between peer-to-peer bus master adapters.

The market responded accordingly. Developers introduced few adapters for ISA-based AT systems that used bus master adapters to boost system performance. This is beginning to change with the introduction of Micro Channel-based bus masters, as developers reconsider the bus master features of the AT.

However, the true benefit of the bus master is realized only with intelligent adapters for advanced architectures such as the Micro Channel,

which enables full bus master capabilities. Figure 2 compares the ideal time (in microseconds) required to transfer 100 bytes when using different architectures for network cards that are I/O-mapped or memory-mapped, or designs that use DMA or DMA bus masters.

Figure 3 interprets the same data from a second vantage point. This figure shows that the simplest card design – an I/O-mapped adapter in an ISA-based PC – could take 741 times longer to transfer data than the most advanced design, which is a DMA bus master Advanced Micro Channel-based 80386 system.

### Bus Master Performance Features

Bus master adapters, built around a dedicated microprocessor, increase throughput, improve concurrency, and reduce bus utilization and overhead by decreasing interrupt usage.

Technically, these benefits translate into more tasks executed simultaneously, a key requirement for multithreaded or multitasking operating systems such as OS/2.

To the user seeking to protect system investments, these technical benefits should ensure the selection and installation of devices that yield as much throughput as possible for current and future system configurations.

### Testing Considerations

Performance results obtained from a site testing bus master designs could be misleading. Normal testing conditions are configured for low-level demand, not high rates of data transfer in advanced environments – obscuring the benefits of bus master adapters.

Benchmark designers should instead bear in mind the old golden rule that the system is only as fast as the slowest of its parts. The difference between “ideal” and “real world” results is a function of the most limiting part of the system, not necessarily the technology of the LAN. Advanced architectures, workstations, operating systems, and applications must be properly configured to ensure they are not the factors that inhibit the performance of a bus master.



Data Transfer							
Time required to transfer 100 bytes (adjusted value) <sup>1</sup>							
Processor	286	286	286	386	386	386	68030
Architecture	ISA	ISA	MC	MC	AMC	EISA	NuBus
Clock frequency (MHz)	4.77	8.33	10	10	10	8	10
Bandwidth (bits)	8	16	16	32	32	32	32
I/O-mapped	741	41	33	22	22	N/S	N/S
Memory-mapped	307	41	33	17	17	N/S	N/S
DMA	167	100	32	16	N/S	20	N/S
DMA bus master	N/A	10 <sup>2</sup>	16	8	1	5	4

1. Fastest implementation – AMC bus master – is used as a reference value.  
2. This value is attainable only on a bus master adapter; otherwise, the result is the same as for DMA.  
N/A = not available in this platform.  
N/S = not specified, as data depends on specific implementation.

Figure 3. Data Transfer

A low throughput workstation such as a non-bus-master, ISA-based PC or AT biases the yield of the complete network performance test. It is also true that in trying to exercise a bus master Micro Channel architecture adapter on a server, several very fast bus master workstations should be used to produce the most accurate indicator of benchmark performance.

Otherwise, the bus master card out-runs and is hindered by the ISA-based workstations. This mistake is common, as most benchmark testing of networks typically involves no more than six ISA workstations.

Benchmark results also depend on whether the system is running software capable of delivering all the throughput. There are many layers of protocol stacks that need to be considered; the two that most affect the testing results are the efficiency of the LAN drivers and the service and communications protocols.

A well-written LAN driver can make 100 percent difference in testing results. Also, bus master adapt-

ers are mostly used in multitasking and multiuser environments, where adapters operating in parallel run multiple tasks under different applications. The faster the bus is utilized, the faster it may be used by the host for other tasks, which is the environment where a token ring bus master adapter will make a mark.

### Additional Considerations

Benchmark performance is also affected by other factors, such as memory buffers on the server, storage speed when the testing involves read-and-write operations, workstation channel architecture, the software protocol stacks involved, the number of workstations, and multitasking functions of the operating system. The tougher the testing environment, the better it is for benchmarking bus master performance.

### Conclusions

In the experience of Lantana, testing Micro Channel architecture bus master adapters under adequate conditions improves performance 20 to 40 percent over non-bus-master adapters. The results vary depend-

ing on the speeds of the server and workstations, as well as the data rate demands of the applications being run.

Today's enterprise-wide networks are environments in which every user needs such multiple services as mainframe access, the ability to run several applications concurrently, and the use of presentation graphics and windows. At the same time, the user must be connected to the LAN and internetwork, and be able to share peripherals. It is in this type of installation that advanced architectures such as Micro Channel are most appropriate, and high-speed token ring bus master adapters out-perform standard LAN adapters.

### ABOUT THE AUTHOR

*Ariel Gomez-Ortigoza is president of Lantana Technology Inc., a company that designs, manufactures, and markets local area networking products.*

*Lantana Technology Inc.  
(619) 565-6400*



## Extension of Wiring Rules for 4-Mbit/s Token Ring Using UTP Lobes

Robert D. Love and Thomas Toher  
IBM Corporation  
Research Triangle Park,  
North Carolina

IBM's published rules for 4-Mbit/s token ring on unshielded telephone wire provide two fixed wiring configurations that are adequate for many but not all customer installations. This article presents guidelines that extend those rules and maximize flexibility when using unshielded twisted pair (UTP). Formulas are given for extending UTP configurations beyond two wiring closets without repeaters, and for trading off distance between wiring closets for additional lobe length.

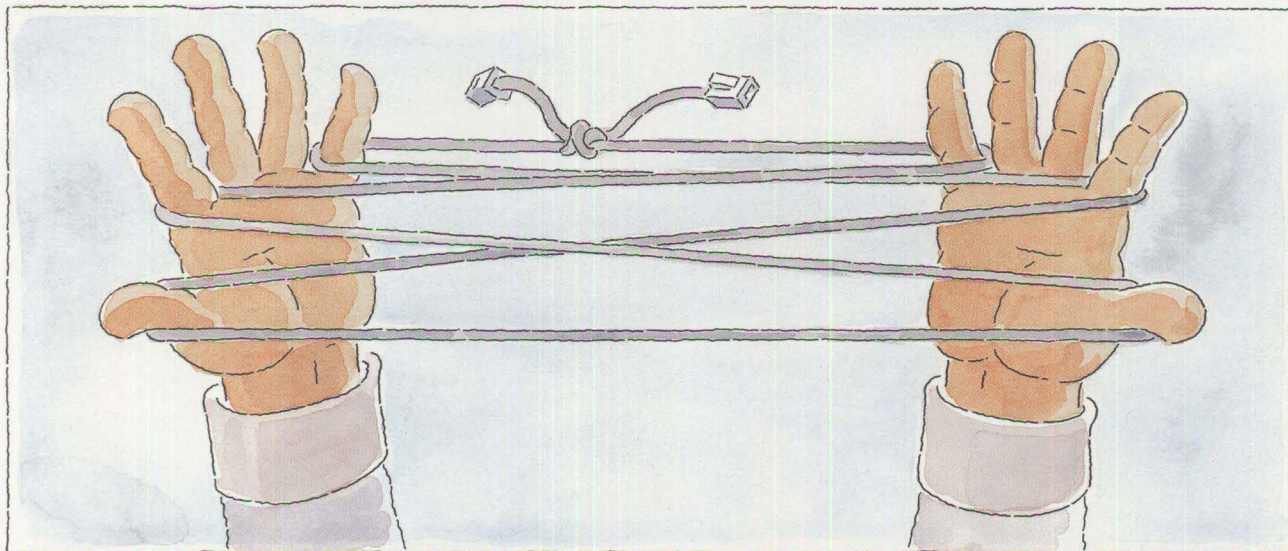
In 1985, IBM introduced its 4-Mbit/s Token-Ring Network and simultaneously announced the capability of using unshielded twisted-pair (UTP) telephone wire as an alternative to the IBM Cabling System's shielded data-grade media. The advantages of using shielded data-grade media with its superior transmission and noise immunity characteristics were apparent; networks using UTP media offered reduced performance and attachment capability. It was expected, therefore, that telephone wire would be an interim solution used primarily for initial trials before installation of the superior data-grade media (IBM cable types 1, 2, and 9). IBM's UTP offering, therefore, was tailored to support 4-Mbit/s token ring with simple, easy-to-apply wiring rules.

However, with the tremendous success of token ring, some users prefer to stay with UTP as the media of choice for 4-Mbit/s operation. This market segment needs more flexible rules to allow installations

using UTP to go beyond the original, simplified configurations without sacrificing network integrity.

The basic wiring rules for 4-Mbit/s token-ring operation on UTP are presented in *IBM Token-Ring Network Telephone Twisted-Pair Media Guide* (GA27-3714). The rules can be summarized as follows:

- 1) The maximum number of stations allowed on a ring is 72.
- 2) Each station using UTP must have a Type 3 Media Filter installed at the station.
- 3) If IBM 8218 or IBM 8219 repeaters are used, the ring can be regarded as consisting of a sequence of segments, with the repeaters at the boundaries of each segment. All maximum allowable length calculations are determined separately for each ring segment.
- 4) All single-wiring-closet rings and single-wiring-closet segments [that is, segments with all IBM 8228





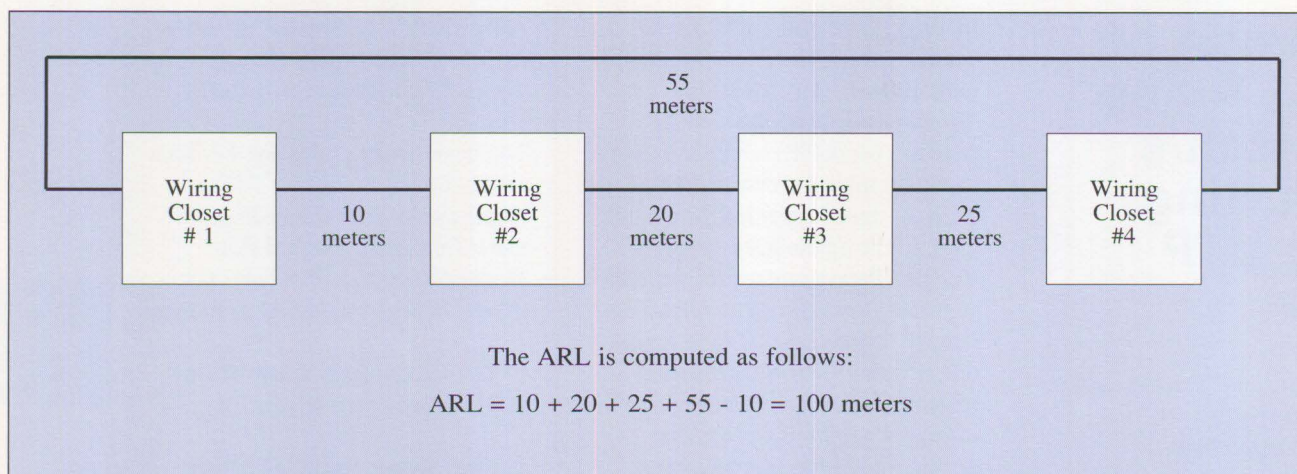


Figure 1. Applying the ARL Formula

Multistation Access Units (MAUs) contained in a single wiring closet], have a maximum allowable lobe length of 100 meters. All IBM 8218 and IBM 8219 repeaters bounding the single closet segment must be within 200 meters of the wiring closet that contains the IBM 8228 MAUs.

5) For two-wiring-closet rings and two-wiring-closet segments, the maximum lobe length is 45 meters of UTP, and the maximum separation between wiring closets without repeaters is 120 meters of IBM Cabling System Type 1 cable. Repeaters must be within 90 meters of the associated MAUs.

6) The maximum copper cabling distance between two pairs of 8218 repeaters is 600 meters of type 1 cable.

7) The maximum fiber cabling distance between two pairs of 8219 repeaters is 2 km of type 5 or 62.5/125 optical fiber cable.

8) Whenever 8218, 8219, or 8220 repeater/converters are used, a Data Grade Media-to-Type 3 Filter must be used with them, as described in

the *IBM Token-Ring Network Telephone Twisted-Pair Media Guide*.

9) All copper cabling between wiring closets must be 150-ohm, shielded, data-grade media.

10) Token rings that contain UTP lobes can be bridged to 4- or 16-Mbit/s rings with no change in rules 1 through 9.

In addition, there are rules for use of data-grade lobes on rings that also contain UTP lobes. The rules can be summarized as follows:

1) Each device attached to a data-grade media lobe must use a Data Grade Media-to-Type-3 Filter. This device is bilateral in operation, so that either end may be attached to the device side of the lobe.

2) You are allowed to use type 1 or type 2 cable that is twice the allowable UTP lobe length as determined by rules 1 through 9.

### Extended Flexibility for 4-Mbit/s Token Rings with UTP Lobes

The existing rules have sufficed for the design of token rings with UTP lobes for over four years. Generally, the rules are adequate and meet the requirements of providing reasonably flexible and easily workable guidelines. However, there is a continual trickle of requests to extend these rules by allowing for a trade-off of inter-wiring closet length and number of 8228s for lobe distance, and rules for configuring a ring with more than two wiring closets between repeaters on 4-Mbit/s rings. The following rules allow for this additional flexibility.

Starting with a two-wiring closet configuration with nine 8228 MAUs, 120 meters between wiring closets and 45 meter lobes, modify the allowable maximum lobe distance based on the following rules and trade-offs.

1) No configurations are permitted that have a computed maximum allowable lobe length of less than 45 meters.



Longest allowed lobe length (LALL) in meters

$$\text{LALL} = 45 + 0.3 (120 - \text{ARL}) + 4 (9 - \#8228\text{s}) - 2.5 (\#WC - 2) \quad (1)$$

Figure 2. Equation 1.

2) Use Adjusted Ring Length (ARL) for configurations with more than two wiring closets.

The ARL is the total cable length in the main ring path between wiring closets (assuming type 1 cable is used) less the shortest inter-wiring closet link. For rings without repeaters or converters, the ARL should not exceed 120 meters.

(For more information about ARL see *IBM Token-Ring Network Introduction and Planning Guide* (GA27-3677), pages 2 through 16.)

An example of a ring with components in four wiring closets is shown in Figure 1.

3) Add 0.3 meters to the maximum allowable lobe length (45 meters)

for each meter the ARL is below the 120-meter maximum.

4) Add 4 meters to the maximum allowable lobe length (45 meters) for each 8228 MAU less than the maximum allowed number of nine.

5) Subtract 2.5 meters from the maximum allowable lobe length (45 meters) for each wiring closet (WC) greater than two. This subtraction accounts for the increase in the main ring path because of the addition of two 8-foot patch cables needed to go into and out of the additional wiring closets.

Using these rules, the original guideline given for a two wiring closet ring can be modified:

- The longest allowable lobe length is 45 meters

- The maximum distance between wiring closets is 120 meters
- The maximum number of 8228s is nine
- The maximum number of wiring closets is two

By trading off distance between wiring closets, maximum number of 8228s, and the number of wiring closets for lobe length, we arrive at a rule with greater flexibility.

The new rule expressed in equation form is shown in Figure 2. The configuration must allow a maximum lobe length of at least 45 meters.

Applying this formula to a ring that spans four wiring closets with seven MAUs and has an ARL of 100 meters, the formula yields a maximum allowable lobe length of 54 meters (Figure 3).

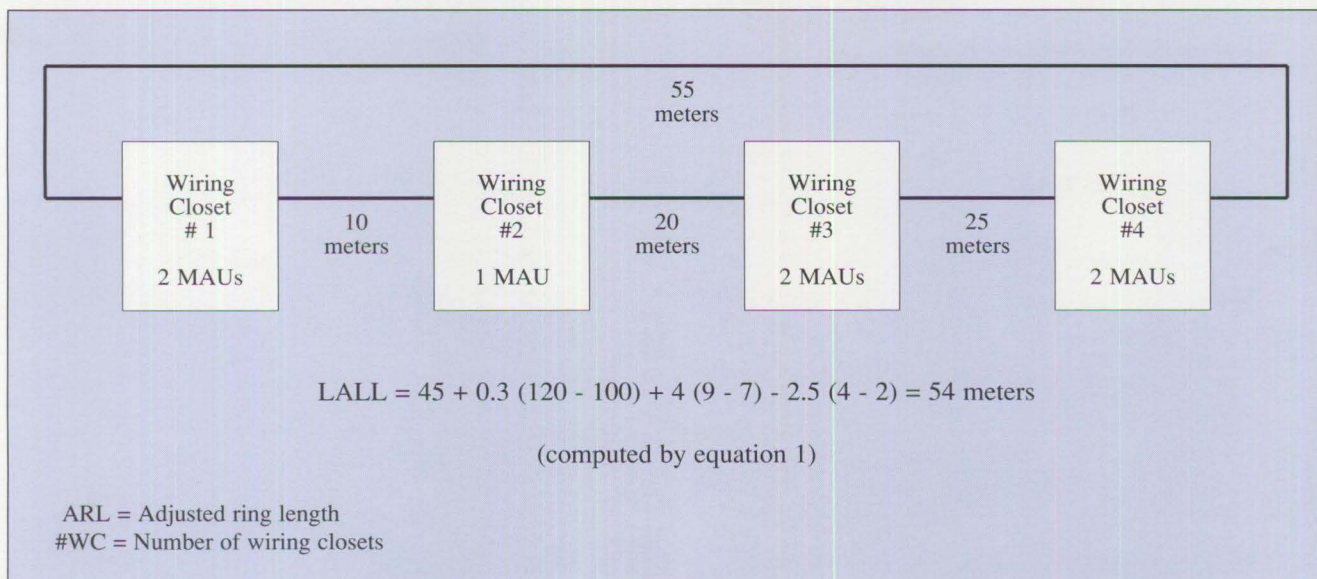


Figure 3. Applying Equation 1



$$LALL = 45 + 0.3 (120 - MRPL) + 4 (9 - \#8228s) - 2.5 (\#WC - 2) \quad (2)$$

MRPL = Main ring path length with the segment

Figure 4. Equation 2.

These guidelines also allow for extending the wiring rules to include the case where repeaters and converters (8218 Copper Repeaters, 8219 Optical Fiber Repeaters, and 8220 Optical Fiber Converters) are used.

When a token ring contains repeaters or converters, maximum cable distances are computed independently for each segment bounded by repeaters or converters. The formulas presented above can be used for the segments with a minor adjustment. Instead of using ARL, the main ring path length (MRPL) is now used within the segment. Each copper segment in the ring must be checked.

When using 8218s or 8219s, the MRPL is the length of the cabling

from the first 8228 in the segment to the last 8228 in the segment. Cable lengths from the input and output repeaters to the 8228s within the segment are not part of the MRPL. The allowable length of these segments is computed as twice the allowable UTP lobe length computed for the segment. Equation 1 is rewritten using MRPL instead of ARL as equation 2 (shown in Figure 4).

Consider the token ring (Figure 5) with two 8228 MAUs in wiring closets 1, 2, 3, and 4, and no 8228s in wiring closet #5. This ring is divided into two segments. The first contains wiring closet #1 and the 20-meter and 140-meter cabling lengths connecting that wiring closet to the 8218s that bound it. The longest allowable UTP lobe

length for a single wiring closet segment is 100 meters, and the longest allowable cabling run from the wiring closet to an 8218 repeater is  $2 \times 100 \text{ meters} = 200 \text{ meters}$ .

The second ring segment contains wiring closets 2, 3, and 4, and the wiring from wiring closet 4 to the 8218s in wiring closet #5. Because there are no 8228 MAUs in wiring closet #5, it is not counted as one of the wiring closets in the segment. The MRPL in this segment is  $30 + 40 = 70 \text{ meters}$ . From equation 2, the longest allowable lobe length is computed as 69.5 meters.

The longest allowable length from wiring closet #4 to the 8218s in wiring closet #5 is  $2 \times 69.5 = 139 \text{ meters}$ . Because the actual distance is 50 meters, the configuration is valid.

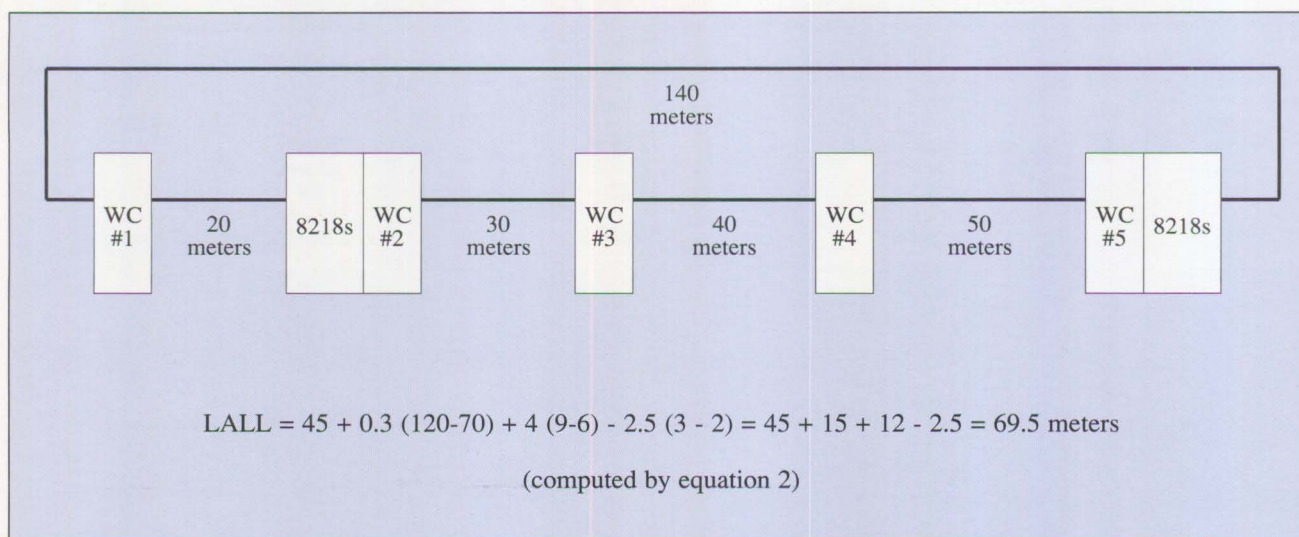


Figure 5. Applying Equation 2



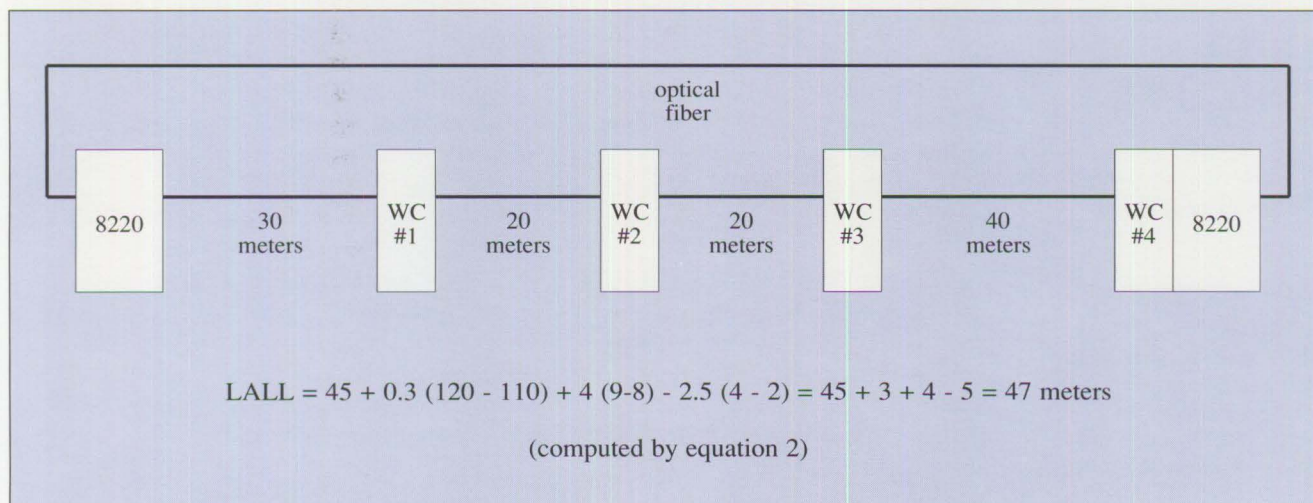


Figure 6. Applying Equation 2

When 8220 optical fiber converters are at one or both ends of the ring segment, the cabling between the 8220 and its adjacent 8228 within the segment is counted as a part of the MRPL. For example, in Figure 6, with two 8228s each in wiring closets 1 through 4, the MRPL includes the 30 meters from the first 8220 to wiring closet #1. Therefore,  $MRPL = 30 + 20 + 20 + 40 = 110$  meters. By equation 2, the longest allowable lobe length is 47 meters.

Optical-fiber cable can be used in the main ring path by bounding the optical-fiber segments with 8219 repeaters or 8220 converters. When using either 8219s and 8220s to drive 62.5/125 or 100/140 multi-mode optical-fiber cable, a distance of at least 2 kilometers can be supported. Detailed fiber-link budgets are presented in the publication *IBM Token-Ring Network Optical Fiber Cable Options* (GA27-3747).

Remember that when repeaters and converters are used, the total num-

ber of devices on the ring must still not exceed 72. Each 8218 and 8219 counts as one device towards that limit, and each 8220 counts as two devices towards that limit.

Using these rules, 4-Mbit/s token rings with UTP lobes can be configured with a maximum of flexibility while minimizing the number of repeaters and converters needed in the network.

#### ABOUT THE AUTHORS

*Robert D. Love is a senior engineer at IBM's Communication Systems Division in Research Triangle Park, North Carolina. He received his B.S.E.E. from Columbia University, and an M.S. in electro-physics from the Brooklyn Polytechnic Institute. He joined IBM in 1968, and presently works in the Future LAN Cabling System department, where he is working on enhancements to the IBM Cabling System. Bob is an expert on data transmission and*

*represents IBM on various standards bodies, including IEEE 802.5. He holds two chip patents and has published several articles on LAN transmission design, transmission comparisons, and token-ring design.*

*Thomas Toher is an advisory information developer at IBM's Communication Systems Division in Research Triangle Park, North Carolina. He received a B.A. and M.A. in English from Hobart College and Clark University, respectively. He joined IBM in 1984, and presently works in the Local Area Network Information Development department. Tom is the lead writer for all IBM Cabling System, IBM Token-Ring Network, and IBM PC Network information products produced at the Research Triangle Park laboratory. He is a frequent speaker on the subjects of cabling and planning for token-ring networks at GUIDE as well as other forums.*



## SCSI and DISK386.SYS

*Supplied by IBM technical staffs at Boca Raton, Florida, and Dallas, Texas*


**This is a description of the Update Program Driver, DISK386.SYS, for IBM SCSI adapters. This program should be installed only if the user is experiencing problems, and only after considering the performance implications.**

The IBM PS/2 Micro Channel SCSI adapters announced by IBM on March 20, 1990, are bus master adapters. Because bus master adapters read and write data directly to or from system memory, they do not use the Direct Memory Access (DMA) controller on the system board. This increases performance by allowing the system processor to perform other tasks while the SCSI bus master adapter transfers data.

The Update Program driver is provided in order to maintain compatibility with certain multitasking DOS applications or memory extenders that assume the fixed-disk drive will use the system DMA controller. When this driver is installed, it reserves a buffer in the first megabyte of system memory that serves as an intermediate location for data being transferred between the SCSI adapter and the requesting

application's buffer. While the driver allows you to run applications that assume the fixed-disk drive will use the system DMA controller, it may cause some performance degradation in file access time.

The Update Program driver should only be installed when applications providing DOS memory extensions fail to work properly when used with a PS/2 SCSI adapter. For example, an application may fail to load and return the user to the DOS prompt; or when attempting to access a SCSI device, the system may "hang" or the screen may go blank.



*This program should be installed only if the user is experiencing problems, and only after considering the performance implications.*

Specific examples of applications that do *not* need the Update Program driver are Windows™ 3.0, DESQview™ 386 with QEMM™ 5.0, 386 MAX™ 4.08, 386 Professional™ 4.08, and VM/386™. Older versions of these applications, and

other applications that use extended or expanded memory under DOS, may need the Update Program driver installed in order to operate properly.

IBM recommends that users update their applications to maximize overall system performance rather than install the Update Program driver. However, this program is made available to provide the option of maintaining compatibility with previously purchased applications.

These instructions also apply to the generic Update Program driver, GEN386.SYS, supplied on the option disks shipped with non-fixed disk SCSI options such as the IBM PS/2 CD-ROM Drive.

The following documents, which are shipped with the products, include instructions for the installation of DISK386.SYS:

*IBM Personal System/2 Micro Channel SCSI Adapter with Cache Installation Instructions* (page 12)

*IBM Personal System/2 Micro Channel SCSI Adapter Installation Instructions* (page 10)

*IBM Personal System/2 Model 65 SX Quick Reference* (page 74)

*IBM Personal System/2 Model 80 Quick Reference* (page 82)



## Operating System Platforms: A Business Perspective

*Pino Ferrari and Fernando Lopez  
IBM Corporation  
Boca Raton, Florida*

**With DOS and OS/2® Version 1.2 and MS® Windows™ 3.0 available now and OS/2 Version 2.0 on its way, deciding on an operating system platform is an important and complex task. This article, taking a business perspective, discusses the requirements for the next generation of operating systems, describes the technology, and compares the three platforms.**

In the good old days life was simple. If you had a PC, you could go out and buy applications (or write your own) without worrying about which operating system they had to run under. Effectively, there was only one operating system, and its name was DOS.

DOS is inexpensive, modest in memory requirements, straightforward to use, and it has a stable, flexible program interface. It was designed around the technology of the original IBM PC, and intended for a personal computing environment consisting of stand-alone PCs processing text data. That environment suited the needs of a wide variety of users, and the investment in DOS applications is measured in

millions of dollars. That investment must be protected, but the needs of businesses are expanding:

- It's difficult to share data among stand-alone personal systems. Most businesses need to put personal systems on a network so users can access data no matter where it resides. Many businesses need an enterprise-wide network with applications that can automatically and concurrently acquire needed data from multiple locations.
- Businesses want a more intuitive user interface. They want to reduce training and technical support costs, make end users more productive, and lower the barriers that prevent new users from using computers.
- Businesses need to process graphical data in applications that range from business charting to sophisticated desktop publishing. The technology is changing to make graphics processing an integral part of hardware and software design.

To meet these needs, an operating system platform must be easy to use, with windowing and an icon-based user interface. It must be capable of exploiting new hardware technology as it becomes available, while preserving previous application investments. And it must be capable of integrating desktop computing systems with the rest of an organization's information processing technology. This means it must be able to run multiple applications and multiple communications

protocols concurrently, and it must be able to perform multiple functions concurrently inside a single application.

Unfortunately, DOS doesn't meet these requirements. It was designed around the technology of the original PC, the Intel 8088/8086 microprocessor, and that technology imposes restrictions unacceptable in today's environment.

### The Technology

The 8088/8086 microprocessors have a 20-bit address bus, which means that they can never directly address more than 1 MB of RAM. Of this 1 MB, 384 KB is for internal system use, leaving only 640 KB available for the operating system and user applications.

*DOS was designed  
around the technology of  
the original PC, the Intel  
8088/8086  
microprocessor.*

The 8088/8086 microprocessors lack the hardware assist features necessary to properly handle more than one application at a time. And since DOS was designed around the 8088/8086, it is strictly a single-task, single-thread operating system; that is, it can maintain only one



program in memory at a time, and it services requests serially. Yet multitasking – the ability to maintain several applications in memory and switch among them so quickly that they seem to be running concurrently – is important in today's environment as a productivity enhancer. With a multitasking system, for example, you could run a long data base query, and at the same time enter data in a spreadsheet or update a document with a text processing program.

Programming techniques that provide a certain degree of multitasking under DOS have been developed, but require additional products (like Windows, DESQview™, and so on). Such techniques also add overhead, and they lack the safety features a true multitasking system provides.

*Multithreading* – the ability to run several parts of an application concurrently – allows more intelligent applications to be developed. For example, with applications running under a single-thread operating system like DOS, a user who must gather data from different sources (host machines, a LAN server, and so on) must know how to perform the data transfers, and in what sequence. A multithreading application can perform the transfers automatically and concurrently.

In 1982, Intel introduced the 80286™, a microprocessor with significantly enhanced function. The 80286 has two modes of operation: *real mode*, in which it simulates an 8088/8086, and a new mode called *protect mode*.

In protect mode, the 80286 has 24-bit addressing, which gives it access to 16 MB of RAM. In addition, protect mode includes memory protec-

tion and privileged instructions, which make true multitasking possible. It also has a mechanism called *virtual memory*, which allows the 80286 to run applications larger than the installed memory. The 80286 cannot, however, run real mode (DOS) applications when it's in protect mode.

*Multithreading – the ability to run several parts of an application concurrently – allows more intelligent applications to be developed.*

When the 80286 is powered up, it goes into real mode, in which it can run DOS applications. In real mode it acts like the 8088/8086, with the 640 KB memory limitation and the lack of hardware assisted multitasking functions. Software can switch the 80286 from real mode to protect mode, but the only way to switch it back to real mode is to reset the microprocessor. That makes it very difficult for an operating system to exploit the protect mode features of the 80286 while providing compatibility with real mode.

The 80286 provides many of the features necessary for an operating system that can satisfy today's business needs, but does not make those features available to DOS applications.

Then, in 1985, Intel announced the 80386™ microprocessor, and later the 80486™ microprocessor. The 386™ and 486™ have 32-bit ad-

ressing, so they can use up to 4 gigabytes of real memory (more memory than is available on the IBM 3090™), and up to 64 terabytes of virtual memory. Segmenting of 64 KB is supported for compatibility with programs designed for the 8088/8086 and 80286, but the 386 and 486 have linear addressing and can use segments as large as 4 gigabytes. Because a program "segment" can be as large as the real memory addressing capability of the microprocessor, programs no longer have to be collections of small pieces, and storage management is much easier for programmers.

In addition, the 386 and 486 microprocessors include a virtual memory mechanism called *paging*, similar to that used on mainframe computers. The paging hardware allows memory to be divided into 4 KB page frames. The contents of the page frames, called *pages*, are swapped between auxiliary storage and RAM as needed. Paging allows virtual and real memory management to be much more efficient.

As is the case with the 80286, the 386 and 486 come up in real mode, simulating the 8088/8086, and are capable of running DOS applications. Under software control, the 386 and 486 can be switched into a new mode called *virtual protected mode*.

In virtual protected mode, the 386 and 486 can run programs designed for the 80286 protect mode as well as programs designed to take advantage of 32-bit addressing, linear addressing, large virtual memory, and paging. But perhaps more significant to a business with an investment in DOS applications, the 386 and 486 can divide memory into a number of simulated 8086



machines, each of which acts like a separate computer. The 386 and 486 can then run multiple DOS applications concurrently, using hardware-assisted multitasking.

Microprocessor technology has evolved far beyond the simple, single-task, single-thread 8088/8086. Today's microprocessors can access large amounts of real memory and can use even larger amounts of virtual memory. They are capable of running several applications concurrently and executing several parts of the same application concurrently. They can run DOS-based applications, but DOS cannot exploit the power of this new technology. IBM and Microsoft realized that a new-generation operating system was needed, and OS/2 is the result.

### Changing Platforms: The Alternatives

Some businesses will stay with the stand-alone, text-based personal computing environments they have now, but many will want to realize the benefits of the new technology. These businesses, depending on their current situation and future business requirements, have three major alternatives: MS Windows Version 3.0 running on DOS, OS/2 Version 1.2, and soon, the 32-bit OS/2 (referred to hereafter as OS/2 Version 2.0.)

In any case, the costs of migrating must be analyzed very carefully, because they will far outweigh the costs of hardware and software. In fact, the hardware and software costs are comparable: the difference between an OS/2 solution and a Windows 3.0 solution comes down to the cost of 1 MB of memory, and (at list price) OS/2 costs only a few dollars more than Windows 3.0 plus

- Train technical staff
- Evaluate current application environment
- Select and test hardware platform
- Determine application availability
- Certify new applications
- Migrate in-house applications
- Prepare deployment plan
- Select pilot groups
- Migrate power users
- Train end users
- Share migration experiences
- Adjust plan as necessary

Figure 1. Migration Plan

DOS. The migration is the expensive part; some of the steps in a typical migration are shown in Figure 1.

**MS Windows Version 3.0:** If a business has a large installed base of 8088/8086 and 80286 machines, and does not need the function of OS/2 now or in the near future, MS Windows 3.0 may be an acceptable solution. Shipped as a single retail product that provides support for 8088/8086, 80286, 80386, and 80486 machines, Windows 3.0 provides many enhancements over previous versions.

The most obvious new features are the icon-based program manager that groups and starts both Windows applications and non-Windows applications, and the graphical file management interface. The windowing controls and the look-and-feel are similar to the OS/2 Version 1.2 Presentation Man-

ager™, although there are some small differences.

The major improvements are the ability of Windows 3.0 applications to access 16 MB of memory, the capability for running multiple DOS applications concurrently, and the use of paging for virtual memory. These improvements are, for the most part, incorporated in two new modes of operation.

MS Windows 3.0 provides three modes of operation: *Real mode*, *standard mode*, and *386 enhanced mode*. The program automatically starts up in the appropriate mode for the system, depending on the microprocessor and memory installed, but can be forced into a lesser mode by command line switches.

Real mode is the only one of the three modes capable of running



applications designed for Windows 2.X. It is similar to previous versions of MS Windows, so the major benefit is the new user interface. Real mode requires an 8088/8086 microprocessor or above and 640 KB of memory, and supports the LIM 4.0 Expanded Memory Standard. Multiple Windows applications can run concurrently using a technique known as *cooperative multitasking*.

In a cooperative multitasking environment, the tasks must be written to voluntarily and periodically give up control of the system to allow other tasks to run. Unless the other tasks are "well behaved," a Windows application has no guarantee that it will get any processor time within a given period. The technique works well, but can cause problems for programs (such as communications) that need regular processor attention. It does not protect tasks from one another as does a true, hardware-assisted, multitasking system.

Multiple DOS applications can be invoked within the Windows environment in real mode, but only one application can be resident and running. DOS applications are swapped out of memory to accommodate the last one launched, and when a DOS application is running, all Windows applications are suspended. Similarly, when a Windows application is running, all DOS applications are suspended.

Standard mode requires an 80286 microprocessor or above with at least 1 MB of memory. Expanded memory for DOS applications is supported only with physical expanded memory cards (not by emulating expanded memory using extended memory). However, standard mode provides a major en-

hancement to Windows: it exploits the 80286 protect mode to give Windows and Windows applications access to 16 MB of memory.

The technique used to give Windows applications access to extended memory (memory above 1 MB) is similar to the techniques used by DOS "extenders." It provides access to more memory and allows the user to run more programs, or to run programs more efficiently. However, it requires more careful application development, and the processor must be reset to real mode and switched back to protect mode. Also, as is the case with cooperative multitasking, the technique cannot protect one application from another.

---

*Windows applications  
operate in a cooperative  
multitasking environment  
in all modes, and  
because Windows 3.0 is  
based on DOS, it is a  
single-thread  
environment.*

An 80386 microprocessor or above with at least 2 MB of RAM is required for 386 enhanced mode. In addition to the new user interface and access to 16 MB of memory for Windows applications, 386 enhanced mode exploits the virtual protected mode of the 80386 and 80486 to provide multiple DOS environments for non-Windows applications. In addition, it uses the 386/486 paging feature to provide virtual memory support.

In 386 enhanced mode, multiple DOS applications run in a time-sliced environment. Each DOS application is given a fixed number of seconds to execute, then control is passed to the next application in a "round-robin" fashion.

However, even in 386 enhanced mode, Windows applications run in a cooperative multitasking environment. Further, DOS extender applications (such as Lotus 1-2-3™ Version 3.0) will not work in 386 enhanced mode, which introduces possible incompatibilities between DOS extenders and the Windows 3.0 extended memory manager, and contains other conflicts with DOS extenders.

MS Windows 3.0 can offer productivity gains through its graphical user interface (and may reduce training costs in the event of a future upgrade to OS/2). When run on an 80286, 80386, or 80486 machine, it provides multitasking benefits and improves program efficiency by allowing Windows applications access to up to 16 MB of memory. On an 80386 or 80486 machine, Windows also provides true multitasking of DOS applications. But it has limitations: Windows applications operate in a cooperative multitasking environment in all modes, and because Windows 3.0 is based on DOS, it is a single-thread environment.

**OS/2:** OS/2 requires an 80286, 80386, or 80486. OS/2 takes advantage of the large memory and hardware-assisted multitasking features of these microprocessors without losing DOS compatibility, and it has the rich function needed by applications critical to the operation of the business. Version 2.0 will exploit the 386/486 virtual protected mode to allow multitasking of



multiple DOS applications concurrently with applications written for the 80286 and 386/486. Yet businesses have been slow in acquiring OS/2 licenses. One reason is the vast installed base of 8088/8086-based and 80286-based systems, but there are other reasons, which, in our opinion, are becoming less and less valid:

- Between 1985 and the OS/2 announcement in 1987, memory cost quadrupled. However, the cost of memory has been declining steadily ever since, at about 30 percent annually.
- When OS/2 was announced, it provided programmers with a new set of functions, a new application interface, and a new user interface – in short, a new programming environment. Programmers were faced with the task of learning the new environment and producing applications that were competitive with their DOS-based counterparts. It has taken time, but large numbers of applications are now being written.

OS/2 has become a viable, cost-effective platform for taking advantage of the advances in desktop information processing technology. And it complies with IBM's Systems Application Architecture™, which aids system integration with other information processing technologies.

If a business has or is ready to migrate to 80386-based machines, and needs to move toward integrating desktop systems with the rest of the enterprise's information processing technology, OS/2 Version 1.2 (and, in particular, OS/2 Extended Edition) is the appropriate choice.

Version 1.2 is a 16-bit operating system. It improves user productivity

by providing a sophisticated but easy-to-use, icon-based, windowing, user interface. Version 1.2 protects the DOS application investment with a DOS compatibility mode for running DOS applications. Yet it takes full advantage of the 80286 and 80386 protect mode to provide multitasking and multithreading support for OS/2 applications. In addition to enhancing productivity, the use of protect mode enhances the integrity of the environment, which is vital when critical business applications move from host to desktop. Preemptive (as opposed to cooperative) multitasking and multithreading provide prioritized access to the system's resources, which is vital for communications applications.

*OS/2 has become a viable, cost-effective platform for taking advantage of the advances in desktop information processing technology.*

In addition, Version 1.2 includes a high performance file system (HPFS) which, unlike the File Allocation Table (FAT) system used by DOS, was designed for large direct access storage devices. HPFS is an installable file system that maximizes performance, minimizes dead space, has improved error processing, and includes extended file attributes. At the same time, OS/2 continues to support the FAT system for compatibility with other systems and for diskette use.

OS/2 Extended Edition Version 1.2 adds a powerful set of integrated communications functions, including the features required for a LAN-based, multiuser environment, and a relational data base manager. These communications and data management functions take advantage of protect mode to make OS/2 a rich, deployable platform that enables businesses to start capitalizing on application and system integration benefits.

If a business is ready to install 80386- or 80486-based systems, and has a large investment in DOS applications or a major requirement to continue using existing DOS applications, yet needs the rich function and system integration features of OS/2, Version 2.0 is the proper alternative.

OS/2 Version 2.0, according to a joint statement of direction by IBM and Microsoft, is scheduled for availability in the near future. It requires an 80386- or 80486-based machine and has all the function provided by Version 1.2, but exploits the features of the 386/486 virtual protected mode.

Version 2.0 can provide up to 16 concurrently running virtual DOS machines. Each DOS machine has the 640 KB limit, but because OS/2 emulates certain DOS functions (such as file I/O), the applications actually have more memory available to them than they would on their own DOS-based systems. Further, each DOS application can use the LIM 4.0 standard automatically, without the extra memory manager required in simple DOS environments. Also, OS/2 Version 2.0 permits DOS applications to be windowed and allows the user to copy data from one application to another.




In addition to enhancing DOS performance, OS/2 Version 2.0 runs applications designed for Version 1.X without changes, but provides a 32-bit application program interface for applications that exploit the 32-bit flat memory model, and uses the 386/486 paging feature to support virtual memory. Because this paging mechanism is so efficient, the system will run acceptably in a nominally overcommitted situation.

OS/2 Version 2.0 is a flexible platform. The base product can provide multiple DOS application multitasking, as well as the multitasking and multithreading functions of Version 1.2. Version 2.0 includes facilities for 32-bit applications. And plans are to make communications and data base functions available to all OS/2 users.

In the good old days, life was simple. Since then, technology has grown more powerful to meet expanding business needs, and with that power has come complexity. But OS/2 is bringing back the good

old days. It has the power, and the rich function needed in today's environment, yet it has the flexibility to run virtually anything. Life is simple again.



## *OS/2 is bringing back the good old days.*

*Editor's note: At COMDEX/Fall 1989, IBM and Microsoft announced their intention to deliver a 32-bit version of OS/2. That version is referred in this article as OS/2 Version 2.0. As we go to press, no formal announcement (official product name, content, availability, and so on) has been made by the two companies. Reference to OS/2 Version 2.0 does not indicate a commitment by IBM to introduce a product under that name.*

### *ABOUT THE AUTHORS*

*Pino Ferrari is a senior marketing program administrator in the National Distribution Division Customer Executive Briefing Center in Boca Raton. He joined IBM in Italy in 1969. Previous positions have included assignments in development laboratories and branch offices, focusing on communications and software aspects of the data processing business.*

*Fernando Lopez is a senior marketing program administrator in the National Distribution Division Executive Briefing Center in Boca Raton. He currently presents briefings about OS/2. Fernando joined IBM in 1968. Past assignments include programming for MVS and VM operating systems, World Trade technical and marketing support for communications products, and software development manager.*







Filename	Extension	Directory	Size	Total DASD Saved (bytes)
<b>If the Command Reference is not needed, these files can be deleted:</b>				
CMDREF	INF	\OS2\BOOK	366050	
VIEW	EXE	\OS2	26672	
VIEWDOC	EXE	\OS2	25392	
VIEWH	HLP	\OS2\HELP	5615	(433,729)
<b>If DOS support is not needed, these files can be deleted:</b>				
ANSI	SYS	\OS2	1976	
APPEND	EXE	\OS2	6946	
ASSIGN	COM	\OS2	1517	
BASIC	COM	\OS2	585	
BASICA	COM	\OS2	36253	
COMMAND	COM	\OS2	29009	
DOS	SYS	\OS2	2100	
EDLIN	COM	\OS2	7996	
EGA	SYS	\OS2	2599	
GRAFTABL	COM	\OS2	8401	
JOIN	EXE	\OS2	24848	
MORTGAGE	BAS	\OS2	6380	
SETCOM40	EXE	\OS2	9387	
SUBST	EXE	\OS2	24848	(162,845)
<b>If Dual Boot is not needed, this file can be deleted:</b>				
BOOT	COM	\OS2	42384	
<b>If a font is not needed, it can be deleted:</b>				
COURIER	FON	\OS2\DLL	307760	
HELV	FON	\OS2\DLL	545456	
SYSMONO	FON	\OS2\DLL	39424	
TIMES	FON	\OS2\DLL	603440	(1,496,080)
<b>If High Performance File System (HPFS) is not used, these files can be deleted:</b>				
CACHE	EXE	\OS2	13984	
HPFS	IFS	\OS2	100378	
STARTLW	DLL	\OS2\DLL	1076	
UHPFS	DLL	\OS2\DLL	157024	(272,462)
<b>If the TOOLKIT or IBM programming languages will not be installed, these files can be deleted:</b>				
CPISPFPC	DLL	\OS2\DLL	129588	
DMPC	EXE	\OS2\INSTALL	2798	
DTM	DLL	\OS2\DLL	1786	
INACALL	DLL	\OS2\DLL	1560	
INSTAID	CMD	\OS2\INSTALL	254	
INSTAID	CNF	\OS2\INSTALL	225	
INSTAID	LIB	\OS2\INSTALL	27648	
INSTAID	PRO	\OS2\INSTALL	175	
INSTAIDE	EXE	\OS2\INSTALL	93827	
ISPD	MSG	\OS2\INSTALL	4551	
ISPM	MSG	\OS2\INSTALL	1489	
STXTDMPC	DLL	\OS2\DLL	14171	(278,072)

Figure 2a. OS/2 SE 1.2 Files That Can Be Deleted



Filename	Extension	Directory	Size	Total DASD Saved (bytes)
<b>If Introducing OS/2 is not needed, these files can be deleted:</b>				
INTROE	EXE	\OS2\INTRO	16905	
INTROI	TUT	\OS2\INTRO	56788	
INTROS	TUT	\OS2\INTRO	39567	(113,260)
<b>If mouse support is not needed, these files can be deleted:</b>				
IBMMOU01	SYS	\OS2	2612	
IBMMOU02	SYS	\OS2	3124	
MOUSE	SYS	\OS2	18484	
MSBUS01	SYS	\OS2	2612	
MSINP01	SYS	\OS2	2612	
MSPS201	SYS	\OS2	2612	
MSPS202	SYS	\OS2	3636	
MSSER01	SYS	\OS2	3124	
MSSER02	SYS	\OS2	3636	
PCMOU01	SYS	\OS2	3124	
PCMOU02	SYS	\OS2	3124	
POINTDD	SYS	\OS2	6562	
VISION01	SYS	\OS2	3124	
VISION02	SYS	\OS2	3124	
<b>If national language support is not needed, these files can be deleted:</b>				
4201	DCP	\OS2	10683	
5202	DCP	\OS2	404	
KEYB	COM	\OS2	17769	
KEYBOARD	DCP	\OS2	94007	(122,863)
<b>If any of these utilities are not needed, they can be deleted:</b>				
ATTRIB	EXE	\OS2	37619	
BACKUP	EXE	\OS2	29095	
FDISKPM	EXE	\OS2	82640	
FDISKPMH	HLP	\OS2\HELP	10508	
LABEL	COM	\OS2	33383	
LINK	EXE	\OS2	142578	
RECOVER	COM	\OS2	53184	
RESTORE	EXE	\OS2	39285	
SORT	EXE	\OS2	34385	
TREE	COM	\OS2	36461	
PICH	HLP	\OS2\HELP	11973	
PICICHG	EXE	\OS2	35280	
PICPH	HLP	\OS2\HELP	33413	
PICPRINT	EXE	\OS2	126512	
PICSHOW	EXE	\OS2	48608	

Figure 2b. OS/2 SE 1.2 Files That Can Be Deleted



Filename	Extension	Directory	Size	Total DASD Saved (bytes)
<b>If OS/2 problem determination support is not needed, these files can be deleted:</b>				
CREATEDD	EXE	\OS2	50976	
LOGDAEM	EXE	\OS2\SYSTEM	27867	
OS2SM	PDF	\OS2	2676	
PATCH	EXE	\OS2	43509	
PMC00000	TFF	\OS2\DLL	4228	
PMC10000	TFF	\OS2\DLL	7125	
PMC20000	TFF	\OS2\DLL	26425	
PMC30000	TFF	\OS2\DLL	1450	
PMC40000	TFF	\OS2\DLL	365	
PMC50000	TFF	\OS2\DLL	21507	
PMC60000	TFF	\OS2\DLL	2872	
PMC70000	TFF	\OS2\DLL	4698	
PMC80000	TFF	\OS2\DLL	469	
PMGPI	PDF	\OS2	7013	
PMGRE	PDF	\OS2	363	
PMPIC	PDF	\OS2	231	
PMSHAPI	PDF	\OS2	1531	
PMSPL	PDF	\OS2	1322	
PMTRACE	DLL	\OS2\DLL	3444	
PMWIN	PDF	\OS2	7740	
PSTAT	EXE	\OS2	18047	
SPL1B	PDF	\OS2	2112	
SPL2B	PDF	\OS2	288	
SYSLOG	EXE	\OS2	91776	
TRACE	EXE	\OS2	27103	
TRACEFMT	EXE	\OS2	73744	
TRSEL	EXE	\OS2	19281	
<b>If the spooler will not be used, these files can be deleted:</b>				
IBM4201	DRV	\OS2\DLL	83056	
OS2SPLFS	DLL	\OS2\DLL	3582	
PMPRINT	QPR	\OS2\DLL	12784	
PMSPL	DLL	\OS2\DLL	13086	
PMSPPOOL	EXE	\OS2	101648	
PMSPPOOLH	HLP	\OS2\HELP	31060	
SPL1B	DLL	\OS2\DLL	69520	
SPL2B	DLL	\OS2\DLL	3759	
SPLDVWRK	EXE	\OS2	9408	
SPLPRM	EXE	\OS2	16848	
SPOOL	EXE	\OS2	21309	
SPOOLCP	DLL	\OS2\DLL	10178	(376,238)
<b>These files can be deleted:</b>				
SYSINST	SCD	\OS2\INSTALL	5968	
CONVERT	EXE	\OS2	19173	

Figure 2c. OS/2 SE 1.2 Files That Can Be Deleted



Filename	Extension	Directory	Size	Total DASD Saved (bytes)
<b>These files are used for display support. If they are not referenced in CONFIG.SYS, they can be deleted:</b>				
BVH8514A	DLL	\OS2\DLL	8810	
BVHCGA	DLL	\OS2\DLL	23146	
BVHEGA	DLL	\OS2\DLL	32362	
BVHMPA	DLL	\OS2\DLL	15978	
BVHVGA	DLL	\OS2\DLL	36970	
<b>The correct file is copied to DISPLAY.DLL. These files can be deleted:</b>				
IBMBGA	DLL	\OS2\DLL	335360	
IBMCGA	DLL	\OS2\DLL	130048	
IBMEGA	DLL	\OS2\DLL	145920	
IBMVGA	DLL	\OS2\DLL	148480	
<b>These are the serial support files. If a file is not referenced in CONFIG.SYS, it can be deleted:</b>				
COM01	SYS	\OS2	9780	
COM02	SYS	\OS2	16948	

Figure 2d. OS/2 SE 1.2 Files That Can Be Deleted

## Part 2 – OS/2 EE 1.2

**Initial OS/2 EE Installation:** A minimum configuration (answering “NO” to install prompts) was installed with the following:

- “NO” serial device support
- “NO” retrieve command support
- “NO” command reference
- “NO” device support diskette to install
- Communications Manager with test configuration file
- Database Manager, RDS and Query Manager included (Requester only, no local databases to be used)
- LAN Requester

The amount of DASD left on the test system was 13,582,336 versus 12,357,632 for a maximum configuration setup (answering “YES” to install prompts). This constitutes a savings of 1,224,704 MB after initial installation.

### Communications Manager

**Features:** Once Communications Manager has been installed and customized, the keyboard remap and configuration services files can be deleted. To do this, execute a REINST of Communications Manager and specify that you want to remove those features.

*Note:* Once configuration services is removed, modifications to the Communications Manager configuration file are not allowed. In addition, verification of other configuration files will not be possible. However, configuration files that have already been verified can be copied from another system and Communications Manager will execute normally.

This amounts to a savings of over 3.5 MB.

**Reinstallation Features:** A number of files that control the installation and removal of EE subsystems (for example, Communications Manager, Database Manager, and LAN Requester) can be deleted. It is important to understand that once

these files are removed, a REINST can never be done again. Before removing the files, make sure that the system is configured correctly or devise a backup scheme for the deleted files so they can be restored if needed again.

The files that can be deleted are shown in Figure 3. Under the test configuration, all of these files were deleted except those in the \OS2\DLL group, which were not present. The DASD saved totaled 1,006,685 bytes.

**Miscellaneous Features:** Once OS/2 is installed, a number of miscellaneous feature files can be deleted. These files may not be needed depending on the user’s installed base.

\OS2\INTRO	797,680
(OS/2 introduction and tutorial)	
\OS2\BOOK	365,717
(used by command reference)	
\OS2\DLL	360,876
(used by Database Services)	



Under the test configuration, all of these files were deleted except those in the \OS2\DLL group, which were not present. The DASD savings totaled 1,006,685 bytes.

Path: \OS2\INSTALL	Path: \OS2\DLL	Path: \OS2\INSTALL	\OS2\INSTALL (cont.)
CMDIR.DA*	ACS3ECT.DL*	CMCUBLD.PI*	EIPINITL.PI*
CUSTBLD.PI*	ACSGCBLD.DL*	CMCUINST.PI*	EIPIPLUP.PI*
CUSTINST.CM*	ACSMGFM.DL*	CMINST.PI*	EIPSAVHV.PI*
DBDIR.DA*	ACSSVC.DL*	CMREMOVE.CM*	EIPYES.TX*
EEIN.MS*	ISPMALLC.DL*	CMREMOVE.PI*	IBMLAN.EX*
EEINST.CN*	PICTDMP.C.DL*	CMRMPREV.CM*	LRCHKSPC.PI*
EEINST.LI*	RCDAPIEX.DL*	CMTITLE.TX*	LRCOPFIL.PI*
EEINST.PI*	RCFGFILE.DL*	DBCUBLD.PI*	LRCUBLD.PI*
EEINST.PR*	RCOMAIN.DL*	DBCUIINST.PI*	LRCUIINST.PI*
EEOLOG.DA*	RCVMAIN.DL*	DBDSGC1.DA*	LRDOINST.PI*
EICCNVSV.EX*	RNLCHARS.DL*	DBDSGC2.DA*	LRFIXINI.PI*
EICDELSV.EX*	RPSMAIN.DL*	DBDSGC3.DA*	LRINST.PI*
EICERCON.PI*	RSMMAIN.DL*	DBDSGC4.DA*	LRMUGINS.PI*
EICERROR.PI*		DBDSGWSN.PI*	LRREMOVE.PI*
EICLGMSG.PI*		DBDSINST.PI*	LRRMPREV.CM*
EICSRVCS.EX*	<b>Path: \OS2\INSTALL</b>	DBDSIPAN.PI*	LRSETVAR.PI*
EIDPNTXT.DA*	ACSBIND.MS*	DBINST.PI*	LRSPECNM.PI*
LPOINST.PI*	ACSE.MS*	DBLOADFT.PI*	LRUPDAIF.PI*
LRDIR.DA*	APPCPC.DA*	DBQMG.C.DA*	LRWKSSVC.PI*
MUGDIR.DA*	CFG.DA*	DBQMINST.PI*	MUGINST.EX*
ROCSDD.EX*	CIKS.CN*	DBREMOVE.PI*	MUGINST.PI*
REINST.CM*	CIKS.LI*	DBRMPREV.CM*	REXXINST.PI*
RFRSHICO.EX*	CIKS.MS*	DBRMVDS.PI*	SQLEMGDB.EX*
VIEWBLOG.CM*	CIKS.PR*	DBRMVQM.PI*	SQLENODE.EX*
VIEWBLOG.PI*	CIKSTART.EX*	DBSETVAR.PI*	
VIEWLOG.CM*	EECFG.CM*	DBSPACE.PI*	Total: 249,185 bytes
VIEWLOG.PI*	ISPD.MS*	DELTITLE.EX*	
	ISPM.MS*	EIPCFGSY.PI*	
Total: 335,524 bytes	PICT.MS*	EIPCHECK.PI*	
		EIPCMPIN.PI*	
	Total: 421,976 bytes	EIPERROR.PI*	
		EIPICECL.PI*	

Figure 3. OS/2 EE 1.2 Files That Can Be Deleted

### Total DASD Savings

Initial install. savings	1,224,704
Introducing OS/2	797,680
Files in Figure 3	1,006,685
Communications Mgr.	
REINST	<u>3,500,000</u>
Total DASD saved:	6,529,069

After deleting all unnecessary files, OS/2 EE requires approximately 25 MB of DASD.

### ABOUT THE AUTHOR

W. Craig Chambers is a senior market support representative. He joined IBM as a systems engineer in 1969 after receiving B.S.M.E. and M.S.M.E degrees from Purdue University. His previous assignments include lead systems engineer at several large MVS/JES2/JES3 accounts, technical support for communications products such as the 3270 system,

the 3270-PC, the Workstation Program, and Enhanced Connectivity Facilities and technical support for OS/2 SE. He has been a presenter for IBM's Technical Coordinator Program television broadcasts to its customers and has published several IBM technical bulletins.



# User Profile Management

Laura Camp, Carolyn Easter,  
and Roy Feigel  
IBM Corporation  
Austin, Texas

**User Profile Management (UPM) is a subcomponent of OS/2 Extended Edition 1.2 (EE). It provides functions for identifying users in support of Database Manager™, Communications Manager™, and the LAN Requester components. Each workstation that has OS/2 Extended Edition 1.2 installed has the UPM function.**

UPM could be described by its functions:

- Creation and deletion of users
- Management of user profiles by UPM administrators
- Creation, deletion, and maintenance of groups
- Management of user logon and logoff functions, and maintaining the state of users (that is, validated, password expired or revoked, and so on)
- Option for users to modify their own passwords and comments

UPM could also be described by its structures:

- Presentation Manager interface
- Application programming interface
- Data structures
- Support for local area network

UPM is an enhancement to and an extension of the Access Control System (ACS). Although database objects and LAN resources are protected individually by these components and not by UPM, these

components use the ACS application programming interfaces (APIs) for access control or to query the validation of a user ID. ACS is a set of APIs for user validation and access control, and a data file, NET.ACC, which contains user ID, passwords, group names, and access control lists.

The Database Manager maintains its own set of authorization tables defining which users and groups may access particular databases, tables, views, access plans, and so on. LAN Server, on the other hand, uses the Access Control System itself, using the application programming interfaces provided by the LAN Server component. Figure 1 shows UPM as part of the Extended Edition 1.2.

## Concepts

A "user" is a unique user ID and password defined to UPM on an OS/2 EE system – whether in a stand-alone or a physical LAN-connected environment (that is,

LAN Server or Requester may or may not be installed). UPM stores these user definitions (user IDs and encrypted passwords) in its own UPM database on disk.

When defining users on a system without OS/2 1.2 LAN Server or Requester installed, the user ID definitions will be stored only on that particular workstation. When defining users on a system on which OS/2 LAN 1.2 Server or Requester is installed and the creator is logged onto the domain, the user ID definitions will be added to the domain controller. During LAN operation, when the domain controller and other LAN servers are operational, the user IDs and group IDs will automatically be propagated to all OS/2 LAN servers in the domain.

## LAN Configuration

Designing a network containing the OS/2 Extended Edition components requires a look at the possible configurations. A domain controller server is the central server for an

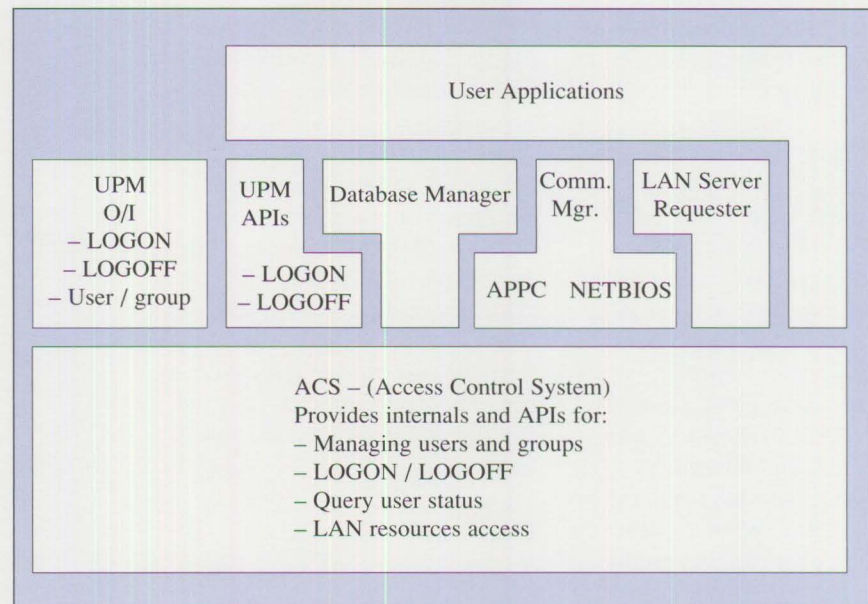


Figure 1. UPM and ACS System Structure



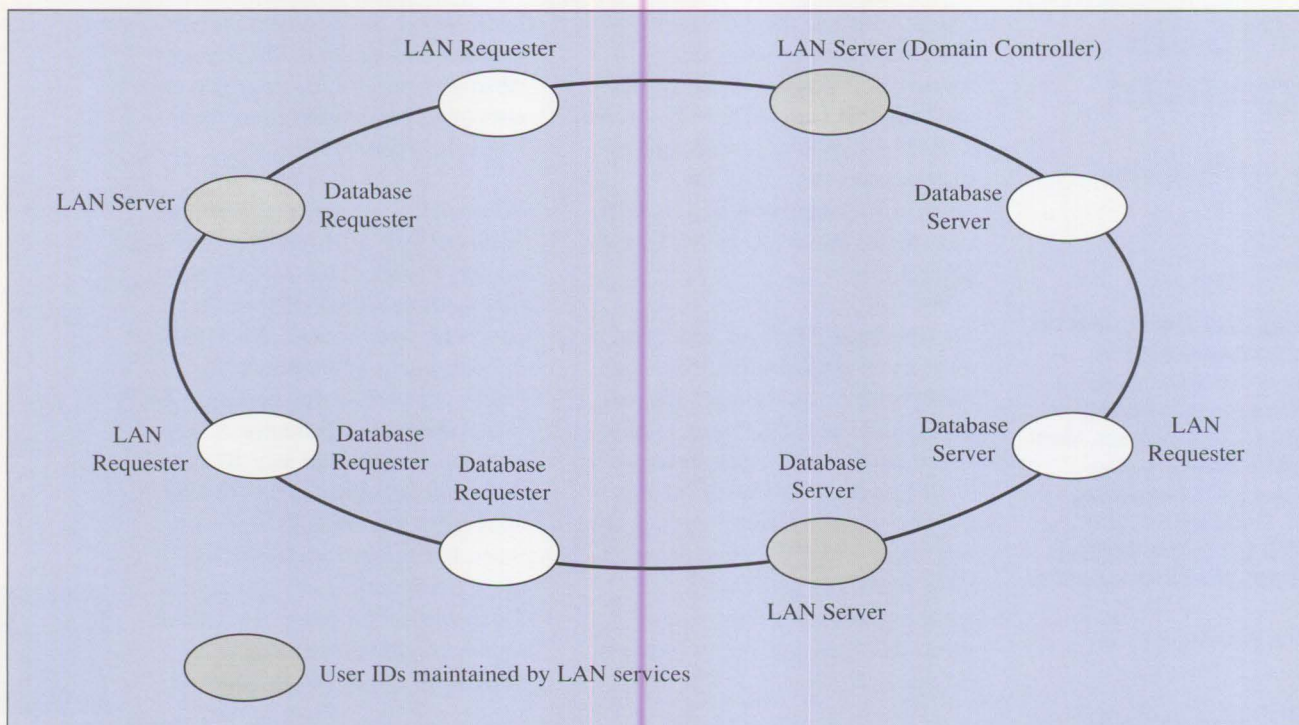


Figure 2. LAN / Database Combinations

OS/2 LAN Server network. This server processes all of the logon requests from DOS LAN Requesters and OS/2 Extended Edition requesters to the named domain. This server also contains all the definitions and profiles for each user, application, and aliases.

The access control for each resource on a server resides on that server machine in a file named NET.ACC. This is the same file that contains all the user IDs, group IDs and passwords for the user IDs. All user IDs are created by an administrator and stored on the domain controller server. Other servers participating in the LAN Server 1.2 domain receive the new user IDs and group IDs. This enables a server that is not the domain controller to allow or deny access to a resource residing on that server. A database server that is not a LAN server cannot participate in the automatic replication of user ID

and group ID information. However, a database server that is also a LAN server can take advantage of the automatic replication of user information.

Figure 2 shows a network with database and LAN servers.

Other advantages of running LAN Server 1.2 and database server on a single server machine are manifested in the OS/2 command line area. With LAN Server 1.2 installed, a network administrator can control the password expiration period, the password length, the number of unique passwords required, and the minimum required password period.

### Users and Logons

A user can log on in one of three ways:

- To a domain (the default for the first logon in a system with OS/2 1.2 LAN Server or Requester installed)
- To a remote node
- Locally (the default for the first logon in a system where OS/2 1.2 LAN Server or Requester is not installed)

The default for second and subsequent logons is local.

If the logon is to a domain and LAN is installed, the LAN requester will be started (if not yet started) and LAN will send the user ID to the domain controller for validation. Again, if the user ID is valid, this fact will be stored in the Logon Entry Table of the requesting workstation. For LAN Server logon, the password is encrypted immediately. If the logon is successful, the LAN component of the requesting work-



station calls the ACS and places the user in the "validated" state by setting an indicator in a UPM-maintained RAM table called the Logon Entry Table. The user remains in the logged-on state until logged off. The Logon Entry Table contains the following information:

- User ID
- Password
- Remotename (if applicable)
- Nametype (local, remote LU6.2 node, or domain)
- Session ID
- Process ID
- Validated indicator

If the logon is local, UPM will validity check the user ID and password against the local UPM database.

If the logon is to a remote node (for example, for remote database access), UPM at the local workstation creates an entry for this logon in the Logon Entry Table. When a connection is made to the remote node, an attempt will be made to log this user on at the remote node. The status of this logon attempt will be stored in the Logon Entry Table of the requesting workstation as well.

There are two types of character sets supported by UPM:

- Minimal character set – this set of characters provides future IBM SAA™ compatibility, and consists of the common subset of all characters supported by the Extended Edition components using UPM. These are the rules that will be applied for validity checking characters when adding or updating a user ID, group ID, or password.
- Expanded character set – Although this set is currently sup-

ported by UPM, it does not conform to IBM SAA compatibility. These are the rules that will be applied for validity checking characters at logon time for user ID and password.

### Group

User IDs may be combined to form an entity called a group. Components using UPM may allow or disallow use of component-specific objects for groups as well as for users. Groups are named with group IDs that have the same naming rules as user IDs. However, a group ID must not be the same as any user ID on that system.

### UPM Authority Levels

Each user is defined to UPM with one of the following three types:

- Administrator
- Local administrator
- User

This definition is made at the time the user is added to UPM.

An administrator has the authority to use all UPM functions. Within any OS/2 EE component using UPM, an administrator is allowed to access all protected system objects and perform all possible system tasks. For example, within the Database Manager component, a UPM administrator has SYSADM authority. (See the *IBM Operating System/2 Extended Edition Version 1.2 Database Manager Programming Guide and Reference: Volume 1*, S01F-0269, for information about SYSADM and other database authorizations.)

A user with local administrator authority has SYSADM authority within the Database Manager on the local system only. Within UPM

(and to any Extended Edition components other than Database Manager), a local administrator has the same level of authority as a UPM user. There can only be one local administrator per workstation. If a second local administrator is defined, the first one will become a normal UPM user. The purpose of the local administrator authority level is to give users the capability to define and use their local databases (since this requires SYSADM database authority) without giving them UPM administrator authority (that is, the ability to create and change user and group definitions) on their own workstations or across networks.

Users with UPM user authority are allowed to log on, log off, and change their own passwords and comments. No other UPM functions can be accessed. Authority to access objects owned by a specific OS/2 EE component (such as authority to access an OS/2 database or objects in that database) is determined according to rules within that OS/2 EE component.

### Human Versus Program Logons

There are two ways to log on via UPM:

- Through human interaction (via the OS/2 command line or Desktop Manager windows)
- From a program (via an API call)

In both situations, multiple user IDs can be logged on to an OS/2 EE system simultaneously, as described in the section *Running under Multiple User IDs*.

For logon by human interaction, User Profile Management provides a Presentation Manager™ window interface to its functions. Selecting



“User Profile Management Services” from the OS/2 Extended Edition Desktop Manager window allows access to the following three functions:

- Logon
- User Profile Management
- Logoff

These three functions can also be accessed directly from the OS/2 command line. (See the command descriptions in the section *UPM Command Interface*.) Executing any of these functions initiates one or more popup windows that enable a user to perform the desired UPM functions.

In addition to allowing a human user to log on and execute programs, an API is provided to allow a program to log on. The program logging on must provide a user ID under which it is to be run and, optionally, a password. To prevent an unauthorized user from executing such a program, additional checking may be specified in the logon call with the authorization check (authcheck) option. The following authcheck levels can be specified:

- The calling process must have been started by the CONFIG.SYS configuration file, or from a process so started
- The logged-on local user for the calling session must have UPM administrator authority
- No special authority is required to execute the program

A process that logs on via the logon API is responsible for logging off via the logoff API when it completes execution. If the logoff is omitted, indication of the logon will remain in the Logon Entry Table until the system is powered-off or the next IPL is executed.

## Running under Multiple User IDs

The main reason for allowing multiple users to be logged on simultaneously is to enable the running of multiple programs where each has a different authority for accessing protected objects. For example, you may want a background application to manage system-level database resources, while simultaneously allowing a user or user-initiated program restricted access to the same database.

*Each logged-on user ID has a unique process and session ID identifying it.*

### Human Logons:

When operating with local human logons, a system can run in one of two modes:

- single local human logon at a time
- multiple simultaneous local human logons

The mode is set by the “/O” option on the OS/2 command line version of the logon command (see *UPM Command Interface*). When running in single user mode (the default), only one local human logon is allowed at a time in the system. Any programs initiated by the user will run under that user ID. If the user re-issues a local logon, the currently active local user ID will be overlaid with the newly specified one. Any protected resources accessed after that time will use the new user ID. (Note: for Database Manager, the user ID used for access control is es-

tablished when the “Start Using Database” call is issued.)

To run in multi-user mode, the “/O=MULTI” option is specified on the logon command. When this option is active, the user will still be allowed to initiate local logons (via the command line or the Desktop Manager windows). However, when a protected resource is accessed (such as a database), the local logon window will be displayed, prompting the user to log on.

Why force the user to be prompted for a logon in the /O=MULTI environment? Each logged-on user ID has a unique process and session ID identifying it. For this logon to be associated with a running application, the application must have the same process and session ID. However, human-initiated logons cause a new process to be started and then closed. When an application is later started, even if a logon was performed by a user from the same session as the application, this logon will not have the same process ID as the application. Hence, UPM will not be able to ensure that the correct user ID is associated with the application. Forcing the user to be prompted for a logon from the running application will ensure that the correct user ID is associated with the application.

The /O option does not apply to remote node or domain logons. For remote node logons, multiple logons are allowed in a single OS/2 session, but each remote logon in that session must be unique. If the user logs on from a session with a remote node name that is already logged on for that session, the new logon simply replaces the existing entry for it in the Logon Entry Table. For domain logons, a workstation can only be logged onto one



domain at a time. Similarly, there can be only one logon at a time for a single user ID within a domain.

**Program Logons:** When operating with program logons, every OS/2 process and session can issue a logon via the Logon API. The user ID specified in the call is associated with the process and session in which the program is running. It is used to verify the program's authorization when it attempts to access protected objects.

There is no limit to the number of program logons per workstation because it is always possible to determine which logged-on user ID is to be associated with a particular program. Each session that logs on can specify a user ID that has not yet been logged on, or one that is already logged on. Each process within a session can also log on with a different user ID. The only restriction is that the authcheck option must be the same as the parent process. If an application in a specific process and session logs on with a new user ID before logging off the previous one, the previous user ID will be overlaid with the new one.

## Distributed and LAN Systems

UPM maintains a RAM table (called the Logon Entry Table) of the user IDs that have been logged on from that system. These entries may consist of local as well as remote and domain logons.

If a user performs an action that takes down a communications connection with the remote node in use, the next time a connection is made to that remote node, the user will not be reprompted for the user ID. Instead, the OS/2 EE component

making the connection will reuse the previously specified user ID and password for that remote node and revalidate it by calling UPM at the remote node.

Additionally, if the user defines the same user ID and password both locally and in nodes and domains that are accessed remotely, only a single logon is required. If the first logon is local, and later a remote logon is required (to access a remote database, for example), the OS/2 EE component needing the remote connection will automatically attempt a remote logon (using the locally logged on user ID). Likewise, if the user has first logged on to a remote node and later a local logon is required, an attempt will be made to log the user on locally using the logged-on user ID for that remote node.

*UPM maintains a RAM table of the user IDs that have been logged on from that system.*

When Query Manager is started, it picks up the currently logged-on user ID (whether that is a domain user, node user or local user), and attempts a local logon if the user is not already logged on locally with userid and password. If the userid/password exists locally, the local logon occurs and Query Manager finishes its initialization without the local logon prompt being displayed.

However, if the userid/password does not exist locally, the local

logon window is displayed with the failed userid displayed in the userid field. The person at the keyboard can disregard this prompt by pressing ESC and continuing Query Manager initialization. Bypassing the local logon operation causes the Query Manager interface to database administration functions to be unavailable for the person at the keyboard. To make the Query Manager interface to these administration functions available (and usable against a remote database), a UPM Administrator userid/password must be defined locally which matches the userid/password defined at the remote database location. The same ID must be defined at the remote database node with an equal level of authority. If a local database is accessed, only a local ID must be defined. If database administration functions are to be used, the ID must be defined as an administrator or local administrator.

Even though Query Manager is packaged with OS/2 EE Database Manager, it is merely an application from Database Services' (and OS/2's) point of view. That is, it uses only published APIs and functions available to any application developer. Query Manager cannot use lower-level UPM APIs that the underlying Database Services uses in connecting to a database. The user *can* escape out of Query Manager's "logon," though this makes certain administrator functions inaccessible from Database Manager's window. When you then connect to a database, the previously described "rules" apply, because connecting to a database is a Database Services function, not a pure Query Manager function.

In more general terms, when an OS/2 EE component (such as Database Services) attempts to ac-



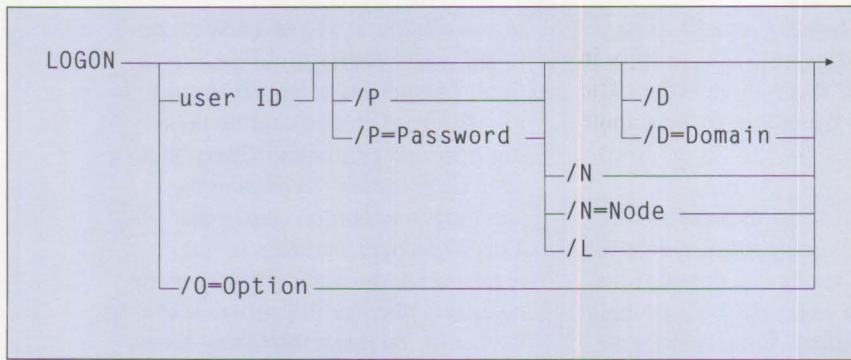


Figure 3. LOGON Command.

cess a protected resource on behalf of an application or user, the component determines the location of the resource (such as a database) and then asks UPM for a logged-on user ID for that location in the current session. If UPM has an ID logged on for that location in the current session, it passes this ID back to the requesting component. Otherwise UPM guesses (using a predetermined algorithm) which user ID entry in the Logon Entry Table to pass back to the requesting EE component. This component then performs the necessary functions to connect to the desired location (if necessary) and perform a logon.

Specifically, in the case of remote database access, Database Manager passes the user ID and password on to Communications Manager, which then calls UPM at the remote location for user ID validation.

Likewise, on a LAN Requester workstation, if the user first does an explicit local logon and later LAN Requester is started, UPM gives LAN Requester this local user ID to be validated at the domain controller.

### UPM Command Interface

UPM provides the following commands that may be executed from the OS/2 command line:

- LOGON
- LOGOFF
- UPMACCTS
- UPMCSET

Other commands that allow a user to work with user and group IDs and view logon status are provided with OS/2 LAN Server 1.2. These commands are documented in the *IBM Operating System/2 Local Area Network Server Version 1.2 Commands Reference* (S01F-0278).

**LOGON:** The LOGON command can be issued at the OS/2 command line. To automatically log a user onto the system at IPL, this command can also be placed in the CONFIG.SYS file (using the RUN= command) or in the STARTUP.CMD file. The format of the LOGON command is shown in Figure 3. The components of this command are:

- User ID – If no user ID is specified, the Logon Popup window is displayed.
- /O – This option may be specified as either SINGLE, for single local logon, or MULTI, for multi-

ple local logons. If the option is set to SINGLE, only one local human logon entry will be allowed in the Logon Entry Table. After the first local logon, all applications run under that user ID until a logoff or another local logon occurs.

If the option is set to MULTI, only implicit local logons are allowed. UPM forces a new logon panel for each new application that accesses a protected resource. Access to multiple protected resources by a single application process does not cause a new logon for that process.

- /P – This option allows the user to specify a password. If no password is specified, but one is required, the Logon Popup is displayed, prompting the user for the password.

One of the next three options can be specified to indicate where the logon will occur. If none is specified, the default for the first logon will be:

- To a domain if the OS/2 1.2 LAN Server or Requester is installed
- Local if LAN Server or Requester is not installed

The default for subsequent logons is local. If specified, these options work as follows:

- /D – This option lets the user specify the name of a domain on which to log on. A LAN workstation can only be logged on to one domain at a time. The first logon to the domain (whether by default or explicit) will start LAN Requester if not yet started. If the LAN is installed:



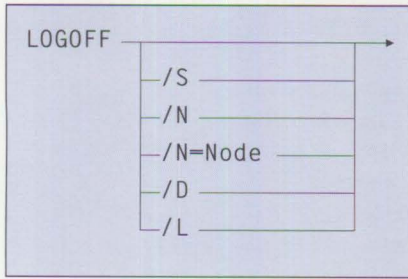


Figure 4. LOGOFF Command

- If /D is specified without a domain name, UPM will first attempt to log on to the default domain for that workstation. If this logon fails, UPM will display the Logon Popup for Remote Domain Access, prompting the user for a domain name.
- If /D=Domain is specified, an attempt will be made to logon to the specified domain.

If the LAN is not installed or could not be started and /D is specified, it will be ignored and a local logon will be performed.

- /N - This option lets the user specify the name of the node on which to log on the next time the system requires that node be accessed. If a node name is not specified after the /N, the Logon Popup for Remote Node Access will appear, prompting the user for the node name.

The user ID and password for the remote node is not validated lo-

cally. Rather, "logon /N=nodename" places an entry in the local Logon Entry Table. The user ID is validated at the remote node at the time the remote connection is made. An indication of user ID and password validity is returned to UPM at the requesting workstation and is recorded in the local Logon Entry Table.

- /L - This option forces a local logon. If the LAN is not installed, this is the default option.

**LOGOFF:** The format of the LOGOFF command is shown in Figure 4.

- /S - This option causes the current session to be logged off.
- /N - Without any parameters, this option specifies that entries for all nodes in this session be removed from the Logon Entry Table. If a node name is specified, all entries with this node name (from any session) are logged off (that is, removed from the Logon Entry Table).
- /D - This option causes the current domain logon to be logged off.
- /L - This option causes the Logoff Popup to display a list of all logons for that workstation. This allows the user to select the specific user IDs and session(s) to log off.

Change logged-on user information	Group management (administrator only)	User management (administrator only)
Change comment	Add new group	Add new user ID
Change password	Update group	Update user information
Select groups for user ID (administrator only)	Erase group(s)	View user profile
		Select groups for user ID
		Erase user ID

Figure 5. Provided Functions

**UPMACCTS:** The UPMACCTS command displays the UPM Primary Window, giving the user access to the various UPM functions. If the current user has UPM administrator authority, the User Management and Group Management functions may be accessed. Figure 5 lists the functions provided. (Note that if LAN is installed, the "Use Domain" function is also provided. See the *IBM Operating System/2 Local Area Network Server Version 1.2 Commands Reference*, S01F-0278, for details.)

When UPMACCTS is initiated, a local or domain logon is required. If no one is logged on to the workstation, the user is prompted for a local logon. If there are user IDs already logged on (local or other), UPM first looks for a local ID to use. If there is none, UPM chooses a non-local ID and validates it locally.

**UPMCSET:** This command determines the character set that will be allowed when defining user IDs, group IDs, and passwords. (For a definition of the character sets, see *Concepts*.) The format of the UPMCSET command is shown in Figure 6.

- /M - Sets the character set to the Minimal Character Set. This is the default if no parameters are specified on the UPMCSET command. This is also the default that comes with Extended Edition if the UPMCSET command is never executed.
- /E - Sets the character set to the Expanded Character Set.

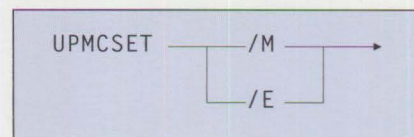


Figure 6. UPMCSET Command



## UPM Programming Interface

The two APIs provided by UPM are for logon and logoff. If the LAN is installed, additional APIs are available. (For a description of these APIs, see the *IBM Operating System/2 LAN Server Version 1.2 Application Programmer's Reference*, S01F-0256.)

**Logon:** The logon function identifies that the process and session is logging on to the system with the specified user ID and password. If LAN is not installed or could not be started and a domain name is specified, an error will be returned to the program. The parameters for the logon call are:

- user ID – The UPM user ID that is to be validated (and will be associated with the session and process ID in which the logon occurs).
- password – The unencrypted password associated with this user ID.
- remotename – The name identifying the node or domain for which the user ID and password are required. This parameter is ignored when logging on to the local workstation (that is, 'remotetype = UPM\_LOCAL').
- remotetype – The type of node named in the remotename parameter. Valid values are:
  - UPM\_LOCAL – the local node
  - UPM\_DNODE – specifies that remotename is an LU6.2 node name
  - UPM\_DOMAIN – specifies that remotename is an OS/2 LAN Server 1.2 domain name

- authcheck – An indicator of the authority required for the calling program to issue this logon. Valid values are:
  - UPM\_CONFIG – The calling process must have been started from the CONFIG.SYS configuration file, or from a process that was so started
  - UPM\_ADMIN – The logged-on local user for the calling session must have UPM administrator authority
  - UPM\_USER – The calling program does not require any additional authority and the program can have been started by a normal UPM user

**Logoff:** This function logs the calling program off the system. There must be one logoff call for every logon call, and the logoff must occur from the same process and session that issued the logon. The parameters to this call are:

- user ID – The UPM user ID to be associated with the logging off process
- remotename – The name identifying the node from which the user ID is to be logged off. This parameter is ignored if 'remotetype = UPM\_LOCAL'.
- remotetype – The type of node named in the remotename parameter. Valid values are:
  - UPM\_LOCAL – the local node
  - UPM\_DNODE – specifies that remotename is an LU6.2 node name
  - UPM\_DOMAIN – specifies that remotename is an OS/2 LAN Server 1.2 domain name

## ABOUT THE AUTHORS

*Laura Camp is an advisory programmer in IBM's Entry Systems Division in Austin, Texas, currently responsible for dissemination of OS/2 Performance Tools and Information. She joined IBM in 1976 in the Office Products Division software architecture. Later she became involved in user interface design and software development for Displaywriter and DisplayWrite® 2, 3, and 4. In 1986, Laura joined the OS/2 Database Manager development team followed by two years in OS/2 EE Product Support, where she focused on OS/2 EE Database Manager. Laura holds a B.A. degree in mathematics from Hope College.*

*Carolyn Easter is an advisory programmer in IBM's Entry Systems Division in Austin, Texas. She joined IBM in 1980 and was assigned to program the files and database system on the 5520 Administration System. She has a B.S. in mathematics from Southern Methodist University and a B.A. in computer science from the University of Texas at Austin.*

*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 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.*



# Understanding OS/2 1.2 LAN Server Performance

Ken Whitfield  
IBM Corporation  
Austin, Texas

**The performance of any local area network file server is influenced by many factors. This article provides some information about the effects of the server's file system, and the server/requester data flow logic and buffers on the performance of Operating System/2® 1.2 LAN Server. Tuning recommendations relative to server workload and certain classes of applications are suggested.**

The performance of any local area network (LAN) file server will be determined by a myriad of factors, including the following:

- CPU speed
- Hardware architecture
- Local area network speed and architecture
- DASD access time and data transfer rate
- File server operating system and file system
- File server/requester software design
- Memory available for file system caching and network buffers
- File server workload
- Other applications running on the server
- Requester applications

The first four items in the list are fixed with respect to OS/2 1.2 LAN Server running on a particular per-

sonal computer and local area network; therefore, this article will not deal with those items. The article contains information about how the remaining factors affect OS/2 1.2 LAN Server performance. There is considerable interaction between some of these factors. Understanding the nature of the individual factors can enable one to isolate the predominant factors in a particular environment and optimize the OS/2 1.2 LAN Server accordingly.

## OS/2 1.2 File Systems

The File Allocation Table (FAT) file system is the file system used in all versions of the DOS and OS/2 operating systems. The cache used by FAT is called DISKCACHE. A read request results in a look into the DISKCACHE memory (look-aside cache) for the physical sectors corresponding with the file handle and offset passed in the read request. If the data is in the DISKCACHE, it is passed to the application from cache memory, avoiding a time-consuming read from the fixed disk.

How did the data get into DISKCACHE? A read request must also pass the amount of data it wants to read (data buffer size). If this amount is seven sectors (3.5 KB) or less, eight sectors of data (two 2 KB cache pages) starting at the first sector containing the data is read into DISKCACHE. If the application is reading 512 bytes sequentially, one read of the fixed disk gets eight sectors of data placed into DISKCACHE, saving the time of seven additional fixed-disk accesses.

The check on the data size being requested is referred to as the threshold of the caching algorithm. A threshold is used to prevent the

DISKCACHE from being flushed (old data being replaced with newer data) by an operation such as program load or file transfer, in which the data buffer size is often 64 KB. The two 2 KB cache pages of data in DISKCACHE are eventually discarded using a least recently used (LRU) algorithm. OS/2 1.2 offers a threshold that can be set if the seven-sector default threshold is not optimal. It can now be increased up to 32 sectors (DISKCACHE = 2048,32). Another important point to note is that the maximum size of DISKCACHE is 7.2 MB.

Bigger is not always better, however, unless you are sure the caching algorithm being used by DISKCACHE is putting data into cache that will subsequently be used. (See *Performance Tuning*.)

The High Performance File System (HPFS) solves many problems of the FAT file system, but it is not derived in any way from the FAT file system. HPFS was designed from scratch to take full advantage of a multitasking environment. (For an extensive description of HPFS, refer to the article "Design Goals and Implementation of the New High Performance File System" in the *Microsoft Systems Journal*, Volume 4, Number 5, and reprinted in *IBM Personal Systems Technical Solutions*, Issue 2, 1990.) Some of the key performance and functional improvements over the FAT file system are listed in Figure 1 (where different characteristics of both the 32- and 16-bit versions are noted).

The cache used by the HPFS is CACHE.EXE. Its characteristics differ from DISKCACHE in the following ways. The caching algorithm uses a threshold value of 2 KB, and it is not changeable. Other factors such as the number of "dirty" cache



	FAT	HPFS
Maximum filename length (characters)	11 (8.3 format)	255
File attributes	Bit flags plus up to 64 KB, text or binary	Bit flags plus up to 64 KB, text or binary
Maximum path length (characters)	64	260
Average wasted space per file	1/2 cluster (1 KB)	1/2 sector per (256 bytes)
Allocation information for files	Centralized in FAT on home track	Located nearby each file in its FNODE
Free disk space information	Centralized in FAT on home track	Located near free space in bit maps
Directory structure	Unsorted linear list, must be searched exhaustively	Sorted B-Tree
Directory location	Root directory on home track, others scattered	Located near seek center of volume
Read-ahead	None prior to DOS 4.0, primitive read-ahead optional on DOS 4.0	Caches read in 2 KB blocks (286 version) Sensitive to data type and usage history (386 version)
Cache replacement strategy	Simple LRU	Modified LRU (286 version) Sensitive to data type and usage history (386 version)
Write-behind (lazy write)	One sector (buffer's function)	Optional, can also be defeated on a per-file handle basis

Figure 1. HPFS Improvements Over FAT File System

pages in the cache are also included in the algorithm, which determines when read data is put into the cache. (More about "dirty" cache pages later.)

Another important difference from DISKCACHE is that the maximum size for CACHE.EXE is 2 MB. Both the threshold and maximum size used in CACHE.EXE are peculiar to the 16-bit version of the HPFS used in OS/2 1.2, and are different from those used in the 32-bit version of the HPFS.

One of the significant performance improvements of the HPFS over FAT is the optional capability to use deferred, or "lazy," writes to the fixed disk. With this feature, writes to the HPFS go first into the cache memory for subsequent writing to the fixed disk at otherwise idle times of the fixed disk. This allows incoming read requests to be processed ahead of write requests, thus providing response time improvements to the user application. The

data placed into the cache memory to be "lazy" written is partitioned into 2 KB cache pages. It is marked as "dirty," meaning that it must be written to the fixed disk (the application was told that it was written to disk). These "dirty" pages of data cannot stay in cache indefinitely. There are parameters associated with CACHE.EXE that define how long the "dirty" pages can remain in cache. (See BUFFERIDLE, MAXAGE, and MAXIDLE in the *IBM OS/2 Extended Edition Version 1.2 Commands Reference*, 01F0282.)

### Flow Control Logic

The OS/2 1.2 LAN Server and Requester software is designed to optimize the movement of file I/O data from the server to the requester. This would be a straightforward task if there were not so many ways in which a user application might access data from a file system. From the many ways that data may be accessed, it is helpful to consider the following:

1) A read request for a small amount of data (for example, 512 bytes), which may reside anywhere in the file being accessed.

2) A series of read requests, each for 512 bytes (one sector), the sectors located one after the other on the fixed disk.

Traditionally, Case 1 is called random file access, and Case 2 is called sequential file access.

#### Case 1) Random File Access

This type of file access is characterized by a request for a small amount of data coming from anywhere in the file. In many cases, this necessitates a disk seek. Seeks take a relatively long amount of time compared with reading the same data from memory, illustrating the importance of caching small record random I/O data. If the file being accessed is small compared to the size of the cache and the file is accessed frequently, then it is likely



that much of the file will reside in cache.

As the size of the file becomes large with respect to cache size, the likelihood of finding the desired data in the cache diminishes rapidly. The file may not be in the cache because unused cache pages are removed from the cache by the LRU algorithm as a result of file I/O requests to other files at the server. An extreme case is where the file is much larger than the cache size. This could lead to rarely finding the data requested in cache. In fact, a large cache size could be detrimental to performance because the time required to search the cache for the data, but never finding it, is added to the time required to physically access the fixed disk.

#### *Case 2) Sequential File Access*

This type of file access is characterized by successive calls to the file system requesting data that is physically contiguous on the fixed disk. It usually requires no physical movement of the disk head assembly. Of course, if the file being read or written is larger than the amount of data contained on a disk cylinder, the disk head assembly would have to be moved to adjacent cylinders. The time to accomplish this is much smaller than multicylinder moves seen in random file accesses. Sequential file access throughput (number of bytes per second) that can be transferred from a fixed disk can be many times higher than for random file access. This is because the physical time to move the disk head assembly to the data is minimized.

Let's now look at how the OS/2 1.2 LAN Server/Requester handles these two extreme cases of file access.

An application running on the requester issues a DOS read request for 512 bytes of data from the server. The request is passed to the redirector code, which constructs a Server Message Block (SMB) for DOS Read and calls NETBIOS to transmit the SMB to the server. NETBIOS transports the SMB over IBM Token-Ring or Ethernet™ to the server, where it is processed. The SMB is interpreted as a DOS Read of 512 bytes, and a call is issued to the server's file system. Because the amount of data requested is less than 2 KB, the file system reads 2 KB from the fixed disk, writes it to cache (assumes the HPFS Cache threshold), and returns to the DOS Read call with the 512 bytes of data requested. The data goes back to the requester in an SMB, which correlates with the original DOS Read SMB. The redirector strips the data out and returns it to the application's DOS Read call. This is the sequence of operations that happens with each random file read. (More about random I/O later when buffers are described in the section *Buffers, Caches, and SMB Protocols*.)

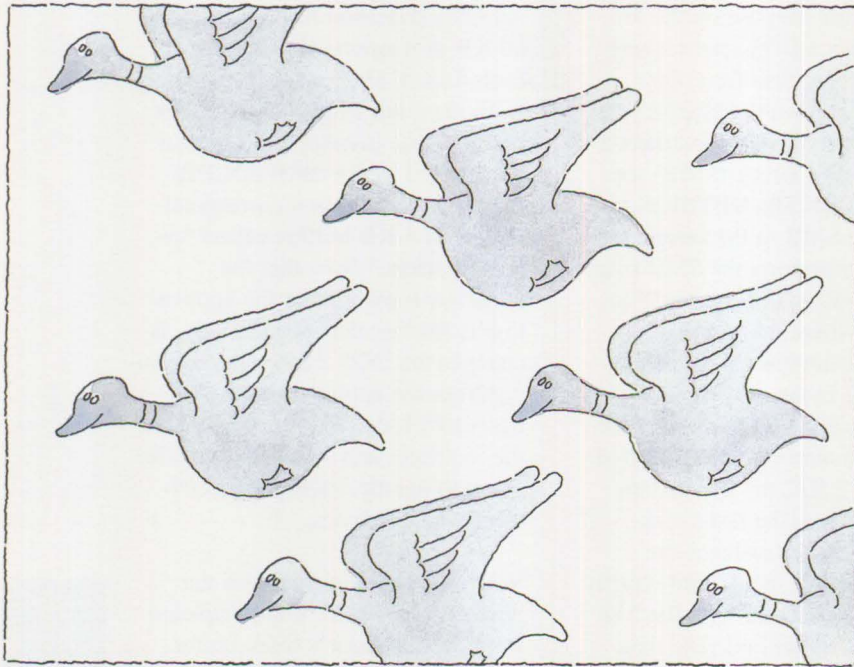
Sequential file accesses can be handled in a more efficient manner than random because it is known where the next data to be read is to be found. An application at the requester issues a read request for 512 bytes of data, like the random read just described, and is processed in the same way. After that read is complete, the application issues a read for the immediately adjacent 512 bytes. The requester code utilizes this adjacent data request as a signal that the file is being accessed sequentially, and initiates a special mode of operation called buffered read-ahead.

An OS/2 requester has, by default, 60 KB of memory set aside for buffering of SMB requests. Memory is divided into fifteen 4 KB "work buffers" (the number and size can be changed in the IBMLAN.INI file). The server has a corresponding set of 4 KB buffers called "request buffers." Now that the requester is aware that the application is reading data sequentially, it changes the DOS Read request coming from the application from 512 bytes to 4 KB. This is because both the requester and server have buffer space to handle 4 KB just as efficiently as 512 bytes.

When the application issues the next 512-byte read, it is completed from the requester's work buffer, and the SMB is not issued. The server software also detects the sequential access and goes into a complementary read-ahead of its own. This brings 4 KB from the fixed disk into another requester buffer, anticipating another 4 KB request from the application. This is why there should be at least two requester buffers in the server for each active workstation on that server.

It is not an insignificant amount of time to construct the SMB, send it to the server, process it, and get the response. Therefore, sending more data with each SMB increases server throughput and decreases response time seen by the application. What if the application is not going to read any more than two 512-byte records? The worst effect is that some data sent to the requester will never be used, at a small response-time penalty. Just as the requester derived a signal to go into buffered read-ahead operation, the buffered read-ahead is cancelled when two successive read requests are not contiguous.





The two extreme cases of file access operations and an overview of how the server/requester logic treats them have been described. Specific descriptions of server/requester buffers, caches, and some SMB protocols follow.

## Buffers, Caches, and SMB Protocols

### OS/2 Server:

#### *Requester Buffers*

The most-used buffers in OS/2 1.2 LAN Server/Requester are the request buffers (**numreqbuf**, **sizereqbuf** in the server IBMLAN.INI file). The default configuration is 36 buffers of 4 KB each. These buffers are used by the server to receive SMBs from the requesters and to hold the data being transferred to and from the network adapter card buffers. All data moved to and from the server, which originated with a read/write

request size of 4 KB or smaller, will use these buffers.

#### *Big Buffers*

The other server buffers are called simply "big buffers" (**numbigbufs** in the server IBMLAN.INI file). They are 64 KB and not changeable. They are used whenever a read/write request is made asking for an amount of data greater than the size of the request buffer (corresponds with work buffer in the requester). An example of these is the DOS Copy command, which requests 64 KB to be read and written all at one time. Another example is program load, although it may cause smaller requests than 64 KB at times. Applications, of course, may issue read/write requests in any size desired up to 64 KB (the DOS and OS/2 limit). As described previously in the section *Flow Control Logic*, server throughput can be increased and requester application response time decreased by transferring more data with each

SMB sent across the LAN. Big buffers are used in association with the SMB Raw Protocol to further enhance throughput and response time for large sequential file transfers.

#### *SMB Protocols Affecting File I/O Performance*

SMB Raw Protocol enables extremely fast data transfer across the network. The term "Raw" means that after the first SMB is sent, the remaining transmission is all data, with no SMB headers.

The SMB Raw Protocol is initiated by the following sequence. An operator at a workstation copies a file from the server to the local fixed disk. The DOS Copy command issues a read request of 64 KB to the redirected drive, which results in the redirector constructing an SMB. Detecting the large data request, the redirector issues a special SMB, which requests 4 KB of data and also polls the server for availability of big buffers. The server sends the 4 KB of data requested and confirms its availability of big buffers. The redirector issues a Read Block Raw SMB for 64 KB to the server. The server fills one of the big buffers with up to 64 KB of data and responds to the SMB Read Block Raw SMB sending 64 KB of data. Again, as in the section *Flow Control Logic*, the server detects sequential accessing of the file being copied and starts its own read-ahead of the next 64 KB of the file into another big buffer, if available. This protocol provides extremely fast data transfer across the network.

A variation to the SMB Read Block Raw Protocol is Read Block Multiplex. Consider the same conditions just described, except that neither the server nor requester has big buffers currently available. Therefore,



the data is transferred using the 4 KB request/work buffers as quickly as they can be set up and processed. This is Read Block Multiplex. Because SMBs are sent with each 4 KB of data, and 4 KB data messages are sent instead of 64 KB messages, it is not as fast as Read Block Raw. The SMB Raw Protocol can be initiated by any read/write request using a data buffer size greater than the request/work buffer size.

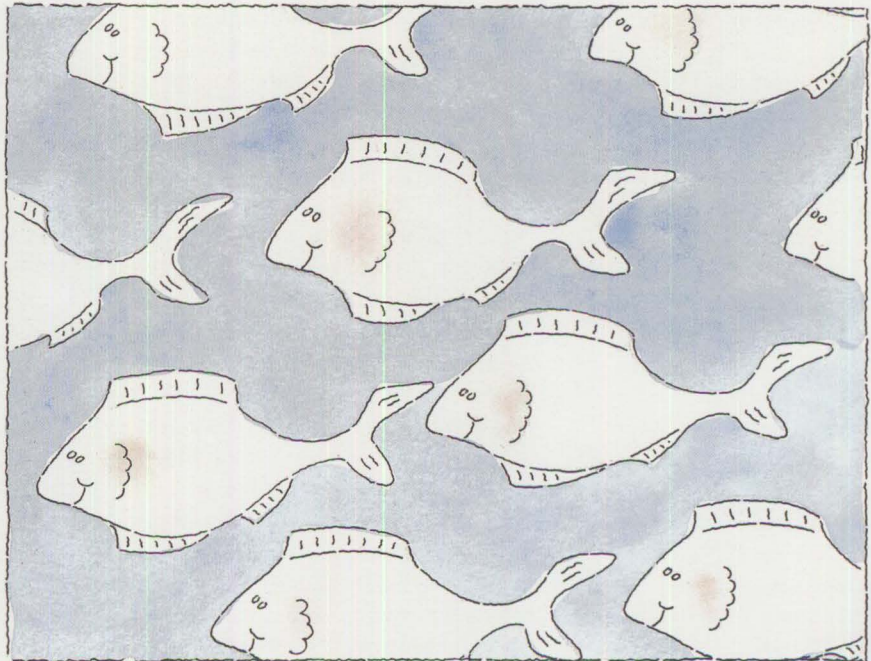
### OS/2 Requester:

#### *Work Buffers*

The OS/2 requester has by default fifteen 4 KB buffers called "work buffers" (**numwrkbuf**, **sizeworkbuf**) as well as 64 KB of memory called "work cache" (**maxwrkcache**). The work buffers are used in constructing the SMBs sent to the server and also provide the data buffering between the application running in the requester and the network adapter card. The work buffers work in conjunction with the request buffers in the server and should be the same size for best performance. The default size of 4 KB is considered to be optimum for the majority of LAN environments, but may be changed if desired. Performance may be improved in some random file access environments by increasing the number of work buffers (**numwrkbufs**), providing local data caching at the requester.

#### *Work Cache*

The memory space called work cache (**maxwrkcache**) is used in conjunction with the big buffers in the server and the SMB Raw Block Read Protocol previously described. Performance may be improved in some sequential file access environments by making work cache larger



(**maxwrkcache**, use 64 KB increments). Note that the local file system cache in the requester does not cache data requests to the redirected drive.

### DOS LAN Requester (DLR):

#### *Network Buffers*

The DLR has by default four network buffers (**NBC=4**) of 1 KB (**NBS=1K**). These buffers are used in constructing the SMBs sent to the server and also provide the data buffering between the application running in the DLR and the network adapter card. Because the DLR is limited to the 640 KB memory space of DOS, these buffers are kept at small default values to conserve memory. If application memory requirements permit, network buffer size (**NBS**) should be increased to 2 KB or even 4 KB to improve sequential file access performance for data requests of sizes less than the big buffer size (**BBS**). **NBC** should not be reduced from its default value.

#### *Big Buffers*

Like the OS/2 requester, the DLR also has big buffers (**BBS=4K**, **BBC=1**) that are used in conjunction with the server's big buffers when the SMB Raw Block Read/Write Protocol is initiated. The amount of memory allocated by default is minimized because of the 640 KB DOS memory constraint. The big buffers in the DLR are not as important as those in the OS/2 requester because of the DLR User Memory Transfer feature described in the next section. The big buffer is important in applications that perform sequential writes to the server in data buffer sizes greater than **NBS**, including DOS Copy operations directed to the server. The write data goes across the network in message sizes of **BBS**. The **BBS** is required to be larger than **NBS** or a Net Start error message will be issued. The message size used in read operations to the server from the requester is described next with the User Memory Transfer feature.



### User Memory Transfer

This is a feature from the PCLP requester, which was incorporated into the DLR. When the read data request size is greater than the BBS, the requester logic causes data to be moved directly from the network adapter card buffer to the user's memory space that was allocated for the data. This works for program load as well as application read requests. It eliminates moving the data through big buffers before sending it to the application. Unless your application issues write requests to the server in data buffer sizes greater than the NBS, big buffers do not come into play and can be set to a minimum – for example, BBS=5K if NBS=4K.

### Performance Tuning

For best performance, the server should have enough request buffers available to handle a peak request workload. A good rule of thumb is to allocate two to three request buffers (**numreqbuf**) for each requester actively sending requests to the server; this should be the typical number of busy users. The request buffers are in a pool and are shared among all requests coming to the server, so the rule-of-thumb number can be diminished as the number of active requesters gets larger. The default values (**numreqbuf=36**, **sizereqbuf=4 KB**) will easily suffice for 12 to 18 typical requesters on a server.

For best performance, the server should have enough big buffers available to handle the normal workload for large sequential file access operations. A suggested number is two big buffers (**numbigbufs**) for each requester that will be sending this type of request concurrently to the server. (These requesters are al-

most always busy handling large file transfers.) Because each big buffer is 64 KB, no more than the number of big buffers necessary to handle the typical environment should be specified.

New heuristics were added in LAN Server 1.2 (**srvheuristics 18,19**) that modify the dynamic usage of big buffers. This allows the server to request additional memory from the operating system when all the big buffers are in use and more are needed. The heuristics set the time that the server keeps memory locked for big buffers and also the frequency at which it will reissue the memory allocation if memory was not available. If the server has sufficient memory, set **numbigbufs** by the suggested number previously described for optimum performance.

#### Applications Running on the Server:

Any application that runs on the server requires CPU and I/O resources and degrades file server performance to some degree. A common application to find in the server is the Print Spooler, which services network print requests. In a set of internal test cases, file server performance degraded an average of 15 percent for random file I/O, and an average of 25 percent for sequential file I/O, when 64 KB print jobs were sent to the server at five-second intervals.

There is a growing class of applications being used in networks, often referred to as client/server applications. These usually include a high-performance computer providing processing services to less powerful computers and communicating with each other over a network using a protocol such as IBM's Advanced Program-to-Program Communication (APPC) or NETBIOS. An example is OS/2 EE 1.2 Database

Manager's Remote Data Services. Structured Query Language (SQL) queries are formulated at a Database Manager requester workstation and sent to a Database Manager server for processing. The results of the query are sent back to the Database Manager requester. Because the APPC protocol is being used, the LAN server/requester code is not required to be running on the network.

However, the point is that Remote Data Services, a LAN server Netrun application, or another client/server application, could be running in the LAN server computer, causing file server performance degradation. By default, the server executes at a higher priority than any other application. To boost the relative priority of an application running on the OS/2 1.2 LAN Server computer, **srvheuristic 6** in the IBMLAN.INI can be changed from four to five or higher. This will, of course, degrade the performance of all other OS/2 1.2 LAN Server services when the client applications are being executed.

#### Applications Running on the Requesters:

Loading DOS programs into the DLR requester from the LAN server is an efficient operation that makes use of the Raw Block Read Protocol and the User Memory Transfer feature of the DLR. Loading OS/2 programs from an OS/2 requester is not as efficient. The OS/2 Loader installs the code by segment, and the segments are not positioned in sequential order in the .EXE file. Additionally, the OS/2 Loader installs all the Dynamic Link Libraries (DLLs) associated with the .EXE file when loading OS/2 programs across a network. Only the DLLs marked "load on call" are installed during a stand-alone OS/2 program load. Design changes are in progress to improve

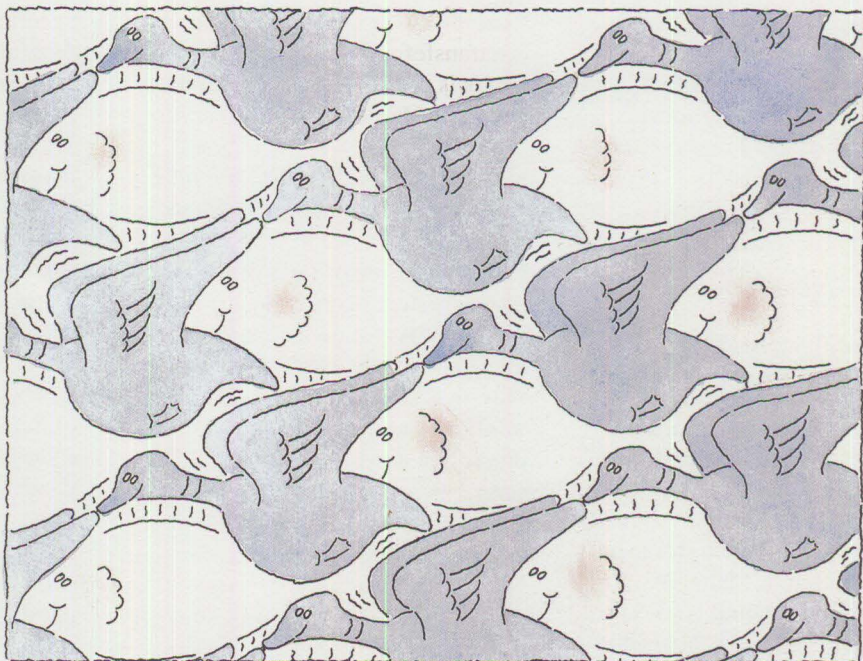


the performance of OS/2 program load across a network.

Recalling the description earlier of the caching algorithm threshold, note that for LAN Server 1.2 neither DOS nor OS/2 programs will be cached in the server. Consequently, if program load time is important in your environment (for example, a classroom), it is advisable to copy the .EXE file to a VDISK in the server, and Net Share the VDISK to the requesters. Alternatively, if you want to get a program on the server into the server's HPFS cache, you can copy the .EXE file to another place on the fixed disk. Because all HPFS "lazy" writes go through cache, the program will then reside in cache.

Usually there will be multiple requesters connected to OS/2 LAN Server 1.2 that are running the same application. An example is a database application with the database files located on the server. If these files are small with respect to the cache size, the database files will be placed into cache, resulting in good response times to the requesters. Conversely, if the database files are large with respect to the cache size, the overhead needed to place the files into cache and subsequently search the cache with few "hits" is detrimental to performance. If this sounds like your environment, reduce the cache size to 64 KB and evaluate the results.

Some database applications maintain their own cache of data likely to be used again. This is often referred to as a "buffer pool." If the database program is running on the



OS/2 1.2 LAN Server computer (client/server mode) performance may be improved by increasing the size of the buffer pool at the expense of cache memory. The idea here is that the database program knows more about the data than does the file system cache.

Many applications in use today were written for stand-alone operation with no thought given to performing file I/O across a network. Consequently, it is not uncommon, especially in DOS database applications, to see reads/writes of very small sizes and in very large quantities. These are usually random file accesses and impose a heavy workload on the server to perform file I/O and process SMBs for data requests and transmissions. To improve performance, (1) get the database files into cache, if possible, and (2) try to structure the

database queries to minimize the amount of data returned to the requester. Encourage the user to move to current database technologies such as client/server.

#### ABOUT THE AUTHOR

*Ken Whitfield is an advisory programmer at IBM's Austin Programming Center. His 26 years with IBM include experience with personal systems hardware and software in both engineering and programming capacities. He managed the Component Evaluation Laboratory in Austin. In the past three years, Ken has worked with LAN performance on PCLP 1.3, OS/2 LAN Server 1.0, and was the lead performance analyst for OS/2 LAN Server 1.2.*



# PM: An Object-Oriented Approach

Phil Spencer  
QA Training  
Cirencester, United Kingdom

**The intent of this article is to explain some of the benefits of Object Orientation (OO) methodologies and introduce some of the initial steps that can be taken down the OO path.**

Object orientation is often discussed in OS/2 Presentation Manager (OS/2 PM) programming circles today. Much of the material written today, however, is conflicting.

As an OS/2 and PM consultant at QA Training, I meet many people who would like to know more about the implications of OO in PM programming. There is no space here for a full discussion of the merits and implications of OO, nor to explain how to write PM programs in general. I assume that readers are familiar with both these subjects and are interested in pursuing them. For those requiring background information, an excellent introduction to OO is given in:

Meyer, Bertrand, *Object-Oriented Software Construction*, Prentice-Hall, 1988.

The standard introduction to PM programming is:

Petzold, Charles, *Programming the OS/2 Presentation Manager*, Microsoft Press, 1989.

I start from the perspective that OO is the way forward and is the best we can currently do to future-proof our applications.

Note that the goals of OO are independent of programming language and operating system. OS/2 PM is quite supportive of (and in some cases insistent upon) object-oriented structure. The main language used for programming OS/2 PM, however, is C, which is not considered an object-oriented language. This should not deter us. As we shall see here, it is perfectly possible to write object-oriented programs without specific language support.

## Some Historical Background

I'll begin by trying to put OO into historical perspective. Three design techniques are of interest to us.

- Top-down
- Modular
- Object-oriented

A brief discussion of each of these follows.

**Top-Down Design:** In the early 1960s the concept of top-down software design began to emerge. It was very successful when applied carefully. The idea is that the solution to complex problems is achieved by breaking them down into simpler subproblems and solving the subproblems. The technique is applied recursively until all the sub-problems have a simple solution.

The transformation from the anarchy that existed before this simple idea was introduced was pure magic!

### Strengths

When applied correctly, this is an effective development strategy for programs written by a small number (ideally one) of programmers. The

technique is ideal for single-source programs (see *Note 1* at the end of the text).

### Weaknesses

As programs became larger, they became more difficult to maintain. Programs developed by more than a small number of people were particularly difficult because no single person could grasp the full implications of a modification in the upper levels of the design. A small change in the original specification cascades down through the levels of the design and can result in almost every function having to be changed. The problem is not the complexity of the changes, but their number – in a well-structured program, the actual changes required should be well defined.

**Modular Design:** The next step was the concept of modular software design. The motivations behind modularity were admirable and simple. The idea is that when a program becomes sufficiently large that it is unmanageable as a single entity, it should be broken down into parts that are easier to manage (modules).

### Strengths

When modularity is correctly applied, applications are split into several modules with a minimal number of interdependencies. Each module can then be treated as a separate program to which top-down design techniques can be applied.

This technique considerably boosted the maximum size of program that could be considered manageable.

### Weaknesses

The problem with modularity has always been finding the ideal modular boundaries. Because overall application design is still top-down in most



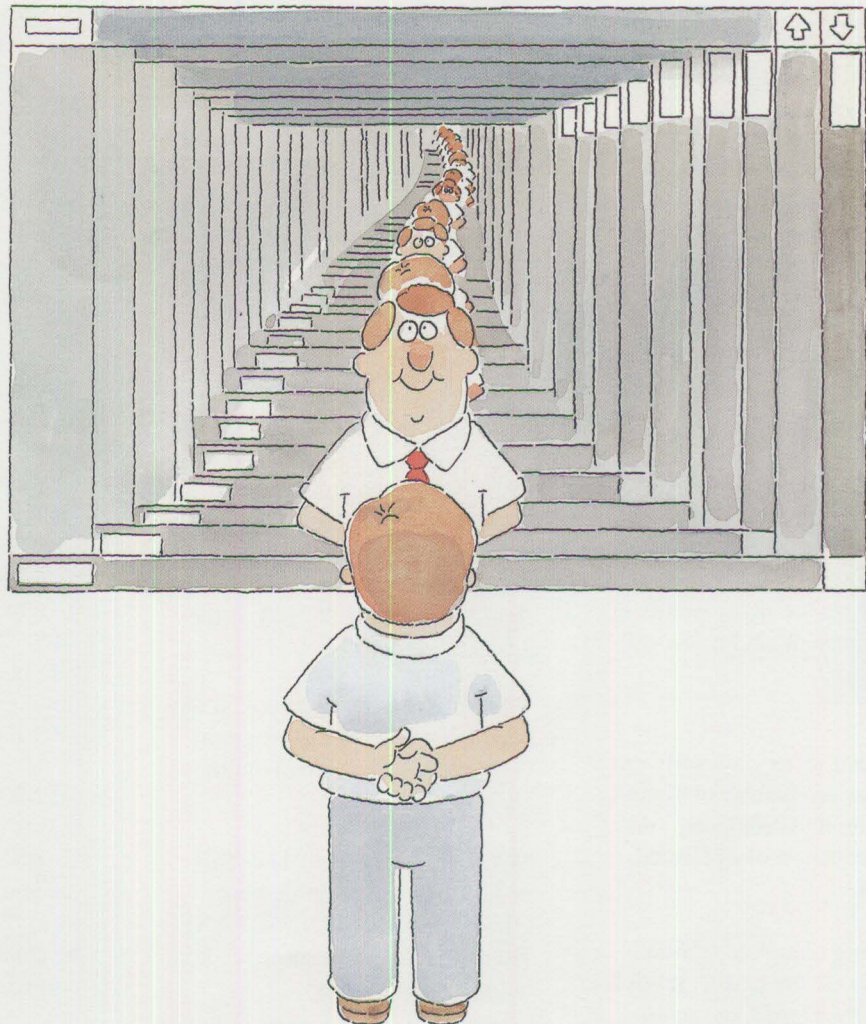
cases, it is often found that a small change in the upper levels of a design results in quite different natural modular boundaries at lower levels. This results in either unmanageable modules or requires extensive redistribution of code. I will show later how OO absorbs the modularity ideal and overcomes this problem.

Misinterpretation did much to discredit modularity. Manageability is the key to success, but unfortunately size was often allowed to influence the division of code between modules (see *Note 2*).

**Object-Oriented Design:** Object-oriented software design has its roots back in the late 1960s. It developed from projects involving simulation of real world events. We will see that the OO approach is radically different to the top-down approach. However, despite all the “forget all you’ve ever learned about designing programs” headlines that you may have seen, the ideas and experiences of modular and top-down program design are still very useful and necessary in the world of OO.

There are a few key OO ideas that need to be introduced before I can suggest a strategy for their implementation. Those of particular importance to us here are:

- The object – In the PM environment an object usually means a window, and the words are often used interchangeably. A pure object takes total responsibility for itself; there is no “higher level” of code that understands its internals.
- The class – A class is essentially a blueprint for an object. In PM, for example, we create a window with a given behavior by first defining a class that specifies the



behavior and then creating a window of that class. Several windows can (and normally will) be created of any class, which means that the class function (also called the window procedure) must be fully reentrant because it will be used by several windows simultaneously.

OO purists see objects not as an integral part of an application, but as a generic component that may be used by any application. Adopting this view means that the classes have to be the highest level of structure and completely self-contained.

This solves the problem with modular programming previously described. Classes make perfect modules because of their self-containment. There is no higher level of structure to be modified, which could make us reconsider this choice as our applications evolve.

An object of any class works by responding to events. In the PM environment an object (or window) is notified of events by the receipt of a “message” (see *Note 3*). In OO jargon, the response of an object to an event is called a “method” (see *Note 4*). Each of the methods of a



class is more or less independent, so when programming in C each method should be coded as a separate function.

Having now reduced the design to a number of individual functions, we can apply traditional top-down design to these functions. This of course, is precisely where top-down design excels. We also have the added benefit of being able to cash in on the vast wealth of experience using this technique.

This is as far as I intend to take OO ideas for the purposes of this article. I don't claim that OO is the solution to all problems – nothing is perfect. However, there are many points in its favor at present, and a few against.

#### **Strengths**

OO ideas lead us to view software design as the production of generic, reusable code units (objects), rather than the production of dedicated applications.

Classes, being complete self-contained units, are the perfect modules that can be developed and maintained in isolation. Any change in design is normally confined to a single class, and hence we overcome the major shortcoming of modular programming strategies.

The class methods are independent of each other and should be separated into individual functions when programming in C.

The design of the method functions is all that is left. Here we can do no better than to use the well-tried and tested top-down design techniques.

#### **Weaknesses**

The ideas of OO require a radically different approach to the design of

software. It is far too easy to slip back into the "old ways" and then blame OO for the resulting chaos.

Lack of widespread experience is currently the biggest problem. Great frustration exists among developers applying OO for the first time. They frequently get to the "I know this isn't quite right but who can set me right" situation.

### **Some Examples**

Let's now look at the sources for two variants of a simple PM program. The programs (DEMO1 and DEMO2) simply create a standard window with a client and four children. (These can all be seen in the figures located at the end of the article.)

**The Sources:** Take a look at the source listings for the programs. Only two files are involved in DEMO1:

**MAKEFILE** (Figure 1) – This makefile actually builds both DEMO1.EXE and DEMO2.EXE. There should be no surprises here, but note particularly:

- The use of the multithread library **llibcmt.lib**
- The compilation flags required for the use of this library
- The assumption that the **include** environment variable includes the path to the multithread C headers (normally **c:\ibmc2\include\mt**) and does not include the path to the normal C headers (normally **c:\ibmc2\include**).

**DEMO1.C** (Figures 2a through 2d) contains all of the source for the traditionally structured program.

DEMO2 takes a more object-oriented approach and as a result its

structure is radically different. This time there are seven files involved:

**MAKEFILE** – This is the same makefile as shown in Figure 1.

**MACROS.H** (Figure 3) – The macros file allows us to take some of the repetitiveness out of the coding of a PM program. I will discuss this file in more detail later.

**MAIN.C** (Figure 4) – contains only minimal startup code. This code will be almost the same for all PM programs. In English, this code says "make sure the classes **CLASS\_DEMO2** and **CLASS\_CHILD** are available and give me a **CLASS\_DEMO2** window." Notice that there is nothing else here. We assume that the window is totally capable of looking after itself.

**DEMO2.H** (Figure 5) – defines all that is public knowledge about **CLASS\_DEMO2** (that is, the class name and the class registration function). Few module interfaces could be smaller than this. Modularity has come home!

**DEMO2.C** (Figures 6a through 6c) – contains all the code for **CLASS\_DEMO2**, including the class registration function. There is a lot here. I will return to it in a moment.

**CHILD.H** (Figure 7) – defines all that is public knowledge about **CLASS\_CHILD**. This is slightly larger than **DEMO2.H**, but still small.

**CHILD.C** (Figures 8a and 8b) – contains all the code for **CLASS\_CHILD**. Again, there is a lot here, and I will return to it in a moment.



**Discussion:** First, an admission. I have deliberately structured DEMO1 to illustrate some points. Few people would write this specific application exactly in this manner, but many larger programs have a similar structure.

What has really happened in the metamorphosis from DEMO1 to DEMO2? The nature of the program has changed:

- from a *program* with a parent window and four child windows
- to a *window* of CLASS\_DEMO2

This change is much more radical than it sounds at first.

The startup code in MAIN.C is now incidental – an unfortunate necessity that would be buried deep, and out of sight, in the heart of a purer OO system. This startup code is so standard that the bulk of it can be put out of sight in a Dynamic Link Library (DLL) and forgotten.

In order to achieve this happy state of affairs, I isolated the responsibilities of the class from the responsibilities of the startup. In particular, I consider class registration as a class method because it is something that only the class should know how to do. This is reflected in the structure of DEMO2:

- Each class source contains a class registration function.
- The public interface to the class is defined in a .H file and contains an absolute minimum of information. In the most isolated classes this will contain only the name of the class and the name of the registration function.

A class of window that displays centered text "Child 1" in green letters on a dark blue background has rather limited use. Producing a class

like this, as in DEMO1, reflects an application-oriented approach – this window is a particular requirement of this application. The OO approach is to recognize the more generic case of a window that displays centered text in any colors. This is what CLASS\_CHILD does. It is a small step now to make this a public class that can be used by any application if required.

Notice the use of the control data parameter in the WinCreateWindow calls in CLASS\_DEMO2. The control data structure is a public structure that is declared in the header of the class that requires it. Don't confuse this with the internal class structure that will be required by most classes.

Careful consideration of the class functions in DEMO1 reveals that conceptually they are tables of chunks of code to be executed upon receipt of specific messages. By breaking out each of these chunks into separate C functions, the tabular structure becomes more apparent, and we find a great deal of repetitive text appearing. In the file MACROS.H I have defined some C macros that allow us to reduce the class functions to pure tables and eliminate some of the repetitiveness. The macros are:

- WINPROTO – prototypes a message or class function
- WINCALL – calls a message or class function

These macros are not particularly complex and should be easily understood by any C programmer. These macros encourage the separation of the class methods into separate C functions to which traditional top-down programming styles are applied.

There is more than this to DEMO2, but time and space prohibit a full discussion. Here is a list of some things in DEMO2 that are a benefit or feature of OO programming:

- restricted scope of local variables
- absence of global variables
- extensive use of the keyword *static* (that is, private)
- re-use of the same function names in different classes
- the use of window IDs rather than saved window handles
- the use of WM\_CREATE and WM\_DESTROY
- the allocation, use, and destruction of per-window data
- the use of control data structures

## Summary

Object orientation is the flavor of the day, perhaps of the decade. OO design seems to have much to offer even in the absence of specific language support. Don't rush to forget the experience of the last three decades though, because possibly the greatest strength of OO is that it finds a place for top-down and modular techniques that solves most of their weaknesses.

I have presented a simple application written in an application oriented style and the same program restructured to embrace some key OO ideas. As I have said, there is much more to OO than this.

*Note 1:* I have seen and worked with carefully top-down structured Algol programs of 50,000 lines or so that are much more maintainable and manageable than some of their later modular or object-oriented counterparts. The fact that such programs can be maintainable and manageable indicates that the top-down design strategy works well. The fact



that such programs can be better than modular or object-oriented counterparts is mostly a reflection of misapplication of the latter techniques.

*Note 2:* I know from personal experience that misapplication of the modularity ideas has done far more damage than good to the quality of some programs. I have worked on a number of projects in organizations where modularity was misapplied. The worst case was an operating system kernel where every function was in a different module. Each module also had a corresponding interface description file (header). The results were disastrous – there were so many intermodule links that the developers didn't have the patience to selectively include the required headers. Instead they

included all the headers in every module. This left us with a program that was effectively the same as a huge, monolithic source, but was much more difficult to maintain because almost every change involved more than one source.

*Note 3:* It is important not to confuse message basis with OO; they are two totally unrelated ideas. It just happens that the former is a convenient way of implementing the latter.

*Note 4:* In some early non-message based systems, methods were called access procedures. I still like this terminology because it always reminds me that the only viewpoint I have on an object is the one it chooses to make available to me through its access procedures (methods).

#### ABOUT THE AUTHOR

*Phil Spencer is a consultant at QA Training, specializing in OS/2 and Presentation Manager. QA supplies training worldwide in DOS, OS/2, Unix, Windows and Presentation Manager, Networking, Data Management, C and C++, and the technology of personal computer systems. For those with an interest in the topic of this article, QA can offer design standards and consultancy.*

*QA Training Ltd  
Cecily Hill Castle Cirencester  
Gloucestershire GL7 2EF  
England*

*Telephone: 011-44-285-655888*

```
all: demo1.exe demo2.exe

###
### BUILD DEMO1
###
demo1.obj: demo1.c
    cl /G2s /A1fu /W3 /c demo1.c

demo1.exe: demo1.obj
    cl $** /Fe$@ /link /nod os2 llibcmt

###
### BUILD DEMO2
###
main.obj: main.c demo2.h child.h macros.h
    cl /G2s /A1fu /W3 /c main.c

demo2.obj: demo2.c demo2.h child.h macros.h
    cl /G2s /A1fu /W3 /c demo2.c

child.obj: child.c child.h macros.h
    cl /G2s /A1fu /W3 /c child.c

demo2.exe: main.obj demo2.obj child.obj
    cl $** /Fe$@ /link /nod os2 llibcmt
```

Figure 1. MAKEFILE



```

#define INCL_WIN /* Include the Windowing subsystem interface...*/
#include "os2.h" /* ... to OS/2 PM */
/*
** This program shows some of the 'unclean' features of traditional PM programs, for example:
**
** (1) Use of global variables
** (2) Single source or unnatural distribution of code across sources
** (3) Public functions
** (4) Non function wide scope of local variables in sometimes very large window functions
** (5) Replication of data maintained by the system
** (6) Inline registration of classes: forcing class registration data to be publicized
** (7) Inline assignment (creation) of children to parents with resulting synchronization
**      problems
*/
ULONG fcf = FCF_STANDARD & ~FCF_ICON & ~FCF_ACCELTABLE & ~FCF_MENU;
CHAR szClientClass[] = "Demo1",
     szChild1Class[] = "Demo1.Child1",
     szChild2Class[] = "Demo1.Child2",
     szChild3Class[] = "Demo1.Child3",
     szChild4Class[] = "Demo1.Child4";
HWND hwndFrame, hwndClient, hwndChild1, hwndChild2, hwndChild3, hwndChild4;

MRESULT EXPENTRY Child1WndProc(HWND hwnd,USHORT msg,MPARAM mp1,MPARAM mp2) {
    MRESULT mr = (MRESULT)0;
    PSZ psz = "Child 1";
    HPS hps;
    RECTL rc1;

    switch (msg) {
    case WM_PAINT:
        hps = WinBeginPaint(hwnd,NULL,NULL);
        WinQueryWindowRect(hwnd,&rc1);
        WinDrawText(hps,-1,psz,&rc1,CLR_GREEN,CLR_DARKBLUE,DT_CENTER|DT_VCENTER|DT_ERASERECT);
        WinEndPaint(hps);
        break;
    default:
        mr = WinDefWindowProc(hwnd,msg,mp1,mp2);
    } /* endswitch */
    return mr;
}

MRESULT EXPENTRY Child2WndProc(HWND hwnd,USHORT msg,MPARAM mp1,MPARAM mp2) {
    MRESULT mr = (MRESULT)0;
    PSZ psz = "Child 2";
    HPS hps;
    RECTL rc1;

    switch (msg) {
    case WM_PAINT:
        hps = WinBeginPaint(hwnd,NULL,NULL);
        WinQueryWindowRect(hwnd,&rc1);
        WinDrawText(hps,-1,psz,&rc1,CLR_PINK,CLR_BROWN,DT_CENTER|DT_VCENTER|DT_ERASERECT);
        WinEndPaint(hps);
        break;
    default:
        mr = WinDefWindowProc(hwnd,msg,mp1,mp2);
    } /* endswitch*/
    return mr;
}

```

Figure 2a. DEMO1.C



```

MRESULT EXPENTRY Child3WndProc(HWND hwnd,USHORT msg,MPARAM mp1,MPARAM mp2) {
    MRESULT mr = (MRESULT)0;
    PSZ psz = "Child 3";
    HPS hps;
    RECTL rcl;

    switch (msg) {
    case WM_PAINT:
        hps = WinBeginPaint(hwnd,NULL,NULL);
        WinQueryWindowRect(hwnd,&rcl);
        WinDrawText(hps,-1,psz,&rcl,CLR_YELLOW,CLR_CYAN,DT_CENTER|DT_VCENTER|DT_ERASERECT);
        WinEndPaint(hps);
        break;
    default:
        mr = WinDefWindowProc(hwnd,msg,mp1,mp2);
    } /* endswitch */
    return mr;
}

MRESULT EXPENTRY Child4WndProc(HWND hwnd,USHORT msg,MPARAM mp1,MPARAM mp2) {
    MRESULT mr = (MRESULT)0;
    PSZ psz = "Child 4";
    HPS hps;
    RECTL rcl;

    switch (msg) {
    case WM_PAINT:
        hps = WinBeginPaint(hwnd,NULL,NULL);
        WinQueryWindowRect(hwnd,&rcl);
        WinDrawText(hps,-1,psz,&rcl,CLR_RED,CLR_BLUE,DT_CENTER|DT_VCENTER|DT_ERASERECT);
        WinEndPaint(hps);
        break;
    default:
        mr = WinDefWindowProc(hwnd,msg,mp1,mp2);
    } /* endswitch */
    return mr;
}

MRESULT EXPENTRY ClientWndProc(HWND hwnd,USHORT msg,MPARAM mp1,MPARAM mp2) {
    MRESULT mr = (MRESULT)0;
    HPS hps;
    RECTL rcl;
    USHORT cx;
    USHORT cy;

    switch (msg) {
    case WM_PAINT:
        hps = WinBeginPaint(hwnd,NULL,NULL);
        WinQueryWindowRect(hwnd,&rcl);
        WinFillRect(hps,&rcl,CLR_BLACK);
        WinEndPaint(hps);
        break;
    case WM_SIZE:
        cx = SHORT1FROMMP(mp2);
        cy = SHORT2FROMMP(mp2);
        WinSetWindowPos(hwndChild1,HWND_TOP,5, cy/2+5,cx/2-10,cy/2-10,SWP_MOVE|SWP_SIZE);
        WinSetWindowPos(hwndChild2,HWND_TOP,cx/2+5, cy/2+5,cx/2-10,cy/2-10,SWP_MOVE|SWP_SIZE);
        WinSetWindowPos(hwndChild3,HWND_TOP,5, 5,cx/2-10,cy/2-10,SWP_MOVE|SWP_SIZE);
        WinSetWindowPos(hwndChild4,HWND_TOP,cx/2+5, 5,cx/2-10,cy/2-10,SWP_MOVE|SWP_SIZE);
        break;
    }
}

```

Figure 2b. DEMO1.C



```

default:
    mr = WinDefWindowProc(hwnd,msg,mp1,mp2);
} /* endswitch */
return mr;
}

void main(void) {
    HAB hab;
    HMQ hmq;
    QMSG qmsg;
    SWP swp;

    hab = WinInitialize(0);
    hmq = WinCreateMsgQueue(hab,0);

    WinRegisterClass(hab,szClientClass,ClientWndProc,CS_SIZEREDRAW|CS_CLIPCHILDREN,0);
    WinRegisterClass(hab,szChild1Class,Child1WndProc,CS_SIZEREDRAW,0);
    WinRegisterClass(hab,szChild2Class,Child2WndProc,CS_SIZEREDRAW,0);
    WinRegisterClass(hab,szChild3Class,Child3WndProc,CS_SIZEREDRAW,0);
    WinRegisterClass(hab,szChild4Class,Child4WndProc,CS_SIZEREDRAW,0);

    hwndFrame = WinCreateStdWindow(
        HWND_DESKTOP,                /* Parent */
        WS_VISIBLE,                  /* Frame style */
        &fcf,                         /* Frame Content Flags */
        szClientClass,               /* Client Class Name */
        " - Demo1",                  /* Titlebar text */
        OL,                           /* Client style */
        (HMODULE)NULL,               /* Resource Module Handle */
        0,                             /* Resource ID */
        &hwndClient);                /* Returns Client Window Handle */

    hwndChild1 = WinCreateWindow(
        hwndClient,                  /* Parent */
        szChild1Class,               /* Class Name */
        NULL,                         /* Window Text */
        WS_VISIBLE,                  /* Window Style */
        0,0,                          /* Position */
        0,0,                          /* Size */
        NULL,                         /* Owner */
        HWND_TOP,                    /* Placement */
        0,                             /* ID */
        NULL,                         /* Control Data Pointer */
        NULL);                        /* Presentation Parameters Pointer */

    hwndChild2 = WinCreateWindow(
        hwndClient,                  /* Parent */
        szChild2Class,               /* Class Name */
        NULL,                         /* Window Text */
        WS_VISIBLE,                  /* Window Style */
        0,0,                          /* Position */
        0,0,                          /* Size */
        NULL,                         /* Owner */
        HWND_TOP,                    /* Placement */
        0,                             /* ID */
        NULL,                         /* Control Data Pointer */
        NULL);                        /* Presentation Parameters Pointer */

    hwndChild3 = WinCreateWindow(
        hwndClient,                  /* Parent */
        szChild3Class,               /* Class Name */
        NULL,                         /* Window Text */

```

Figure 2c. DEMO1.C



```

        WS_VISIBLE,
        0,0,
        0,0,
        NULL,
        HWND_TOP,
        0,
        NULL,
        NULL);
    hwndChild4 = WinCreateWindow(
        hwndClient,
        szChild4Class,
        NULL,
        WS_VISIBLE,
        0,0,
        0,0,
        NULL,
        HWND_TOP,
        0,
        NULL,
        NULL);

    /* Force a WM_SIZE to position children */
    WinQueryWindowPos(hwndFrame,&swp);
    WinSetWindowPos(hwndFrame,HWND_TOP,swp.x,swp.y,swp.cx+1,swp.cy+1,SWP_MOVE|SWP_SIZE|SWP_ZORDER);

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

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

```

```

/* Window Style */
/* Position */
/* Size */
/* Owner */
/* Placement */
/* ID */
/* Control Data Pointer */
/* Presentation Parameters Pointer */

/*Parent */
/*Class Name */
/* Window Text */
/* Window Style */
/* Position */
/* Size */
/* Owner */
/* Placement */
/* ID */
/* Control Data Pointer */
/*Presentation Parameters Pointer */

```

Figure 2d. DEMO1.C



```

/*
** SOME MACROS TO ASSIST WITH OUR CODE STANDARDS
*/
#define WINPROTO(name) MRESULT EXPENTRY name(HWND hwnd,USHORT msg,MPARAM mp1,MPARAM mp2)
#define WINCALL(name) name(hwnd,msg,mp1,mp2)

```

Figure 3. MACROS.H

```

#include "process.h"           /* C runtime library interface */
#include "os2.h"              /* OS/2 PM interface */
#include "macros.h"          /* Application wide macro assistance */
#include "demo2.h"          /* Our own calss interface description */
#include "child.h"          /* CHILD_CLASS interface description */

void main(void) {
    /*
    ** A standard PM 'main'. Every PM program will have a 'main' that looks
    ** almost identical to this.
    */
    QMSG qmsg;
    ULONG fcf = FCF_STANDARD & ~FCF_ACCELTABLE & ~FCF_MENU & ~FCF_ICON;
    HAB hab = WinInitialize(0);
    HMQ hmq = WinCreateMsgQueue(hab,0);
    BOOL brc = RegisterClassDemo2(hab)
                && RegisterClassChild(hab);
    HWND hwnd = WinCreateStdWindow(
        HWND_DESKTOP,           /* Parent */
        WS_VISIBLE,           /* Frame style */
        &fcf,                 /* Frame Content Flags */
        CLASS_DEMO2,         /* Client Class Name */
        " - Demo2",          /* Titlebar text */
        OL,                   /* Client style */
        (HMODULE)NULL,       /* Resource Module Handle */
        0,                   /* Resource ID */
        NULL);               /* Returns Client Window Handle */

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

    WinDestroyWindow(hwnd);
    WinDestroyMsgQueue(hmq);
    WinTerminate(hab);
    exit(0);
}

```

Figure 4. MAIN.C



```

/*****
/** THIS FILE DECLARES ALL THE PUBLIC INTERFACES TO CLASS_CHILD **/
*****/

/***** The Class Name Required by 'WinCreate[Std]Window' *****/
#define CLASS_DEMO2 "Demo2 Client Class"

/***** The Class Registration Function Reauired by 'main' *****/
BOOL RegisterClassDemo2(HAB hab);

```

Figure 5. DEMO2.H

```

#define INCL_WIN /* Include the Windowing subsystem interface...*/
#include "os2.h" /* ...to OS/2 PM */
#include "macros.h" /* Application wide macro assistance */
#include "demo2.h" /* Our own class interface description */
#include "child.h" /* CHILD_CLASS interface description */

/***** C l a s s M e t h o d *****/

static WINPROTO(Wm_Create) {
/*
** Note one of the rare correct usages of global data - constant tables.
** This one is a table of the colors for the children.
*/
static struct {LONG fclr; LONG bclr;} clr[] = {
    { CLR_GREEN, CLR_DARKBLUE },
    { CLR_PINK, CLR_BROWN },
    { CLR_YELLOW, CLR_CYAN },
    { CLR_RED, CLR_BLUE },
};
USHORT id; /* Window ID */
SEL sel; /* Selector */
PCHILD_CTRLDATA pccd; /* Pointer to a CHILD_CTRLDATA */

/*
** Allocate a child control data structure, initialize it, and create the child, passing the
** address of the structure. Note it is the responsibility of the child to take its own copy
** of the control data structure during its WM_CREATE processing. We are therefore at liberty
** to free or reuse our structure anytime after the 'WinCreateWindow' call.
*/
DosAllocSeg(sizeof(CHILD_CTRLDATA), &sel, SEG_NONSHARED);
pccd = MAKEP(sel, 0);
for( id=0; id<4; id++ ) {
    pccd->ChildNo = id+1;
    pccd->ClrText = clr[id].fclr;
    pccd->ClrBack = clr[id].bclr;
    WinCreateWindow(
        hwnd, /* Parent */
        CLASS_CHILD, /* Class Name */
        NULL, /* Window Text */
        WS_VISIBLE, /* Window Style */
        0, 0, /* Position */
        0, 0, /* Size */
        NULL, /* Owner */
        HWND_TOP, /* Placement */
    );
}
}

```

Figure 6a. DEMO2.C



```

        id,                /* ID */
        pccd,             /* Control Data Pointer */
        NULL);          /* Presentation Parameters Pointer */
    } /* endfor */
    DosFreeSeg(sel);
    return NULL;
    if (hwnd||msg||mp1||mp2) /* Suppress compiler 'unreferenced...' messages */;
}

/***** C l a s s M e t h o d *****/

static WINPROTO(Wm_Paint) {
    HPS hps;
    RECTL rcl;

    /*
    ** Paint our window. In this example simply fill the window with black.
    */
    hps = WinBeginPaint(hwnd,NULL,NULL);
    WinQueryWindowRect(hwnd, &rcl);
    WinFillRect(hps,&rcl,CLR_BLACK);
    WinEndPaint(hps);
    return NULL;
    if (hwnd||msg||mp1||mp2) /* Suppress compiler 'unreferenced...' messages */;
}

/***** C l a s s M e t h o d *****/

static WINPROTO(Wm_Size) {
    USHORT id;
    USHORT cx = SHORT1FROMMP(mp2);
    USHORT cy = SHORT2FROMMP(mp2);
    SWP aswp[4];

    /*
    ** Our window has been resized - resize our children. This is normal WM_SIZE processing.
    ** Note that in general, windows are only resized by their parents.
    */
    for (id=0;id<4;id++) {
        aswp[id].fs = SWP_MOVE|SWP_SIZE;
        aswp[id].x = (id==0)||id==2? 5: cx/2+5;
        aswp[id].y = (id==2)||id==3? 5: cy/2+5;
        aswp[id].cx = cx/2-10;
        aswp[id].cy = cy/2-10;
        aswp[id].hwndInsertBehind = HWND_TOP;
        aswp[id].hwnd = WinWindowFromID(hwnd,id);
    } /* endfor */
    WinSetMultWindowPos(WinQueryAnchorBlock(hwnd),aswp,4);
    return NULL;
    if (hwnd||msg||mp1||mp2) /* Suppress compiler 'unreferenced...' messages */;
}

/***** T h e C l a s s F u n c t i o n *****/

static WINPROTO(fnwp) {
    /*
    ** The Class Function is effectively a simple 'jump table'! I would like to have left that
    ** comment at that simple statement, but the casual reader may not have grasped the
    ** implications:
    ** (1) It is ripe for automation - name the messages you're interested in and the class
    ** function is fully specified
    */
}

```

Figure 6b. DEMO2.C



```

** (2) It is ripe for optimization - as implemented here it is a 'call table' rather than
**      a 'jump table'
** (3) In the light of 1 & 2 let your dreams run wild - automatically generated optimal
**      code!? As a first step I have a tiny macro assembler source which allows me to
**      declare this class function as follows:
**          INCLUDE MASMFWP.INC
**          FNWP <WM_CREATE,WM_PAINT,WM_SIZE>
**          END
**      more of this in a future article perhaps
*/
switch(msg) {
case WM_CREATE:      return WINCALL(fnmsgWmCreate);
case WM_SIZE:       return WINCALL(fnmsgWmSize);
case WM_PAINT:      return WINCALL(fnmsgWmPaint);
default:            return WINCALL(WinDefWindowProc);
} /* endswitch */
}

/***** T h e C l a s s R e g i s t r a t i o n *****/

BOOL RegisterClassDemo2(HAB hab) {
/*
** The provision of a class registration function like this is recommended. It could be
** regarded as a class method but since it is invoked explicitly by external call rather
** than in response to a message it is better to describe it as an 'access procedure' as
** mentioned in the article. The purpose of this function is to hide more class internals.
**
**      - The window procedure can be made 'static'
**      - The class styles no longer have to be public
**      - The number of per window bytes no longer has to be public
*/
return WinRegisterClass(
                hab,                /* Anchor Block Handle */
                CLASS_DEMO2,        /* Class Name */
                fnwp,               /* Class Function */
                CS_SIZEREDRAW | CS_CLIPCHILDREN, /* Class Styles */
                0);                 /* # per window bytes */
}

```

Figure 6c. DEMO2.C



```

/*****
** THIS FILE DECLARES ALL THE PUBLIC INTERFACES TO CLASS_CHILD **
*****/

/***** The Class Name Required by 'WinCreate[Std]Window' *****/
#define CLASS_CHILD "Child Class"

/***** The Class Registration Function Required by 'main' *****/
BOOL RegisterClassChild(HAB hab);

/***** The Class Control Data Structure Required by 'WinCreateWindow' *****/

typedef struct {
    USHORT   ChildNo;
    LONG     ClrText;
    LONG     ClrBack;
} CHILD_CTRLDATA, *PCHILD_CTRLDATA;

```

Figure 7. CHILD.H

```

#include "stdio.h"           /* C runtime library interface */
#define INCL_WIN             /* Include the Windowing subsystem interface... */
#include "os2.h"            /* ...to OS/2 PM */
#include "macros.h"         /* Application wide macro assistance */
#include "child.h"          /* Our own class interface description */

/***** The Class Private Data Structure *****/

typedef struct {
    CHILD_CTRLDATA ccd;     /* The Class Public Data Structure */
} CLASSSTRUCT, *PCLASSSTRUCT;

/***** Class Method *****/

static WINPROTO(Wm_Create) {
    SEL          sel;       /* Selector */
    PCHILD_CTRLDATA pccd = PVOIDFROMMP(mp1); /* Get ctrldata address */
    PCLASSSTRUCT pcs;      /* Pointer to a CLASSSTRUCT */

    /*
    ** Allocate a CLASSSTRUCT and save its address in the per window data
    */
    DosAllocSeg(sizeof(CLASSSTRUCT), &sel, SEG_NONSHARED);
    WinSetWindowPtr(hwnd, 0, pcs=MAKEP(sel, 0));
    /*
    ** Initialize the CLASSSTRUCT
    */
    pcs->ccd = *pccd;
    return NULL;
    if (hwnd|msg|mp1|mp2) /* Supress compiler 'unreferenced...' messages */;
}

/***** Class Method *****/

static WINPROTO(Wm_Destroy) {
    PCLASSSTRUCT pcs = WinQueryWindowPtr(hwnd, 0); /* Get CLASSSTRUCT address */
    /*
    ** Normally we would then free any resources described in the CLASSSTRUCT here AND THEN

```

Figure 8a. CHILD.C



```

** free the structure itself. There are no resources that need freeing here so we only
** have to free the structure.
*/
DosFreeSeg(SELECTOROF(pcs));
return NULL;
if (hwnd||msg||mp1||mp2) /* Suppress compiler 'unreferenced...' messages*/;
}

/***** C l a s s M e t h o d *****/

static WINPROTO(Wm_Paint) {
    PCLASSSTRUCT pcs = WinQueryWindowPtr(hwnd,0); /* Get CLASSSTRUCT address */
    char buf[32];
    HPS hps;
    RECTL rcl;

    /*
    ** Paint our window. In this example simply draw some text with WinDrawText. Note how we
    ** extract the required colors etc. from the per window data thus making this class generic
    ** and not having to use register a new class for each variant.
    */
    hps = WinBeginPaint(hwnd,NULL,NULL);
    WinQueryWindowRect(hwnd,&rcl);
    sprintf(buf,"Child %d",pcs->ccd.ChildNo);

    WinDrawText(hps,-1,buf,&rcl,pcs->ccd.ClrText,pcs->ccd.ClrBack,
        DT_CENTER|DT_VCENTER|DT_ERASERECT);
    WinEndPaint(hps);
    return NULL;
    if (hwnd||msg||mp1||mp2) /* Supress compiler 'unreferenced...' messages */;
}

/***** T h e C l a s s F u n c t i o n *****/

static WINPROTO(fnwp) {
    /*
    ** The Class Function is effectively a simple 'jump table"!
    ** See notes in DEMO2.C
    */
    switch(msg) {
        case WM_CREATE: return WINCALL(Wm_Create);
        case WM_DESTROY: return WINCALL(Wm_Destroy);
        case WM_PAINT: return WINCALL(Wm_Paint);
        default: return WINCALL(WinDefWindowProc);
    } /* endswitch */
}

/***** T h e C l a s s R e g i s t r a t i o n *****/

BOOL RegisterClassChild(HAB hab) {
    /*
    ** The provision of a class registration function like this is recommended.
    ** See notes in DEMO2.C
    */
    return WinRegisterClass(
        hab, /* Anchor Block Handle */
        CLASS_CHILD, /* Class Name */
        fnwp, /* Class Function */
        CS_SIZEREDRAW, /* Class Styles */
        sizeof(PCLASSSTRUCT)); /* # per window bytes */
}

```

Figure 8b. CHILD.C



# DOS 4.00 SHARE

Bill Lawton  
IBM Corporation  
Boca Raton, Florida

This article discusses the utility SHARE and its function within DOS. In fact, the behavior of DOS changes depending on whether SHARE is loaded and the method used to load it. Because the function of SHARE was not usually required or used much before version 4.0 of DOS (except with file servers), it remains a mystery to most users. This article explains the various functions and methods of loading SHARE.

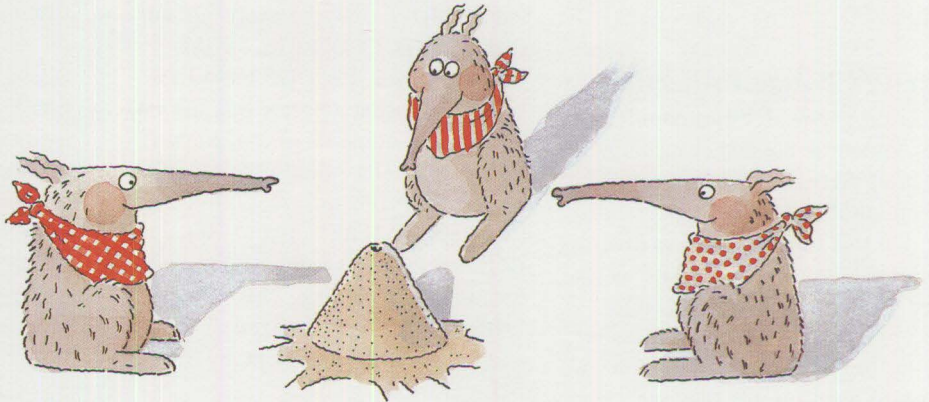
SHARE is an extension of DOS that provides four major functions:

- File sharing – SHARE is required if the machine is a PC local area network (LAN) file server
- File locking
- Diskette change protection
- Support for File Control Blocks (FCB) file I/O on fixed-disk partitions greater than 32 MB.  
SHARE is required if any partitions are greater than 32 MB.

Because SHARE is separately loadable from DOS itself, memory can be saved if it is not needed.

## File Sharing and File Locking

Before the introduction of DOS 4.00, SHARE was mostly used on file servers in networking environments to allow files to be shared among several users or running programs. SHARE provides the file-sharing capability in two different ways.



First, SHARE enforces file-sharing rules on all DOS OPEN calls. (Refer to *IBM Disk Operating System Version 4.00 Technical Reference*, 15F1420, for a list of sharing modes available in DOS 4.00.)

Second, it activates the file-locking capability of DOS. When SHARE is not loaded, the DOS FILE LOCK calls (INT 21H, function 5CH) do not perform any function, but return without error. File-locking is not used by the PC LAN, but is used by applications to lock portions of a file to prevent access by other users or programs. This allows more than one program to open a file and read from it, but not change a locked section of it, as in a data base application.

On the other hand, the file-sharing rules of a DOS OPEN restrict programs from opening a file already opened by another program, depending on the sharing modes specified on the open call. In DOS, this is called a *sharing violation*. DOS retries the open call several times and, if still unsuccessful, returns a sharing violation to the calling program. The application must also retry the open until it can gain access to the file, or return an error to the user.

The advantage of using file-locking is that a portion of the file can be locked so other programs cannot change it, but still have access to other sections of the file. Therefore, SHARE provides two different methods of sharing files between users and applications.

There are two side effects of the file-sharing rules enforced by SHARE. The first deals with the DOS RENAME (INT 21H, function 56H) and DOS Change File Mode (INT 21H, function 43H) functions. If a file is opened in compatibility mode when SHARE is loaded, then a DOS RENAME or DOS Change File Mode (CHMOD) to that file causes SHARE to close the file. Some applications that run without SHARE loaded may not expect this to happen, but it is correct behavior. DOS closes the file to force the changes to the file to take effect. If the file had been opened under a different sharing mode, the DOS RENAME or DOS CHMOD would have failed, thereby enforcing the file-sharing rules. Compatibility mode does not restrict a second open of an already open file.

The second side effect deals with the DOS DELETE function. If SHARE is loaded, DOS will not delete a file previously opened with a



sharing mode that restricts writes to the file.

### Diskette Change Protection

A function of SHARE not commonly known is the diskette change protection. SHARE does not actually contain the diskette change logic, but this logic in DOS is inactive (turned off) unless SHARE is loaded. This function allows DOS to ensure that users do not remove a diskette from a disk drive while a file is still open on that diskette. This will prevent the data corruption that could occur without this protection.

Part of this diskette change protection involves keeping track of the number of open files on a diskette. When a diskette change is detected by DOS, this number is checked. If it is not zero, DOS displays an error, requesting that the previous diskette be reinserted in the diskette drive. This prevents any running applications from reading or writing to the wrong diskette. However, this also creates a compatibility problem with some existing applications.

Some applications written before this function was added to DOS open files during installation, but never close them before asking the user to insert the next installation diskette. These applications should be used without SHARE loaded or by allowing DOS to load SHARE if you have a fixed-disk partition greater than 32 MB.

### FCB Support on Fixed-Disk Partitions Greater than 32 MB

The last major function of SHARE, and one of the most misunderstood, is the "greater than 32 MB" support for FCBs. This particular function is new with version 4.00 of DOS.

DOS versions prior to 4.00 did not support fixed-disk partitions greater than 32 MB. One reason for this limitation was that DOS used 16 bits to represent a sector number on a disk. The contents of a disk are divided into units called *sectors*. DOS reads and writes information in 512-byte sectors. With version 4.00, DOS uses 32 bits to represent a sector number on a disk. This allows DOS to access larger fixed disks as one partition instead of many 32 MB partitions.

This change to DOS did not come without some compatibility problems. Many applications, such as low-level, disk data analysis programs and disk utility programs, depended on DOS using 16-bit sector numbers, so they had to be modified to work with DOS 4.00.

Another compatibility problem created by this change in sector number size deals with FCBs. Using SHARE fixes the problem.

The use of FCBs is one of two methods of accessing files in DOS. The other method is the use of file handles. An FCB is actually a data structure that contains information about an open file. It does not reside in DOS itself, but in an application program's memory. (Refer to the *Disk Operating System 4.00 Technical Reference* for the contents of an FCB.) Applications pass an FCB to DOS whenever accessing a file using the FCB DOS calls. One of the fields in an FCB is a 16-bit sector number representing the location of an open file. Because DOS 4.00 must remain compatible with previous versions, the FCB structure could not be changed to accommodate 32-bit sector numbers. Therefore, DOS needed to keep the other 16 bits of the 32-bit sector number internally invisible to the ap-

plication program and the FCB structure.

When DOS opens a file, using either FCBs or file handles, it keeps an internal file table entry of information about the file. With file handles, all handles referring to the same file point to the same internal file table entry. With FCBs, the information is kept in an internal file table entry only temporarily during the execution of the DOS command on that particular file. The information is always copied back to the FCB after each DOS call.

DOS always regenerates the internal file table entry with each FCB call. For DOS to keep the other 16 bits of the sector number as just discussed, each FCB OPEN must be directly tied to a specific file table entry. The entry keeps the 16-bit number. But this means DOS must keep the file table entry intact between DOS FCB calls. The file table entry cannot be regenerated by DOS from the FCB. This also means that an infinite number of FCB opens can no longer be handled since each FCB must map to a corresponding internal file table entry. (Refer to the FCBS CONFIG.SYS command in *Disk Operating System 4.00 Command Reference*, 6280253, for more information on the limits imposed on the number of FCBs.)

These restrictions are actually already a part of SHARE and how it implements file sharing for FCBs and file handles. This is the reason SHARE is needed for greater than 32 MB support using FCBs. As you can see, FCBs behave differently when SHARE is loaded as opposed to when it is not. It is important to understand these differences and how they relate to the fixed-disk partitions larger than 32 MB.



## Loading SHARE

SHARE has two parameters:

- **/F:**filespace – Allocates memory for file name information necessary for the file-sharing function of SHARE. Filespace is represented in bytes with a default value of 2048 bytes.
- **/L:**locks – Specifies the maximum number of locks needed. The default value is 20 locks.

There are several ways to load SHARE:

- For PC LAN file servers, SHARE will be loaded if found by the network start-up software at boot-up time. The parameters for share can be specified when loading the network program. (Refer to the *PC Local Area Network Program*, 84X0162, for the correct syntax.)
- Include the following in the CONFIG.SYS:  
INSTALL=SHARE.EXE
- Include the following in the AUTOEXEC.BAT:  
SHARE.EXE
- Type SHARE from the command line immediately after boot-up. SHARE should always be loaded before running any application that expects to use any function of SHARE.
- For fixed-disk partitions greater than 32 MB, DOS will attempt to load SHARE from either the root directory or the directory specified on the SHELL= statement in the CONFIG.SYS if SHARE is not already loaded. If DOS cannot find SHARE to load, it will send a warning message to the display indicating that SHARE should be loaded for “greater than 32 MB” support. This all

happens after the CONFIG.SYS file has been parsed and the commands inside processed. This method loads SHARE with default parameters; therefore, if the default is not desired, use the INSTALL= method to load SHARE and specify any parameters on that line in the CONFIG.SYS. (Refer to the *Disk Operating System 4.00 Technical Reference* for the correct syntax.)

What are the differences?

All the preceding methods of loading SHARE are equivalent, except for the method dealing with fixed-disk partitions greater than 32 MB. If DOS determines that the machine has at least one fixed-disk partition greater than 32 MB, it will attempt to load SHARE. Because the “greater than 32 MB” support requires only the internal data structure kept in SHARE, the file-sharing, file-locking, and diskette change protection remain inactive (turned off). Some applications do not expect any form of file-sharing rule to be enforced during file opens, and therefore may not work with SHARE loaded unless this function is inactive. This is why DOS loads a partial function version of SHARE if it detects a “greater than 32 MB” partition, and SHARE is not already loaded.

If any of the inactive functions are needed, one of the other methods to load SHARE during bootup must be used; or, SHARE can be loaded from the command line. If SHARE is loaded from the command line, you can specify parameters for SHARE, but they will not take effect because SHARE is already loaded. It will, however, trigger the original SHARE to activate the inactive functions.

There is one other difference between the last method of loading SHARE and the others. Many users try to check for the presence of SHARE by trying to load SHARE from the command line. If SHARE was previously loaded, DOS will return with an error message “SHARE already installed,” because SHARE can only be loaded once on a PC. This indicates that SHARE is loaded, but this will work only if SHARE was loaded explicitly by the user using one of the methods described earlier. It will not work if DOS loaded SHARE for “greater than 32 MB” support (the last method just described). This is correct behavior, and it will prevent error conditions in applications such as the PC LAN program that load SHARE and do not expect SHARE to be already loaded.

## Summary

As fixed-disk sizes increase, more users will need to load some form of SHARE for “greater than 32 MB” support. This article has explained some of the problems that can arise while using SHARE and how to load the function of SHARE that is needed.

## ABOUT THE AUTHOR

*William W. Lawton is a senior associate programmer at IBM's Entry Systems Division in Boca Raton, Florida, working as technical team leader in Multimedia Software Development. He joined IBM in 1987, and until recently, he was technical team leader in the DOS Kernel Development department. Bill received his B.S. in computer science from Purdue University.*



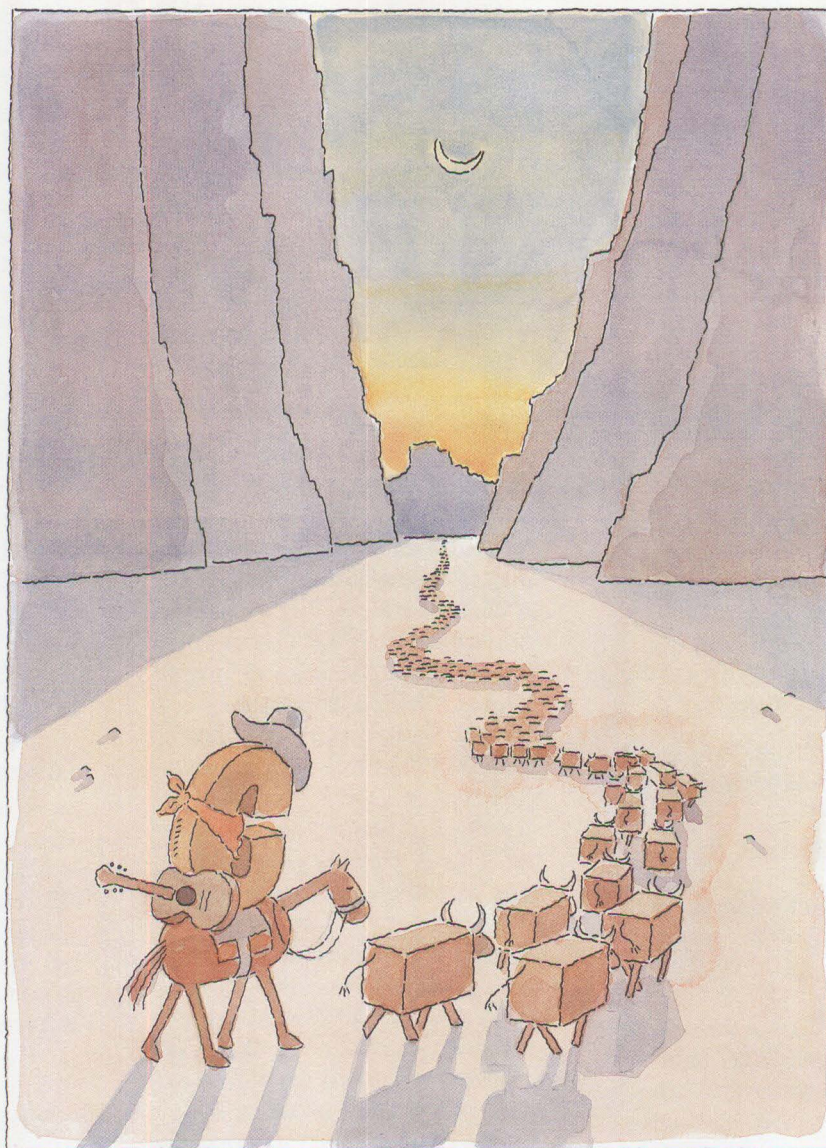
# A "C" Programming Model for DOS Device Drivers

Dan Feriozi  
IBM Corporation  
Boca Raton, Florida

**A practical and proven method of writing DOS device drivers in the C programming language is presented here. The C language was chosen because of its documented suitability as a systems programming language and because of its universal availability for use on small systems.**

The recent explosive growth in the number of new and different types of devices for use with personal computers has put pressure on software engineers to come up with new and better ways of developing software to run the devices. For DOS device drivers, a simple and fundamental improvement in the development process would be to code in a high-level language rather than to follow the current practice of using assembly language. The problem that many developers may face is that it is not immediately apparent that it is possible to code DOS device drivers in a language other than assembly.

This article sets forth the general outline for a C language programming model for DOS device drivers. Some knowledge of device drivers in general, and DOS device driver specifically, is assumed. Refer to the DOS Technical Reference for more information about DOS device drivers. This programming model was developed for use with the IBM C/2™ compiler. It



should translate to the Microsoft C® compiler with little or no modification. The general concepts presented should work with any C compiler; however, specific details of implementation may vary somewhat from one compiler to another. Consult your compiler reference manual to resolve any discrepancies.

The current DOS model for device drivers assumes that they will be written in assembly language. As a

result, it is not possible to write a DOS device driver completely in C, using the IBM C/2 compiler. The programming model set forth here provides the ability to develop DOS device drivers largely in C. The exact ratio of C code to assembly code in the device driver is unimportant. The primary concern is that sections of code that require high-level data and control structures are written in C, whereas sections of code that require hardware access or



very high performance are written in assembly.

### Problems to Be Solved

The C compiler creates object modules that are meant to be linked together to form an executable program. The problem is that a device driver is not an executable program. It is a special type of file that resembles a library module more than anything else. The programming model used for a DOS device driver must be able to do the following:

- Logically group all code and data into one segment
- Eliminate the C startup routines
- Eliminate the C runtime
- Convert a register calling convention into a stack calling convention
- Discard the initialization code

These tasks are accomplished by an assembly entry module in conjunction with a set of link and compile flags.

### Device Driver Structure

The device driver source code consists of the following modules, at a minimum:

- DOSDD.ASM – Main entry module
- COMMANDS.C – Device driver commands
- IOCTL.C – Ioctl commands
- RUNTIME.ASM – Assembly subroutines
- INIT.C – Device driver initialize command

Each module is described separately in order to build a picture of the device driver from its components.

DGROUP	group	_DATA, _BSS, CONST, _TEXT, INIT
_DATA	segment	word PUBLIC 'DATA'
	assume	ds:DGROUP
_DATA	ends	
_BSS	segment	word PUBLIC 'BSS'
	assume	ds:DGROUP
_BSS	ends	
CONST	segment	word PUBLIC 'CONST'
	assume	ds:DGROUP
CONST	ends	
_TEXT	segment	word PUBLIC 'CODE'
	assume	cs:DGROUP
_TEXT	ends	
INIT	segment	word PUBLIC 'INIT'
	assume	cs:DGROUP, ds:DGROUP
INIT	ends	

Figure 1. Beginning DOSDD.ASM Code

**DOSDD.ASM:** This is the main module of the device driver. It performs the following functions:

- Declares, groups, and orders all segments
- Provides the device driver header
- Provides the strategy and interrupt entry points
- Routes the device driver command

This module begins with the assembly code shown in Figure 1.

The default compiler-produced segments are grouped and ordered by this code. The DOS device driver model requires that the strategy and interrupt points be in the same segment as the device header. This is the reason for including the code segment, TEXT, in with DGROUP. Discardable initialization code and data is placed in the INIT segment. This segment is also included in DGROUP so that it may be ac-

cessed within the Small Model framework.

The address of the end of code is defined in the TEXT segment. This is because most of the initialization code is written in C. Because the INIT segment follows the TEXT segment, it is discarded entirely. The INIT segment contains all of the initialization data as well as any assembly initialization routines that may be required.

It should be clear at this point that the proper definition and ordering of segments is crucial to allowing the initialization code and data to be discarded. In addition, it is necessary to order module linking to properly size the TEXT segment. The initialization code is sequestered in the INIT.C module for this reason. The link statement must specify INIT.OBJ as the last link module so that no code will follow it and be discarded accidentally. The link statement should be something like that shown in Figure 2.



The main module must be specified first, and the init module must be specified last. The order of the other modules is not important.

The second major function of the DOSDD.ASM module is to define the device header at the beginning of the device driver. The device driver header is defined as the first data object in the DATA segment. Because the DATA segment is the first segment of the DGROUP and the DGROUP is the only group of the device driver, and DOSDD is the first link module, the device header appears at offset zero of the device driver, as required by the DOS device driver model.

The strategy and interrupt entry points are also defined in the DOSDD.ASM module. The entry points to the device driver must be defined in an assembly module in order to convert the DOS register parameter-passing protocol to a stack-based protocol that can be recognized by C functions. The strategy routine saves the address of the request packet for later use by the interrupt routine. The address of the request packet is contained in the ES:BX register pair, which is saved in the data area as a segment and an offset.

The interrupt routine establishes a C language environment and routes the device driver command to a C subroutine that does the work of satisfying the request. The address of the request packet is converted to a far-pointer parameter to the C function by pushing the previously saved segment and offset onto the

stack. Control is transferred to the C routine by making an indirect call through a call table of DOS device driver command codes. The function of the interrupt routine is summarized as follows:

1. Save the entry context.
2. Switch to a local stack.
3. Restore ES:BX from the saved data areas.
4. Verify the DOS command code.
5. Push ES:BX onto the stack.
6. Indirectly call CommandTable [command].
7. Pop ES:BX.
8. Set the DONE bit of the status that was returned.
9. Move the status code to the request packet.
10. Switch back to the system stack.
11. Restore the entry context.
12. Return to DOS.

The entry context must be saved by the interrupt routine because DOS does not perform this duty before transferring control to the interrupt entry point. All registers must be saved before calling the C subroutine. A stack switch is also necessary because DOS provides only a minimum stack. The C language function calls and local data declara-

tions tend to use up a fair amount of stack space.

An unfortunate side effect of placing all code and data in the same logical segment is that the address of TEXT segment items must be specified as being offset from DGROUP. This is not a problem in the assembly modules because a segment override can be specified. For instance, the strategy entry point address is specified in the device header as

```
DGROUP:strategy
```

and the call table elements are similarly defined as being offset from DGROUP. However, there is no corresponding construct available in the C language. This means that it is not possible to take the address of a function in this implementation of C, and that indirect function calls may not be made in the C code. Also, an assembly subroutine is necessary in order to set the ending address of the device driver in the initialization request packet.

The DOSDD.ASM module is almost entirely reusable as source code. Only minor changes are needed in the device header. For instance, the device name and attribute bits may change from one driver to another.

Refer to Figures 5a through 5e at the end of this article for a sample copy of DOSDD.ASM. Note the conditionally compiled code that is marked as UNIT\_TESTING. This code enables the device driver to be unit tested as a normal executable program by using a symbolic debugger.

**COMMANDS.C:** This file contains C functions that correspond to the DOS device driver commands.

```
link DOSD.OBJ COMMAND.OBJ IOCTL.OBJ RUNTIME.OBJ INIT.OBJ
```

Figure 2. Link statement



These are the entry points that are called indirectly by the interrupt routine through its call table of DOS command codes. Example function names are shown in Figure 3.

Each function can access its DOS request block through the far pointer that is passed to it from the interrupt routine. The following duties are performed.

1. Satisfy the DOS request.
2. Set any necessary values in the command-specific part of the request header.
3. Return the completion status code to the interrupt routine.

The status code is returned to the interrupt routine rather than being set directly for two reasons. First, it makes for a clean interface, with the C function satisfying a request and then returning the result of that request to its caller. Second, it is more efficient to set the status from the assembly module. Access through far pointers is somewhat expensive from C routines.

Refer to Figures 6a and b at the end of this article, for a skeleton sample of COMMANDS.C.

**IOCTL.C:** This module contains the entry point for the Ioctl commands. These commands and the Initialize command are the only DOS device driver commands that are not included in COMMANDS.C. A separate module is used for the Ioctl commands because they require further routing to a worker routine. Also, the Ioctls comprise a good functional unit that can be separated from the rest of the code.

```
MediaCheck ( MediaReqBlk far * RequestBlock )
Read ( ReadReqBlk far * RequestBlock )
DeviceOpen ( OpenReqBlk far * RequestBlock )
```

Figure 3. Example function names

Recall that the C programming model described herein does not allow indirect function calls to be made from the C code. As a result, Ioctl routing must be done through a SWITCH statement in this module. If this is inconvenient or not efficient enough, Ioctl call tables can be defined in an assembly module. In this case, the router must also be defined in assembly. The main command router would route the Ioctl request to a similar assembly language Ioctl router that verifies the subcommand code and makes an indirect call through an Ioctl call table.

Except for the additional level of indirection, the IOCTL module performs exactly the same duties as the COMMANDS module. That is, the IOCTL module:

1. Satisfies the DOS request.
2. Sets any necessary values in the command-specific part of the request header.
3. Returns the completion status code.

The extra function call means that the Ioctl routine must pass its argument along to the Ioctl worker routine. In turn, the worker routine must return the request status completion code to the Ioctl function, which passes it back to the interrupt routine. This may seem like a lot of shuffling of the status code; however, the C compiler handles func-

tion return values efficiently, so that very little penalty is incurred.

**RUNTIME.ASM:** This assembly module contains any assembly language routines that may be needed after initialization time. These are generally short routines that are callable from the C code. They are required in order to:

- Convert the C language register parameter-passing protocol into a stack-based, parameter-passing protocol.
- Provide code optimization where necessary.

A typical example of the first type would be a BIOS interface routine that would:

1. Load stack-based parameters into registers.
2. Make the software interrupt call to BIOS.
3. Return the BIOS return value to the calling C function.

The Ioctl router and Ioctl call table are good examples of the second major type of assembly code that may be needed.

Refer to your compiler reference manual for specific details on how to write assembly subroutines that are callable from C routines. Briefly, parameters are referenced indirectly through the BP register. Word size return values are placed in the AX register and double-word



declared as:	<code>extern int</code>	<code>data_object</code>
and defined as:	<code>_data_object</code>	<code>public _data_object dw</code>

Figure 4. Declaration and Definition of an Object

size return values are passed in the DX:AX register pair.

**INIT.C:** This module contains all of the C language initialization code for the device driver. The Initialize command is segregated in this module so that it can be discarded after initialization. Any auxiliary routines used only during initialization are also placed in this module and discarded after use. Some examples of INIT functions are:

- Display status and error messages.
- Find and register devices that are active.
- Initialize devices.
- Set the end address of the device driver.

As is the case with all of the DOS device driver commands, Initialize is called indirectly by the interrupt routine that passes it a far pointer to the request block. Initialize is required to return its status completion code to the interrupt routine upon exit.

Initialization code is discarded by reporting the address of the first C routine in INIT to the DOS kernel. This is accomplished by calling an assembly subroutine that is hard-coded with the address of the Initialize function, offset from the DGROUP as alluded to earlier. Any additional C language initialization code must be located after the Initialize function for it to be properly discarded. External data that is de-

clared in this module is not discardable. Data must be defined in the INIT segment in order for it to be discarded after use.

Discardable data that is defined in an assembly module is made visible to C routines by the PUBLIC declaration and by using the proper naming conventions. C compilers generally prefix an underscore to all external names. Therefore, a data object would be declared in the C file and defined in the assembly module as shown in Figure 4. This naming convention also applies to assembly subroutines that are to be called from C code, and to C subroutines that are called from assembly code.

Refer to Figure 7, at the end of this article, for a simple example of the INIT.C module.

**Structure Summary:** It should be apparent at this point that the device driver is actually an assembly program with C language subroutines. It must appear this way to the linker so that the overall structure of the device driver can conform to the DOS model. The normal C ordering of segments is reversed so that the device header will appear at the beginning of the device driver. The linkage ordering is further controlled in order to allow the initialization code to be discarded after use. All code and data are logically grouped into one segment as required by DOS.

Though the device driver is technically an assembly program, it appears to be a C language program to the programmer. Most of the code is written in C. The logic of the assembly code is mainly trivial, easy-to-understand bookkeeping. In addition, the assembly modules are reusable. The assembly code command router never changes. Unsupported commands can be handled in the COMMANDS module as empty C functions that return UNKNOWN COMMAND to the interrupt routine. Many of the RUNTIME module routines would be generally useful to any DOS device driver.

### Compile and Link Issues

The C startup routines as well as the C runtime can be eliminated by simply specifying that default libraries not be searched. With the IBM C/2 Compiler, this is accomplished by using compile and link control flags. The compile flag is /Z1, and the link flag is /NOD.

It should be apparent that the C startup routines have no place in a device driver. They establish the environment for an executable program, which a device driver is not. The reason for not using the C runtime may not be so obvious. The answer is that any library code that is used will be linked after the main body of the program. This means that it will be discarded along with the initialization code. Also, the library routines may contain stack probes or system calls that will fail in the device driver environment.

The other recommended compile flags for use with the IBM C/2 Compiler are:

```
-Gs -Zp -0x
```

The Gs option instructs the compiler not to generate stack probes



with the code. The stack probe code is designed to work with the stack that is created by the C startup routines. The device driver is linked without a stack segment, because DOS provides a stack for use at runtime. Under these conditions, the stack probes could not be expected to work properly.

The Zp option directs the compiler to pack data structures. That is, structures are created with contiguous members. If this option is not specified, the compiler may generate structures that contain holes in order to align data items on their boundary type. Device driver data structures must be packed because the DOS request packets are packed.

The Ox option causes the compiler to maximally optimize the code. A good optimizing compiler can produce surprisingly efficient code if given a free rein to do so. For example, with this option specified, the IBM C/2 Compiler will convert a data movement FOR loop statement in C to a REP MOVSW assembly instruction.

It is also suggested that the /ML option be used with the assembly modules and that the /NOI link option be used. Using these options will enforce case sensitivity of names within the assembly modules and across the link process. Case sensitivity must be preserved in order to ensure compatibility with the C language modules, because names in C are always case-sensitive.

### Performance Considerations

Because a DOS device driver is system level code, every effort should be made to ensure that the device driver code is efficient without sacri-

ficing clarity in the source code. The C programming language is very flexible, providing many different ways to accomplish the same task. This makes it incumbent on the programmer to make the best choices for the application at hand. Experienced C programmers know all the tricks; the following are some tips for beginners.

- Use register variables, especially for pointers and counters.
- Inspect the compiler assembly listing. This is useful for finding possible inefficiencies in the code. A knowledge of how the compiler handles different constructs makes it easier to write efficient C code.

### *The C programming language is a remarkable software development tool.*

- Do a performance analysis of the code and recode any bottlenecks in assembly if it will help. The original C code serves as pseudo-code and documentation.
- Pay particular attention to far pointer usage. Function calls cause the compiler to forget the extra segment, resulting in the re-loading of a segment register. The use of local temporary variables to store values until they can all be moved through the same far pointer is sometimes helpful.

These really are minor optimization points that will have little effect on the overall performance of the device driver. The algorithms that are used always play the major role in system performance. The use of a high-level language allows the programmer to concentrate on algorithm development because the compiler keeps track of so many of the low-level details of the code. In many cases, this means that code developed in the C language actually performs better than similar code developed directly in assembly language. The greater the size and complexity of the project, the more likely it is that this will be the case.

The C programming language is a remarkable software development tool. It combines the best features of a high-level language with the best features of a low-level language to produce an ideal general-purpose programming language. The high-level data and control constructs make the structure and function of the code visible, while relieving the programmer of the tedious details associated with assembly language programming. The close correlation with machine-level instructions allows the compiler to produce compact and efficient code. These qualities make the C programming language the perfect choice for DOS device driver development.

### ABOUT THE AUTHOR

*Dan Feriozi is a programmer in IBM's Entry Systems Division laboratory in Boca Raton, Florida. He is currently working on the development of device drivers for SCSI devices. He holds an M.S. degree in computer science from Florida Atlantic University.*



```

;*****
;* FUNCTION: To enable writing DOS device drivers in C
;*****
;
; .286c
; .seq
;
rh          struc          ; DOS request header
rh_length  db             ?
rh_unit    db             ?
rh_command db             ?
rh_status  dw             ?
rh_reserved1 dd          ?
rh_reserved2 dd          ?
rh          ends

rh0         struc          ; initialize request
rh0_rh      db             size rh dup (?)
rh0_numberOfUnits db      ?
rh0_endAddress dd         ?
rh0_bpb     dd             ?
rh0_blockDeviceNumber db  ?
rh0         ends

PRIVATE     macro          ; dummy macro to
                endm          ; label private

extrn      _UnknownCommand:near ; C functions to
extrn      _Initialize:near     ; handle DOS requests
extrn      _Input:near
extrn      _PeekInput:near
extrn      _InputStatus:near
extrn      _InputFlush:near
extrn      _Output:near
extrn      _OutputStatus:near
extrn      _OutputFlush:near
extrn      _DeviceOpen:near
extrn      _DeviceClose:near

DGROUP     group          _DATA, _BSS, CONST, _TEXT, INIT
_DATA      segment       word PUBLIC 'DATA'
                assume   ds:DGROUP
_DATA      ends

_BSS       segment       word PUBLIC 'BSS'
                assume   ds:DGROUP
_BSS       ends

CONST      segment       word PUBLIC 'CONST'
                assume   ds:DGROUP
CONST      ends

_TEXT      segment       word PUBLIC 'CODE'

```

Figure 5a. Sample DOSDD.ASM File



```

_TEXT          assume      cs:DGROUP, ds:DGROUP
               ends

INIT          segment      word PUBLIC 'INIT'
               assume      cs:DGROUP, ds:DGROUP
INIT          ends

;***** DATA SEGMENT *****

_DATA         segment

ifndef UNIT_TESTING
    entry_point:
endif

               PUBLIC     _Device_header, _Device_name

_Device_header label      word
               dd         -1                                ; next device
               dw         8000h                             ; attribute
               dw         DGROUP:strategy                  ; entry point
               dw         DGROUP:interrupt                 ; entry point
_Device_name   db         'DOSDD'                          ; device name
               dw         0                                ; reserved
               db         0                                ; drive letter
               db         1                                ; number of units supported

rh_offset     dw         ?
rh_segment    dw         ?
saveSP        dw         ?
saveSS        dw         ?
localstack    dw         128h dup ('s')
topofstack    dw         DGROUP:$

CommandTable  label      word
               dw         DGROUP:_Initialize
               dw         DGROUP:_UnknownCommand
               dw         DGROUP:_UnknownCommand
               dw         DGROUP:_UnknownCommand
               dw         DGROUP:_Input
               dw         DGROUP:_PeekInput
               dw         DGROUP:_InputStatus
               dw         DGROUP:_InputFlush
               dw         DGROUP:_Output
               dw         DGROUP:_Output
               dw         DGROUP:_OutputStatus
               dw         DGROUP:_OutputFlush
               dw         DGROUP:_UnknownCommand
               dw         DGROUP:_DeviceOpen
               dw         DGROUP:_DeviceClose
               dw         DGROUP:_UnknownCommand

_DATA         ends

;***** TEXT SEGMENT *****

_TEXT        segment

```

Figure 5b. Sample DOSDD.ASM File



```

;*****
;* FUNCTION: To save the pointer to the request header
;*****

strategy          PRIVATE
                  proc    far

                  mov     cs:rh_offset, bx      ; save the offset
                  mov     cs:rh_segment, es    ; save the segment
                  ret

strategy          endp

;*****
;* FUNCTION: Handle the interrupt call from DOS
;*****

ifdef UNIT_TESTING
    entry_point:
endif

interrupt         PRIVATE
                  proc    far
                  pushf
                  pusha
                  push    es
                  push    ds                    ; save entry context

                  mov     ax, cs                ; establish addressing
                  mov     ds, ax                ; for C and local data
                  mov     saveSP, sp           ; save entry stack
                  mov     saveSS, ss

                  cli                            ; stack switch
                  mov     sp, topofstack
                  mov     ss, ax                ; ax == cs
                  sti

                  mov     ax, rh_segment        ; restore es:bx
                  mov     es, ax                ; from strategy routine
                  mov     bx, rh_offset
                  mov     al, es:[bx].rh_command ; call request
                  cmp     al, 15                ; unknown command
                  jb     not_extended           ; in the table
                  mov     al, 15                ; unknown command
not_extended:     xor     ah, ah
                  shl     ax, 1

                  ; for word offset in table

                  mov     si, ax
                  push    es
                  push    bx
                  call    CommandTable[si]
                  pop     bx

```

Figure 5c. Sample DOSDD.ASM File



```

        pop     es
        mov     es:bx.rh_status, ax                ; set status

        cli
        mov     sp, saveSP
        mov     ss, saveSS
        sti

        pop     ds
        pop     es                                ; restore entry context
        popa
        popf
        ret
        even
        interrupt                                ; align for next part of seg
    _TEXT    ends

;***** INIT SEGMENT *****
INIT        segment                                ; init segment code is discarded

;*****
;* FUNCTION :  Display a String
;*****

_DisplayString    PUBLIC    _DisplayString
        proc    near
        push   bp
        mov    bp, sp
        push   di
        push   si

        mov    ah, 9
        mov    dx, [bp+4]
        int    21h                                ; display a string
                                                ; string$ pointer
                                                ; DOS functions interrupt

        pop    si
        pop    di
        pop    bp
        ret
_DisplayString    endp

;*****
;* FUNCTION:  Set the end of code address in the request header
;*****

_SetEndAddress    PUBLIC    _SetEndAddress
        proc    near
        push   bp
        mov    bp, sp
        push   di

```

Figure 5d. Sample DOSDD.ASM File



```

                push    si
                mov     di, [bp+4]                ; offset of rh
                mov     ax, [bp+6]                ; segment of rh
                mov     es, ax
                mov     word ptr es:di.rh0_endAddress, offsetGROUP:_Initialize
                mov     word ptr es:di.rh0_endAddress+2, cs

                pop     si
                pop     di
                pop     bp
                ret

_SetEndAddress  endp

;***** INIT DATA *****
                PUBLIC  _Init_msg1, _Init_msg2
                PUBLIC  _acrtused
_Init_msg1     db      ' was loaded', 0Dh, 0Ah, '$', 0
_Init_msg2     db      ' failed to load', 0Dh, 0Ah, '$', 0
_acrtused      dw      1                        ; for the MS compiler

INIT           ends

                end entry_point                ; entry point for debugging
                                                ; or for exe2bin (at 0)

```

Figure 5e. Sample DOSDD.ASM File



```

/*****
* FUNCTION: Handle requests to the device driver
*****/

#include "status.h"
#include "reqhdr.h"

PUBLIC
UnknownCommand (rh)    struct Rh    far * rh;           /* for unsupported commands */
{
    return (ERROR + UNKNOWN_COMMAND + DONE);
}

PUBLIC
Input (rh)    struct Rh4    far * rh;                 /* read from the device */
{
    return (DONE);
}

PUBLIC
PeekInput (rh)    struct Rh5    far * rh;             /* look at the next character */
{
    return (BUSY + DONE);
}

PUBLIC
InputStatus (rh)    struct Rh6    far * rh;           /* check the input status */
{
    return (DONE);
}

PUBLIC
InputFlush (rh)    struct Rh7    far * rh;           /* flush the input queue */
{
    return (DONE);
}

PUBLIC
Output (rh)    struct Rh8    far * rh;               /* write to the device */
{
    return (DONE);
}

PUBLIC
OutputStatus (rh)    struct Rh10    far * rh;         /* check the output status */
{
    return (DONE);
}

PUBLIC
OutputFlush (rh)    struct Rh11    far * rh;         /* flush the output queue */
{
    return (DONE);
}

```

Figure 6a. Skeleton COMMANDS.C File



```

PUBLIC
DeviceOpen (rh)    struct Rh13    far * rh;    /* prepare to use the device */
{
    return (DONE);
}

PUBLIC
DeviceClose (rh)  struct Rh14    far * rh;    /* finished using the device */
{
    return (DONE);
}

```

Figure 6b. Skeleton COMMANDS.C File

```

/*****
* FUNCTION: Initialize a DOS device driver
*****/

#include "status.h"
#include "reqhdr.h"

extern
char    Init_msg1[],    /* in the INIT DATA segment */
        Device_name[];

PUBLIC
Initialize (rh)    struct Rh0    far * rh;
{
    char    name [12];    /* local storage for the device name */
    register
    char    *p = name,
            *dev_name;

    uint n;

    dev_name = Device_name;    /* get the device name from the header */
    for (n = 8; n>0; --n){
        if (*dev_name == ' ')
            break;    /* stop at spaces */
        *p++ = *dev_name++;
    }
    *p = '$';    /* end of string marker */

    DisplayString (name);
    DisplayString (Init_msg1);    /* display the initialization message */
    SetEndAddress (rh);

    return (DONE);
}

```

Figure 7. Simple Example of the INIT.C File



## An Electronic Bulletin Board for PC Users

*Wyn Easton  
IBM Corporation  
Atlanta, Georgia*

**This article talks about the IBM National Support Center Bulletin Board System, how users can join, and what information can be found on it.**

Imagine being able to share information with thousands of computer users who have interests and concerns similar to yours. Also imagine being able to easily find product information, or having access to files and software that you can use on your own systems.

Is this reality? Yes, it is. This is what an electronic bulletin board is all about: fast, organized access to information that interests you. Bulletin boards are a medium for all users to obtain and share information.

IBM has an electronic bulletin board called the IBM National Support Center Bulletin Board System (BBS). This board is operational 24 hours a day, seven days a week. After free registration, PC users can call into the system; their only cost is toll charges.

### Accessing the BBS

To connect to the BBS, you will need a modem and communications

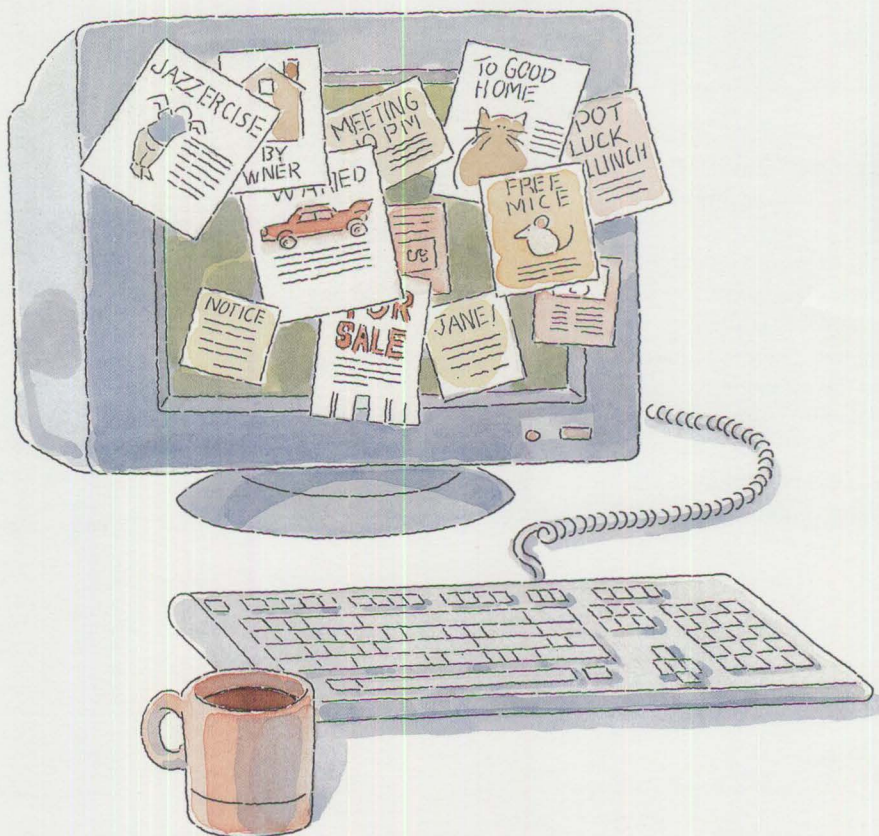
software. The BBS supports asynchronous modems from 300 to 9600 baud, and your software should be set to 8-N-1. Once ready, dial (404) 835-6600, and get connected. First-time callers can then register for the service by answering a few questions and assigning themselves a password. After that, callers have full access to the system for up to two hours per day.

### What's on the BBS?

The BBS has extensive databases of available IBM hardware and software. Users can search on a multitude of topics, ranging from

software available on OS/2 to Micro Channel adapters. IBM employees join the discussions in product conferences and answer users' questions.

One section of the bulletin board lists all the files and programs that can be downloaded on users' systems. Your communications software must have downloading capabilities in order for you to do this. In this section, there are entries such as OS/2 device drivers, DOS corrective service diskettes, and programming examples. Public domain "shareware" is also available. All of





Conference Number	Conference Topic
1	General Q&A
2	OS/2 General Usage Q&A
3	OS/2 Programming Q&A
4	OS/2 Application Specific Q&A
5	OS/2 on Non-IBM Hardware Q&A
11	DOS Q&A
12	DOS Programming
13	AIX Q&A
15	Networking Q&A
18	Personal System/2 Q&A
19	RISC Based Systems Q&A
20	IBM PC, XT, AT, PCjr Q&A
24	User Group Officer Messages
28	NSCPD Messages – National Support Center for Persons with Disabilities
30	Suggestions and Comments

Figure 1. Conference Selections

the programs come from IBM or from registered PC user groups.

In the bulletin section, you can obtain IBM's latest hardware and software announcements pertaining to personal systems. These announcements can be downloaded for reading at your leisure.

On your first call, you might want to explore some of the available con-

ferences to see if any spark your interest. A conference is an area of the bulletin board system where you can view, respond to, or leave messages. There are many conferences, each devoted to a specific topic. Figure 1 lists the conference selections.

To join a conference, type "J L" (for "join list") at the main command prompt, and press Enter. A menu of all the available confer-

ences will be listed. Then type a conference number, and press Enter. At this point, you can read messages, or perhaps leave a message with the system operator (SysOp). When ready to terminate the session, type "G" (for "goodbye").

Help in using the conference commands is available in the user guide (HELP.BBS), which can be downloaded. Online help is available by pressing "H" at the command line.

The BBS offers PS/2 users many benefits. Once connected, you will find a wealth of information that is well organized and easy to use. Give it a try!

#### ABOUT THE AUTHOR

*Wyn Easton is a market support representative for the IBM National Distribution Division National Support Center in Atlanta, Georgia. In 1973, after receiving a degree in mathematics and computer science from Central State University of Oklahoma, he joined IBM as a customer engineer in Tulsa, Oklahoma. His previous assignments include supporting software developers, programming, and systems engineering. Wyn's current responsibility is managing the IBM National Support Center Bulletin Board System.*



## Little Solutions



We invite you to share bits of information that may solve a problem or explain "why" to your fellow PC enthusiasts. Send us your **Little Solutions** in care of the editor.

### Long File Names for HPFS

There's a little confusion about the implementation and use of Long File Name support for OS/2 1.2 and HPFS in an application. Let's see if a few points can be clarified.

The only requirements for including the ability to use Long File Names in an application you are writing are to:

- Include the phrase LONGNAMES on the NAME line in your .DEF file
- Use the LINK.EXE that comes with OS/2 1.2

LINK.EXE files dated after the OS/2 1.2 version should be OK also.

Does C support Long File Names? The answer is "no," but C does not support printing either. C does not know (or care) about the concept of Long File Names. As for printing, a PRINTF(...) function was written to

allow printing. Long File Names can be included in an application just by an FOPEN("THIS IS A LONG FILE.NAME", ... and it's up to the operating system to deal with it from there.

Remember, Long File Names are not compatible with the FAT file system. If you want to create an application that will use the HPFS and Long File Names, remember we are still very committed to the FAT system and floppies. You will have to take into consideration the process to be used if a file with a Long File Name is to be copied to floppy, and perhaps later copied back. — *Larry Pollis, Dallas*

### OS/2 Print & System Error

If a "System Error" is displayed during the print screen operation, it's usually because the printer device driver is not associated with the appropriate printer queue. Check to ensure the correct device driver has

been selected during Print Queue Setup under the Printer Manager icon. The printer queues point to specific printer(s); incorrectly specifying the device driver can result in a "System Error" message. If you have selected more than one printer driver to be associated with a given printer, the last driver selected becomes the default driver. Deselecting the other printer drivers results in the desired single printer driver being selected for that printer.

— *Michelle Hillery, Dallas*

### Fast Mouse

Are you an OS/2 user who is using an IBM mouse? Are you frustrated with the sluggish performance of said rodent? Do you require a 400-square-inch mouse pad, where others can get away with 100 square inches? If you answered "yes" to these questions, try this:

Make sure the file MSPS202.SYS is in your \OS2 directory. Then, in your CONFIG.SYS file, locate these two lines:

```
DEVICE=C:\OS2\IBMMOU02.SYS
DEVICE=C:\OS2\MOUSE.SYS
        TYPE=IBMMOU$
```

Change these lines to read:

```
DEVICE=C:\OS2\MSPS202.SYS
DEVICE=C:\OS2\MOUSE.SYS
        TYPE=MSPS2$
```

Now reboot your system and observe the difference. Using the new driver should cut mouse movement in half.

It's always wise to save a copy of CONFIG.SYS before altering it. This way, if you make a mistake that renders your system unusable, you can boot up from the installation disk, copy the saved file into CONFIG.SYS, and try it again. — *Mark Chapman, Atlanta*



## OSO001.MSG Error

Are you getting the following error at boot time?

```
SYS0319: The system cannot
read message file OSO001.MSG
```

One cause for this is a bad (misspelled or missing) entry in your DPATH statement. If there is such an entry, the search for the message file stops at this point. Valid entries beyond this point are never looked at, and chances are that the path (C:\OS2\SYSTEM) for the message files are found there.

Another cause, recently discovered, deals with path entries that are pointing to HPFS drives. For example, in the DPATH statement

```
DPATH=C:\OS2\C:\MUGLIB\DLL;
C:\CMLIB;C:\OS2\INSTALL;
F:\TK12\IPFC;F:\TK12\DM;
F:\TK12\DTL;G:\EASEL-EE;
C:\OS2\SYSTEM
```

F: and G: are logical drives formatted for the HPFS. Because C:\OS2\SYSTEM follows the entries for the F: and G: drives, the system cannot find OSO001.MSG.

To fix this problem, remove all references to invalid or nonexistent paths from DPATH, and place all paths to the HPFS at the end of the statement. The system can now find the message files and operate properly. — Larry Pollis, Dallas

## Accessing Your OS2.INI File

As you may have found out, once OS/2 has control of your hardware, the only way to access the OS2.INI file is through Presentation Manager. External attempts to update or delete the file are prevented by OS/2, which informs you that the file is already in use by another process. I needed to control the menu

options and screen colors of 45 user workstations from a single point. Therefore, I had to devise a way to download a “generic” OS2.INI file from the file server to each user station. I did this by implementing the following three steps:

1. Place the OS2.INI file in a shared subdirectory on the file server. Call this directory \CONTROL\OS2\ . The file will be named OS2.INI. When users log on, they access this subdirectory as network drive S:.

2. When the user logs on, copy the shared OS2.INI file to a “staging” area on the user’s hard disk. Why to a staging area? Because copying the file to its final destination will have to be accomplished before Presentation Manager has started, but before the user has access to shared resources on the network. This is accomplished by creating a batch file that first logs the user on, and once logged on, successfully copies the new .INI file to the staging area.

- Log the user on to the network.

```
NET LOGON PPOGGI
```

- Execute file UPDATE.CMD to copy the new .INI file to the user’s local hard disk (Figure 1).

3. Move the “staged” OS2.INI file to its final destination. How do you do this if the file will be locked by PM? Do it by performing the copy prior to PM starting. I did it by placing a RUN command as the first statement in the user’s CONFIG.SYS file (Figure 2).

```
COPY S:\CONTROL\OS2\OS2.INI D:\CONTROL
```

Figure 1. Copy Command

```
RUN=C:\OS2\XCOPY.EXE D:\CONTROL\OS2.INI C:\OS2\OS2.INI
```

Figure 2. RUN Statement

Once the system has completed the boot process and OS/2 begins executing, Presentation Manager uses the new OS2.INI file, which is placed there by the XCOPY command from CONFIG.SYS.

You can use this method to save hours by distributing changes to user menus from the file server, rather than implementing the changes one at a time on each user station. You could use this to:

- Change menu options
- Establish screen color standards
- Control just about anything that is maintained in the OS2.INI file, and set by the Presentation Manager Control Panel. — Peter Poggi, Rockville, Maryland

*Editor’s note: The purpose of the .INI file is circumvented by this process. The .INI file is used to store the positions and sizes of windows in some applications, and data and information to restore features and functions in others. If you replace the .INI file at startup, data set by the person using the system will be lost. In a classroom or training seminar this may be ideal. In other instances it may not be.*

*The methods described here may be helpful to keep other files current. Data files that an application may use or updates of executable code can be transferred in the same manner. Just “request” the file at startup before you start the application.*



## New Products

### Hardware

#### Token Ring Network 16/4 Trace and Performance Adapters

The Token Ring Network 16/4 Trace and Performance Adapter and Token Ring Network 16/4 Trace and Performance Adapter/A are feature cards that can be installed in IBM PCs and PS/2 products. These feature cards work with the Token Ring Network 16/4 Trace and Performance Program to provide a real-time view of the traffic on token ring networks operating at 4 or 16 megabits per second (Mbps). This combination of hardware and software analyzes trace data, measures usage, and collects ring station traffic statistics. Using this data, companies can get valuable information for debugging application software, capacity planning, and network reconfiguration.

#### Highlights:

- System management of token ring networks is enhanced with the visibility into the network provided by the combination of 16/4 Trace and Performance Adapters and 16/4 Trace and Performance Program
- The 16/4 Trace and Performance Adapters and 16/4 Trace and Performance Program provide utilization data that will enable companies to decide when to increase the ring speed from 4 to 16 Mbps
- Token ring network administrator productivity is increased with an identical, easy-to-use, menu-driven interface to debug and troubleshoot ring segments running at 4 or 16 Mbps
- Investments are protected by allowing a machine to be used as a standard workstation when not used for trace or performance tasks

Letter #190-090, June 5, 1990

### Software

#### IBM Token-Ring Network 16/4 Trace and Performance Program

The IBM Token-Ring Network 16/4 Trace and Performance Program works in conjunction with the IBM Token-Ring Network 16/4 Trace and Performance adapters. This program provides a realtime view of the traffic on the token ring networks operating at 4 or 16 megabits per second (Mbps). This combination of hardware and software analyzes trace data, measures media usage, and collects ring station traffic statistics. Using this data, companies can get valuable information for debugging application software, capacity planning, and network reconfigurations.

Letter #290-284, June 5, 1990

#### IBM Token-Ring Network Bridge Program Version 2.2

IBM Token-Ring Network Bridge Program Version 2.2 connects two local or remote IBM Token-Ring Networks into one logical ring. The connected rings can be any combination of 4 or 16 megabit per second (Mbps) IBM Token-Ring Networks. IBM Token-Ring Network Bridge Program Version 2.2 offers increased frame sizes from 2,052 bytes to 4,472 bytes in the remote configuration and feature dial support.

Dial support allows the remote IBM Token-Ring Network Bridge halves to initiate and receive calls by the command of a user workstation. The connection between the bridge halves would be made only when there was data to be transmitted. Once the data transmission is complete, the connection would be dropped. Dial support is provided through a V.25 bis (functionally equivalent) modem such as the IBM 7855-10 V.32 Modem connecting the bridge to the switched teleprocessing (TP) line. Line speeds for dial support are modem-dependent; the typical line speed is 9.6 kilobits per second (Kbps).

IBM Token-Ring Network Bridge Program Version 2.2, like IBM Token-Ring Network Bridge Program Version 2.1, also features remote bridge connectivity via a leased TP line at speeds up to 1.344 Mbps. IBM Token-Ring Network Bridge Program Version 2.2 also supports the IBM 7820 Integrated Services Digital Network (ISDN) Terminal Adapter attachment.

Letter #290-303, June 19, 1990

#### IBM Local Area Network Support Program Version 1.2

The IBM Local Area Network Support Program has been extended to support Ethernet™ DIX Version 2 and IEEE 802.3 LANs in addition to token ring and PC Networks.

#### Highlights:

- Additional business solutions are provided by allowing DOS applications using IBM LAN Support Program Version 1.2 to run on 802.3 and Ethernet LANs
- Growth is enabled by allowing 802.3 and Ethernet users to expand their networks by connecting to IBM Token Ring LANs via an IBM 8209 LAN Bridge
- Investment protection is achieved by supporting existing 802.3 and Ethernet users

Letter #290-305, June 19, 1990

#### IBM LinkWay™ Version 2.0

IBM LinkWay is an easy-to-use, low-cost application development tool that enables users to create, present, and modify applications containing text, pictures, video images, and sound without requiring a background in programming or other computer skills. This version of IBM LinkWay contains major improvements, which include VGA graphics, enhanced paint program, improved text editors, and a full-function script language.



**Highlights:**

- High-function script language with subroutines, bit maps, and button controls
- Many usability aids such as contextual help, text reflow, and on-line tutorials
- Enhanced paint program with scaling, cut-and-paste, test smoothing, and pixel editing
- Reference buttons to provide scrolling "hypertext"
- VGA mode (640 x 480 by 16 colors)
- Grey-scale printing of MCGA 256-color pictures
- User selection and creation of palettes
- Performance improvements in picture loading and page-to-page transitions

Letter #290-385, July 3, 1990

## **BookManager™ Solutions for Online Documentation**

BookManager software programs provide an on-line solution for managing large amounts and complex documentation used in the workplace. These new and enhanced BookManager programs are for VM, OS/2, and DOS environments. The BookManager solution will be available as a Systems Application Architecture (SAA) application in a number of key IBM systems environments. The OS/2 program is the first SAA program in the BookManager family.

The BookManager solution transforms information ordinarily used to produce printed documents into on-line information, also known as softcopy. In this form, the information is electronically distributed and available on line to give end users the ability to find, browse, and use the information at their workstations. Organizations can ensure higher quality, current information, increased information and worker productivity, and paper-handling delays and expense are reduced.

On-line information created with BookManager is usable without modification across different operating systems. This capability and the support for key platforms makes it possible for companies to implement softcopy applications to be used throughout the enterprise. On-line information can be used across enterprises, extending benefits to customers, vendors, and business partners.

IBM is using BookManager to make on-line versions of IBM product applications. IBM software products can include softcopy documentation in the form of on-line files directly viewable with BookManager READ programs, source files, or both.

Copies of the softcopy documentation included in IBM products may be used in as many systems as needed within an enterprise. Electronic libraries containing IBM product manuals will be available so end users can access accurate and consistent IBM product information whenever needed. After receiving permission from IBM, companies can modify source files or create customized printed or softcopy manuals using BookManager and other IBM publishing products.

**Highlights:**

- Helps organizations manage the "information explosion" by enabling computerized on-line delivery of information that traditionally was provided only in hardcopy
- Enables enterprise and cross-enterprise on-line documentation application by providing a common on-line information format and BookManager programs to provide softcopy functions across platforms
- Enables IBM to ship softcopy versions of IBM product manuals to and meet company requirements for electronic distribution and storage
- Allows companies using IBM Document Composition Facility (DCF) and Master Series products to move from hardcopy to softcopy to publishing using existing source files and without significant special effort

- Helps end users find information faster and handle larger libraries
- Improves end-user productivity by enabling on-line annotation as well as copying and printer-retrieved information
- Improves support for non-U.S. language documentation, providing a solution for multinational and non-U.S. firms

Letter #290-374, July 3, 1990

## **BookManager READ/DOS 1.2**

IBM BookManager Read/DOS 1.2 enables users to read softcopy information, and use the hypertext facility to search, take notes, and print that information. It gives businesses a solution for handling information by easily and quickly accessing information at reduced cost. End users' productivity can be improved by finding information quickly and easily without leaving the workstation. BookManager READ/DOS 1.2 increases businesses' ability to manage information vital to their growth, by making electronically stored information useful. System management is improved through the use of BookManager READ/DOS 1.2 by accessing IBM softcopy manuals and publications at the workstation.

BookManager READ/DOS 1.2 is a member of the BookManager family of on-line documentation solutions.

**Highlights:**

- Softcopy books enhance users' ability to manage information. BookManager READ/DOS 1.2 expands the ability to access business information at the workstation
- Productivity gains are realized by professional, technical, and office end users, by locating and accessing information using sophisticated search-access books and linking capabilities within those books
- Business growth is enabled through managing increased amounts of



information in softcopy publications. Softcopy documents stored on a LAN server, an AS/400 folder, fixed disks, diskette, or CD-ROM, allow quick access to information as it is needed

- Information is a key asset to any business. A company's investment is lost if information in the "stacks" cannot be found and used. BookManager protects the company's investment by making all information accessible and usable upon demand
- System management is improved by using BookManager READ/DOS 1.2 to access IBM softcopy publications, or with the company's own manuals built with BookManager BUILD/VM Release 2 or Release 1
- The needs of many companies are supported by BookManager's ability to use softcopy books that are written in many languages

Letter #290-375, July 3, 1990

### **BookManager READ/2 1.2**

Softcopy documentation can help companies manage the ever-increasing volumes of information needed to operate a business and reduce the expenses of publishing in hard copy. Making information available on softcopy can increase worker productivity by giving end users the ability to find, read, and use the information they need right at their OS/2 workstations.

IBM BookManager READ/2 1.2 enables users to read softcopy information, perform searches using the hypertext facility, take notes, and print that information. It offers businesses a solution for handling information by providing a method to easily and quickly access information at reduced cost. End users' productivity can be improved with its ability to find information quickly and easily without leaving the workstation. BookManager READ/2 1.2 increases businesses' ability to manage information that is vital to their growth. The company's investment is therefore protected by making electronically stored information available at a workstation.

By using BookManager READ/21.2, system management is improved by its ability to access IBM softcopy manuals and publications at the workstation.

BookManager READ/2 1.2 is a member of the BookManager family of on-line documentation solutions.

#### **Highlights:**

Highlights are the same as those listed for BookManagerREAD/DOS.

Letter #290-376, July 3, 1990

### **IBM Classroom LAN Administration System Version 1.30**

IBM Classroom LAN Administration System Version 1.30 is an educational product that combines software and documentation to help manage courseware and other applications in a LAN environment. This offering is available only to qualifying educational institutions with an IBM Educational Allowance Amendment in effect. Version 1.30 simplifies network installation, which enables teachers to set a bookmark for the Basic Skills applications for the entire class or for some students, individually.

Version 1.30 includes a teacher's menu that can be tailored for easy application access. It has a teacher/student function that allows teachers to distribute files to their students and copy files from the students' home directory as well containing send and receive files. Version 1.30 provides office/administrative personnel access to the system through the addition of a new group called "Office." IBM Classroom LAN Administration System Version 1.30 is appropriate in school environments requiring a network.

#### **Highlights:**

- Teachers can create their own menu to run applications from outside a class, using two directories from which applications (courseware and noncourseware) can be selected
- Individual menus for teachers and office users can be tailored by selecting

programs from a list of available applications

- Version 1.30 maintains support of the installation tape
- A SUPERADMIN ID has been added that belongs to the Office group, but has more rights and privileges than the Office group
- SUPERADMIN has menus that enable the loading and running of non-courseware applications on the network
- The print station function has been enhanced to show the print queues that are already in use
- Teachers may set bookmarks for those applications that support bookmarks, such as the Basic Skills applications
- Installation of noncourseware applications on the system is now available for teachers and administrative personnel
- Installation of courseware from sub-directories is included to facilitate installing applications from a fixed disk or from diskettes that have more than one application

Letter #290-386, July 3, 1990

### **IBM Education LAN and Tools**

IBM Education LAN and Tools is an education network package that consists of network software, productivity tools, and related publications for each item. This offering is available only to qualifying educational institutions with an IBM Educational Allowance Amendment in effect. The network software includes IBM Classroom LAN Administration Version 1.30 and Novell Advanced Netware™ Version 2.15. The productivity tools are comprised of IBM Link-Way Version 2.00, Microsoft Works® Version 2.0A, LANSchool™ Version 3.0, Excelsior® GRADE Version 1.6, Excelsior QUIZ Version 2.5, and Express Publisher™ Version 1.1.



IBM Classroom LAN Administration System Version 1.30 is an education network product that combines software and documentation to help manage courseware and other applications in a LAN environment. It uses the Novell NetWare LAN operating system and IBM DOS.

The productivity tools found in Microsoft Works consist of a word processor with spell checking, spreadsheet with graphics, data base with reporting capabilities, and communication. IBM LinkWay Version 2.00 is an easy-to-use, teacher/student productivity tool that enables users to organize, store, and retrieve text, pictures, graphic (video) images, and sound (audio) without requiring an extensive programming or computer skills background.

LANSchool is an instructional tool that allows teachers to "broadcast" computer screens to student workstations connected to the network and project an on-screen pointer for easy instruction.

Excelsior GRADE is a teacher's grade book offering a student data base, grade data management, test scoring, and report generation. Excelsior QUIZ is a design and compiler application for customizing tests from a teacher's data base of questions to develop student tests and quizzes. QUIZ can provide a direct link with the Microsoft Works word processor for creating test items. Express Publisher is a page design program that allows for graphics and text to be combined to produce newsletters, reports, flyers, or presentation graphics. It can import pictures and text from other word processors.

Letter #290-383, July 3, 1990

### **IBM PROLOG for OS/2**

IBM PROLOG is for the OS/2 platform. A key feature of the programming language is its environment for developing, debugging, and compiling Prolog applications.

IBM PROLOG for OS/2 runs under OS/2 Presentation Manager (PM) and lets Prolog programs be created, debugged, interpreted, or compiled and executed. It provides access to Database Manager, Dialog Manager, and IBM PROLOG's on-line help information.

IBM PROLOG Runtime for OS/2 is required by applications created with IBM PROLOG for OS/2 for deploying other PS/2 devices.

#### **Highlights:**

- IBM PROLOG is designed to provide versatility in addressing business problems, particularly in the areas of complex data structures and relational databases
- Rapid prototyping and deployment of applications using IBM PROLOG are key to user productivity gains
- IBM PROLOG applications can grow with the businesses' need to grow
- Portability to multiple platforms, extensive system integration, and multiple syntax support minimize the impact to existing solutions and skills
- IBM PROLOG has been structured to fit well into today's data processing environments

Letter #290-401, July 24, 1990

## **Statement of Direction**

### **IBM Writing to Write™ Statement of Direction**

IBM Educational Systems intends to announce the Writing to Write elementary school writing series that will be designed to enable elementary school children to learn to write what they think. Writing to Write will be a balanced

curriculum where the teacher and the courseware are equal instructional partners. The program design will be instructional, so students actively participate and learn by doing, as opposed to "drill and practice." Writing to Write will encompass the stages of the writing process approach: prewriting, drafting, editing, revising, publishing, and sharing. The spiral curriculum, the combination and coordination of computer exercises, the ancillary print activities, and the teacher instruction will make Writing to Write a courseware product that takes full advantage of technology.

Writing to Write elementary school writing series will make extensive use of PS/2 graphics and voice capability. Writing to Write will serve as a follow-on to Writing to Read® and complement IBM's Teaching and Learning with Computers methodology. Writing to Write will employ a spiral curriculum with problem-solving techniques, and will be designed as the primary source of instructional sequence and methodology for teaching writing in elementary schools.

Writing to Write elementary school writing series will be made up of networked products only, and will run under the IBM Classroom LAN Administration System Version 1.11 or later.

Field testing of the new series has begun at selected schools. IBM is announcing this Statement of Direction at this time to allay uncontrolled speculation that might otherwise result as our field testing becomes more apparent to the general public. IBM's current intentions and plans are subject to review, and announcement of products in this new series will be based on IBM's technical and business judgment.

Letter #290-249, May 29, 1990



**Index to Past Issues of  
IBM Personal Systems Technical Solutions**

**Issue 3, 1990 (G325-5007)**

DOS – A Look under the Hood to See How It Spins  
Memory Management in a DOS Environment  
FASTOPEN – The DOS Performance Enhancer  
DOS 4.00 Compatibility Issues  
'Out of Environment Space' Errors  
A New LAN Requester for DOS Systems  
Creating a Dialog Box Dynamically Using WinCreateDlg  
An Alternative for the OS/2 START Command  
CUA: A Consistent Interface

**Issue 2, 1990 (G325-5006)**

OS/2 End User Advantages  
What's New in OS/2 Standard Edition Version 1.2?  
An Application Developer's View of OS/2  
Object-Oriented Programming with C and OS/2 PM – Is It Possible?  
Design Goals and Implementation of the New High Performance File System  
OS/2 EE 1.2 Database Manager – Remote Data Services  
OS/2 EE Database Manager Precompiler API  
UNION, INTERSECT, EXCEPT  
Writing a Database Manager COBOL/2 Program  
Database Manager Programming with Procedures Language 2/REXX  
APPC Performance Tips for OS/2 EE  
EASEL OS/2 EE PROFS: Host Code Interface  
PS/2 RPG II Application Platform and Toolkit  
The IBM Independence Series Products

**Issue 1, 1990 (G325-5005)**

Introduction to Local Area Networks  
IEEE 802.3 LAN Considerations  
The IBM Token-Ring Network: A New Generation  
How to Design and Build a 4-Mbit/s Token-Ring LAN  
How to Design and Build a 16-Mbit/s Token-Ring LAN  
Making the Cabling Decision  
IBM Cabling System Highlights  
Communications Strategy for Growth

**Issue 4, 1989 (G325-5004)**

Requirements for Advanced Bus Architecture  
Micro Channel System Configuration Considerations  
Features and Benefits  
Bus Masters and Applications  
Overview of Extended Micro Channel Functions  
New Micro Channel Features  
SCB – An Architecture for Micro Channel Bus Masters  
Design Alternatives with Micro Channel Systems  
The IBM PS/2 Micro Channel SCSI Adapters  
PS/2 Wizard Adapter  
Experience in Bus Master Design  
Bus Master Adapters from Independent Option Vendors  
Bus Masters and OS/2  
Micro Channel Issues in AIX PS/2  
Information for Developers  
Book Report: "The Winn Rosch Hardware Bible"



“The first thing you notice about the PS/1 is its compact size. (page 2)

“If a driver for the 4019 PostScript option is not available, consider the drivers for the Personal Page Printer Models 030 and 031. (page 17)

“The 82325 chip set supports the 32-bit Streaming Data Procedure. (page 20)

“A well-written LAN driver can make 100 percent difference in testing results. (page 26)

“Multithreading – the ability to run several parts of an application concurrently – allows more intelligent applications to be developed. (page 34)

“After deleting all unnecessary files, OS/2 EE requires approximately 25 MB of DASD. (page 44)

“Each logged-on user ID has a unique process and session ID identifying it. (page 48)

“SMB Raw Protocol enables extremely fast data transfer across the network. (page 56)

“Object-oriented software design has its roots back in the late 1960s. (page 61)

“This is what an electronic bulletin board is all about. (page 91)

6325-5009-00

