Chapter

5 Windows 3.1 and Memory Management

This chapter contains information about how Microsoft Windows 3.1 interacts with memory. You can use this information to manage memory while running Windows and to troubleshoot various problems related to memory management. For specific information about optimizing your system configuration, see Chapter 6, "Tips for Configuring Windows 3.1."

Related information

- *Windows User's Guide:* Chapter 5, "Control Panel," and Chapter 14, "Optimizing Windows"
- *Windows Resource Kit:* Chapter 7, "Setting Up Non-Windows Applications," and Chapter 13, "Troubleshooting Windows 3.1"
- *Glossary terms:* Expanded Memory Specification, Extended Memory Specification, multitasking, page frame, protected mode, virtual memory

Contents of this chapter

About Memory

Your computer's random access memory (RAM) is a volatile medium where applications and data are stored while you are working with them. When you finish working, the information is transferred back to permanent storage on the hard disk or floppy disks.

Windows applications such as Microsoft Word and Microsoft Excel have to be loaded into memory before they can run. Generally, when you run Windows, the more memory your system has available, the more applications you can run at the same time, and the faster the applications will run.

This section describes the types of memory in a PC and provides a technical overview of expanded memory. Other sections in this chapter present issues specific to using memory in Windows standard mode and 386 enhanced mode.

Note This chapter supplements, rather than repeating, the information in Chapter 14, "Optimizing Windows," in the *Windows User's Guide.*

Types of Memory: An Overview

Your computer system can have three different kinds of memory: conventional memory, extended memory, and expanded memory. Windows also creates a fourth type of memory, virtual memory, which is discussed in "Virtual Memory and 386 Enhanced Mode" later in this chapter.

- **Conventional memory** consists of the first 640K of memory available on your machine. Most PCs have at least 256K of conventional memory. Your system must have 640K of conventional memory to run Windows.

  When you boot your machine, MS-DOS runs the utilities and applications listed in the CONFIG.SYS and AUTOEXEC.BAT files. These files often use conventional memory to function. The remaining memory is available for running other applications such as Windows.

- **Extended memory** is essentially a seamless upward extension of the original one megabyte address space available in the memory of 80286 and 80386 machines. Extended memory always starts exactly at 1024K, where the upper memory area ends. The first 64K of extended memory is referred to as the high memory area (HMA).
- **Expanded memory** can be installed as an expanded memory card or, on an 80386 machine, emulated by an expanded memory manager (EMM). The EMM software maps pages of expanded memory onto the system's upper memory area (from 640K to 1024K). Applications must be designed to interact with EMM software to take advantage of expanded memory.

Expanded memory is slower and more cumbersome to use than extended memory, because the expanded memory manager gives applications access to only a limited amount of expanded memory at a time.

An 80286 processor can address 16 megabytes of total memory, and an 80386 processor can address 4 gigabytes of memory. The 8086 and 8088 machines have hardware limitations that exclude use of extended memory. For such machines, expanded memory is the only option for extra memory.

Working with conventional and extended memory under Windows 3.1 is a straightforward process that seldom demands your attention, because Windows handles it automatically through HIMEM.SYS. If you want to run non-Windows applications that require expanded memory, you need a deeper understanding of how it is structured and how to access it. For more details, see "Expanded Memory: A Technical Discussion" later in this chapter.

Figure 5.1 shows the relative addresses of conventional memory, the upper memory area (used by expanded memory), and extended memory.

Figure 5.1

Extended memory goes from 1024K to 16 MB for 80286 PCs, or to 4 GB for 80386 PCs

Expanded memory uses page frames in the upper memory area

Conventional memory goes from 0K to 640K

To see the kind and amount of memory in your system:

- Type **msd** at the command prompt to run the Microsoft Diagnostics utility that is installed with Windows 3.1. This utility prepares an extensive report on the memory and drivers installed in your system.
- Or, if MS-DOS 5.0 is installed on your system, type **mem** at the command prompt to see a brief summary of your system's memory.

You can also use parameters with the MS-DOS 5.0 **mem** command. The **mem /c** command reports free memory for both conventional and upper memory addresses, and lists the decimal and hexadecimal sizes of all programs loaded into conventional and upper memory. The **mem /p** command presents a detailed listing of memory use, including load order, size, and address information for programs (listed separately from the environment). For more information, see your MS-DOS 5.0 documentation.

Note If you use the **mem** command to view memory from within a Windows 3.1 virtual machine, you will only see the amount of memory available to the virtual machine.

The Windows 3.1 Memory Device Drivers

Both Windows 3.1 and MS-DOS 5.0 provide these device drivers for managing memory:

- HIMEM.SYS provides access to extended memory and the HMA. The extended memory it provides conforms to XMS 3.0.
- EMM386.EXE takes the XMS 3.0 memory provided by HIMEM.SYS and uses it to emulate expanded memory or to provide UMBs, or both. EMM386.EXE is DPMI-compliant and emulates expanded memory that conforms to LIM 3.2 or LIM 4.0.
- RAMDRIVE.SYS creates a RAM disk.

- SMARTDRV.EXE is a disk caching utility.

In addition, Windows 3.1 has two built-in memory managers:

- Swapdisk, which manages the disk swap process for caching Windows application code in both standard and 386 enhanced modes.
- Virtual Memory Manager (VMM), which manages swapping to a temporary or permanent swap file in 386 enhanced mode.

Expanded Memory: A Technical Discussion

Note This discussion is relevant if you want to take advantage of expanded memory installed in your system, or if you are running Windows 3.1 with non-Windows applications that require expanded memory. If your system doesn't have expanded memory, or if you are not running applications designed to use expanded memory, you can skip this discussion.

The original IBM PC design, based on the Intel 8086/8088 CPU, restricts usable memory to about 640K. A Lotus/Intel/Microsoft (LIM) collaboration developed a technique for adding memory to PC systems. The LIM Expanded Memory Specification (EMS) bypasses the memory limits by supporting memory cards that contain 16K pages (or banks) of RAM that must be enabled or disabled by software, but that cannot normally be addressed by the CPU. Instead, each page is mapped into the address space of the processor.

Applications can use expanded memory to get around the memory limitations of 8086/8088 processors by using a special area of the machine's memory called the upper memory area. This upper memory area is always located in the same area of the computer's address space: from 640K to 1024K (A000 to FFFF hexadecimal). The upper memory area (sometimes referred to as the "adapter segment" because that part of memory is used by hardware adapters such as display adapters) is also where the ROM BIOS read-only memory segment is located. Figure 5.2 shows the upper memory area, its address range, and the items that occupy fixed portions of it.

An application must be specifically written to take advantage of

expanded memory. Many non-Windows applications use expanded memory:

- To gain more effective performance from large non-Windows applications such as spreadsheets and CAD programs.
- To run memory-resident programs or applications that use shared data.

This section describes the LIM expanded memory specifications and discusses two operations for using expanded memory: bank switching and backfilling.

Note The 80386 and higher microprocessors can emulate EMS hardware by using extended memory with memory managers and special software such as EMM386.EXE. For more information about using this device driver with Windows 3.1, see Chapter 6, "Tips for Configuring Windows 3.1."

The Expanded Memory Specifications

The two kinds of expanded memory are differentiated from one another by their LIM expanded memory specification version numbers:

- **LIM 3.2** This expanded memory specification moves data in 64K blocks, each made up of four contiguous 16K pages to form a 64K page frame. The LIM 3.2 standard works well for storing data from spreadsheets in expanded memory, but it is insufficient for multitasking.
- **LIM 4.0** This expanded memory specification allows data to be moved in blocks of 1 to 64 pages (overcoming the LIM 3.2 limitations on size and flexibility). LIM 4.0 also removes the restriction that the pages must be contiguous, essentially abandoning the page frame requirement.

Figure 5.2

The upper memory area

(640K to 1024K)

General HEX Address	Specific HEX Address	Decimal Address	
	FFFF	1024	1024K
FC00	FC00	1008	
	FBFF	1007	ROM BIOS
F800	F800	992	
	F7FF	991	(Basic
F400	F400	976	Input/Output
	F3FF	975	System)
F000	F000	960	
	EFFF	959	
EC00	EC00	944	
	EBFF	943	Not available
E800	E800	928	on PS/2s
	E7FF	927	and some
E400	E400	912	other machines

E000	E3FF E000	911 896		
DC00	DFFF DC00	895 880		
D800	DBFF D800	869 864		
D400	D7FF D400	863 848		
D000	D3FF D000	847 832		
CC00	CFFF CC00	831 816		
C800	CBFF C800	815 800		
	C7FF	799	8514/A	

C400	C400	784		Non-PS/2	
	C3FF	783		VGA	
C000	C000	768			EGA
BFFF	BFFF	767	EGA/		
BC00	BC00	752	VGA	Hercules	
BBFF	BBFF	751	Text/	Page 2	CGA
B800	B800	736	Low Res		
B7FF	B7FF	735			
B400	B400	720		Hercules	
B3FF	B3FF	719	MDA	Page 1	
B000	B000	704			
AFFF	AFFF	703			
AC00	AC00	688			
ABFF	ABFF	687		EGA / VGA	
A800	A800	672		High Resolution	

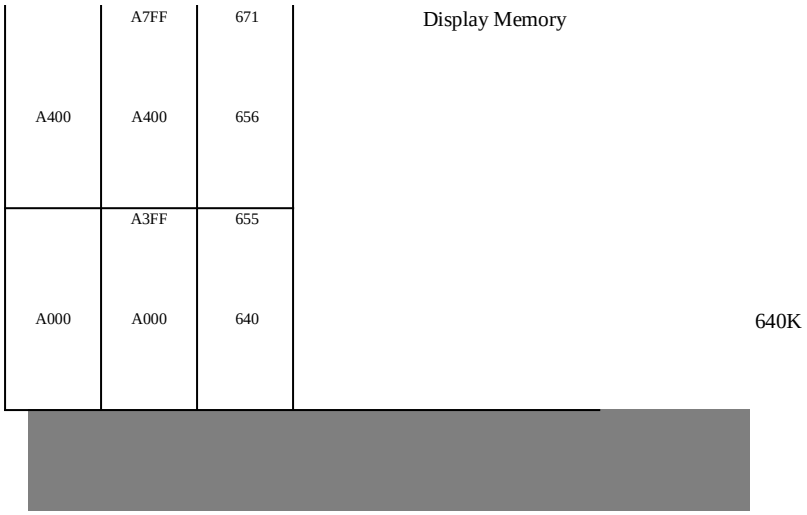
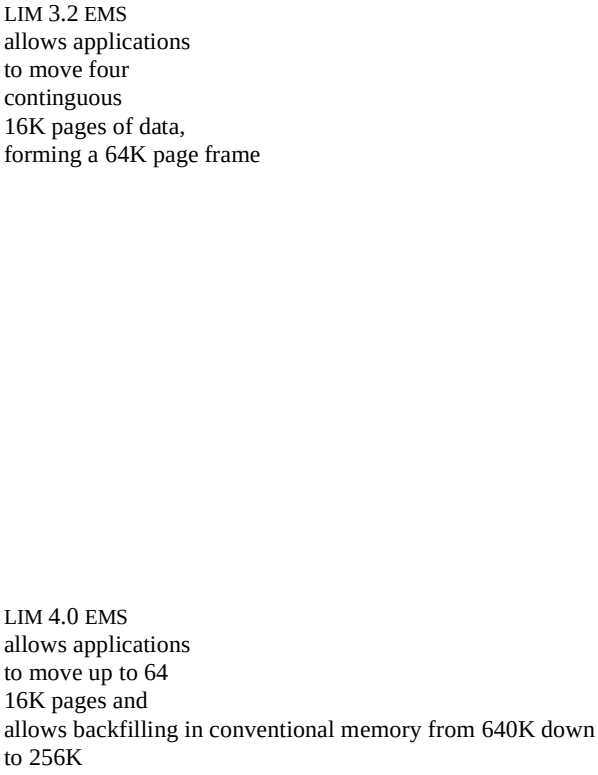

A7FF
671
Display Memory
A400
A400
656
A3FF
655
A000
A000
640
640K

Figure 5.3 shows some key differences between the LIM 3.2 and LIM 4.0 specifications. If a non-Windows application requires expanded memory, usually its documentation will describe the LIM version under which it works.

Figure 5.3



Bank Switching in Expanded Memory

Application programs take advantage of expanded memory by making calls to the expanded memory manager (EMM) to request blocks of expanded memory. Bank switching is the process of mapping or temporarily assigning memory from a pool of expanded memory to an empty address space in the upper memory area.

For example, if an application needs more memory for data, it contacts the EMM, which allocates expanded memory 16K at a time. The application writes up to 64K of data to the address allocated by the EMM, then requests another 64K allocation. This request is the bank switch request. The EMM allocates another 64K of memory at a different address while tracking where the first 64K of data was placed.

The EMM continues this bank switching activity, 16K at a time, until the application makes no more request for memory space. With bank switching, the EMM can manipulate several megabytes of data through the single 64K space.

Backfilling in Expanded Memory

With LIM 4.0, the expanded memory manager can backfill conventional memory addresses from 640K down to approximately 256K, using memory that is usually reserved for MS-DOS, utilities, and TSRs. With backfilling, the system can manipulate large amounts of data, which is particularly important for multitasking. A rotating pool of backfilled memory allows entire applications and considerable data to be loaded into memory at once, greatly speeding operations.

One trick to maximize the benefit of backfilling on 80286 machines is to disable as much of the motherboard memory as possible (down to the 256K level) and let the expanded memory card supply that memory. Because the expanded memory card is supplying the memory, it can bank switch freely and make the best use of available memory.

For 80386 machines, you can convert extended memory into

expanded memory by using an expanded memory manager such as EMM386.EXE. If you are using an expanded memory card on an 80386 machine, read your manual carefully before trying to backfill. Not all memory cards have the register support to supply more than four 16K pages. Also, because Windows 3.1 supports extended memory directly, backfilling expanded memory will not give you any advantage for running Windows.

Difficulties with Expanded Memory

Expanded memory uses the upper memory area, which it shares with hardware adapters such as SuperVGA video cards, network cards, 3270 emulation cards, and ESDI disk controllers. Several potential difficulties can arise when expanded memory contends for address space with the adapters:

- **Lack of free space**. The primary problem for the EMM is finding at least 64K of contiguous free space in which to place the page frame. Even though LIM 4.0 technically doesn't require a 64K page frame, most expanded memory applications will not use expanded memory unless the 64K page frame is present. Frequently, the address areas of various adapter cards need to be shuffled around to free enough space for a contiguous 64K page frame. This process gets complicated with boards such as the IBM 3270 emulator, which has a nonmovable address in most machines.
- **Mapping conflicts.** Most expanded memory managers such as EMM386 use a search algorithm to find unused memory between the hexadecimal addresses C000 through DFFF. The EMM then uses this memory as page frames. Some cards don't reserve their address space until the card is accessed, so the EMM can inadvertently map expanded memory pages on top of the adapter, causing crashes and intermittent operation. This problem is fairly rare, because the page search routine can locate almost all popular adapters.

If problems occur, start by disabling expanded memory to see if a page conflict is causing the problem. If the problem goes away, then the EMM needs to be told to exclude the adapter address from consideration as a page location. The adapter might also have to be

moved. You do this in different ways with different EMMs. For information on how to exclude an address range, consult the documentation for your expanded memory manager.

For more information about taking advantage of memory in the UMBs, see "Optimizing Use of the UMBs" in Chapter 6, "Tips for Configuring Windows 3.1." For related troubleshooting tips, see "Troubleshooting EMS Memory Problems" in Chapter 13, "Troubleshooting Windows 3.1."

Windows Standard Mode and Memory

Standard mode is the normal operating mode for Windows on 80286 machines, providing direct access to extended memory.

Conventional memory takes no special consideration for Windows in standard mode. Windows running in standard mode treats the total free conventional and extended memory as one contiguous memory block. This section describes how standard mode Windows takes advantage of extended memory.

Standard mode does not use expanded memory for Windows operations, but can work with expanded memory for non-Windows applications running under Windows, as described at the end of this section.

For information about entries in SYSTEM.INI that affect performance in Windows standard mode, see the description of the entries in the **[standard]** section in Chapter 4, "The Windows Initialization Files."

Extended Memory and Standard Mode

Flowchart 5.9 Cannot Run Windows in Standard Mode

Standard mode Windows accesses extended memory directly through HIMEM.SYS (or through a third-party XMS driver), providing the total free conventional and extended memory for Windows applications to use. Any non-Windows applications that uses extended memory can run under standard mode Windows.

Standard mode Windows takes advantage of extended memory by:

- Performing code caching with Windows applications.
- Swapping non-Windows applications to extended memory.

Using Extended Memory for Code Caching

Standard mode Windows can speed up its operations by caching code in extended memory through HIMEM.SYS. To perform code caching, Windows takes advantage of certain attributes of Windows applications. The code for each Windows application is divided into segments with specific attributes, including:

- The Movable code segment, which means that the segment can be moved around in memory.
- The Discardable code segment, which means that the segment can be overwritten and then reloaded from disk when necessary.
- The Swapable code segment, which can be swapped to the hard disk.

Each Windows application keeps a minimum amount of code (its "swapsize") loaded in memory. If Windows runs out of memory when a new application executes, Windows discards part of an old application from active memory and overwrites it with new code from the new executable file.

Using Extended Memory with Non-Windows Applications

Non-Windows applications that use extended memory can run under standard mode Windows. The amount of extended memory that the application requires should be specified in the application's program information file (PIF).

For example, your system might have 2048K of extended memory, and you might specify in the PIF that the non-Windows application requires 1024K of extended memory. If standard mode Windows is already using all of the extended memory when you run the non-Windows application, then the information in the first megabyte of extended memory is swapped to disk, and the non-Windows application is given access to the newly free extended memory.

When you switch back to Windows from the non-Windows application, the data in the original 1024K of extended memory is reloaded from disk.

Because this file-swapping process can be slow, don't request any more extended memory in a PIF than is absolutely necessary to run the non-Windows application.

Some non-Windows applications use MS-DOS Extender technology such as VCPI or DPMI to run in protected mode. If you are running such applications under standard mode Windows, you must allocate extended memory in the application's PIF, as described in Chapter 7, "Setting Up Non-Windows Applications." For more information about VCPI, see "DPMI and VCPI Specifications" later in this chapter.

Expanded Memory and Standard Mode

Windows running in standard mode does not use expanded memory at all for its operations. But non-Windows applications running under standard mode Windows can access expanded memory if the system has a physical EMS card such as AST RAMPage! or the Intel Above Board.

The expanded memory manager for the EMS card uses upper memory blocks (UMBs) in the upper memory area. If you suspect a UMB conflict is causing a problem in your system, remove the expanded memory manager to see if that solves the problem. For more information, see "Troubleshooting EMS Memory Problems" in Chapter 13, "Troubleshooting Windows 3.1."

Because non-Windows applications running under standard mode can only use expanded memory with a physical EMS card, an external 386 expanded memory manager such as EMM386.EXE cannot provide the required expanded memory support in Windows. However, an external 386 memory manager can provide expanded memory support for non-Windows applications when you aren't running Windows.

If you want to use memory in the upper memory area, or if you have an application that requires expanded memory but you don't have a physical EMS card, you must run Windows in 386 enhanced mode. For more information about using EMM386.EXE, see Chapter 6, "Tips for Configuring Windows 3.1."

Windows 386 Enhanced Mode and Memory

Windows 386 enhanced mode is the normal operating mode for systems with 80386 and higher processors.

Flowcharts 5.1 and 5.2 Cannot Run 386 Enhanced Mode

When Windows runs in 386 enhanced mode, it adds up the amount of free conventional and extended memory and treats the total amount as an available block of memory, in much the same way as standard mode. The entries in the **[386enh]** section of SYSTEM.INI that control how Windows allocates conventional memory are **PerformBackfill=**, **ReservePageFrame=**, **WindowKBRequired=**, and **WindowMemSize=**. For more information, see these entries in Chapter 4, "The Windows Initialization Files."

Although 386 enhanced mode doesn't use expanded memory for Windows operations, it can simulate expanded memory for use by non-Windows applications, as described later in this section.

You can also create a swap file so that Windows 386 enhanced mode can take advantage of the virtual memory capabilities of the 80386 and higher processors, as described at the end of this section.

WINA20.386 and 386 Enhanced Mode

The MS-DOS 5.0 Setup program installs a virtual device driver that

resolves conflicts between Windows 3.0 and MS-DOS 5.0 when both try to access the HMA. This driver is a read-only file named WINA20.386, which is automatically installed in your root directory. Windows 3.0 will not run in 386 enhanced mode without this file.

- If you upgrade to Windows 3.1, or if you never run 386 enhanced mode with Windows 3.0, you can remove the WINA20.386 file by changing its read-only attribute, then deleting it in the usual way.
- If you are running Windows 3.0 and you move the WINA20.386 file to a different directory, you must ensure that Windows can find this file by adding a **switches=/w** command to CONFIG.SYS and also adding a **device=**[*new path*]**\wina20.386** entry to the **[386enh]** section of SYSTEM.INI.

Extended Memory and 386 Enhanced Mode

Windows 386 enhanced mode uses HIMEM.SYS, the extended memory device driver, to load itself and its drivers into extended memory. As in standard mode, Windows 386 enhanced mode provides the total free conventional and extended memory for Windows applications to use directly. So code caching for Windows applications can also be performed in 386 enhanced mode.

Windows 386 enhanced mode also allows non-Windows applications to run in protected mode if the application uses the DOS Protected Mode Interface (DPMI) specification (as described later in this chapter). Lotus 1-2-3 version 3.1 is an example of a DPMI application.

Windows 386 enhanced mode provides access to extended memory for non-Windows applications by creating virtual machines up to 640K in size, or the size defined by the **CommandEnvSize=** entry in the **[NonWindowsApp]** section of SYSTEM.INI.

Each virtual machine inherits the environment present before you started Windows. This means that every driver and terminate-and-stay-resident program (TSR) loaded before running Windows

consumes memory in every subsequent virtual machine. The memory available within each virtual machine under 386 enhanced mode is slightly less than the free memory available at the command prompt before you start Windows, depending on your system configuration.

When creating virtual machines for non-Windows applications, Windows 386 enhanced mode uses the upper memory area for two purposes:

- To allocate translation buffers for protected-mode API calls for MS-DOS and the network.
- To place the page frame for expanded memory (if required).

Frequently, all of the free pages in the upper memory area are used by 386 enhanced mode. The memory conflicts that can result are discussed in the next section.

Expanded Memory and 386 Enhanced Mode

Windows 386 enhanced mode does not use expanded memory for itself, and Windows applications don't need expanded memory, because they run in protected mode and can access extended memory directly. However, Windows 386 enhanced mode can create expanded memory for use by non-Windows applications such as Lotus 1-2-3 that require or can take advantage of expanded memory.

Windows 386 enhanced mode automatically provides expanded memory for non-Windows applications that require it when you run such applications under Windows. It cannot provide this memory, however, if you load EMM386.EXE with the **noems** switch. Use the **ram** switch when loading EMM386.EXE in CONFIG.SYS, or use the **x=mmmm-nnnn** parameter to allocate enough space in the upper memory area for Windows to create an EMS page frame.

Note The expanded memory required by a non-Windows application should be allocated with PIF parameters as described in Chapter 7, "Setting Up Non-Windows Applications."

Page-Frame Conflicts in 386 Enhanced Mode

Windows 386 enhanced mode provides additional page frames for LIM 4.0 expanded memory in all virtual machines. But most non-Windows applications use only the 64K page frame itself, not the additional bankable pages in conventional memory that LIM 4.0 EMS supplies. So to use expanded memory to run non-Windows applications, you must have a contiguous 64K page frame, made up of four contiguous 16K pages in the upper memory area. The key issue, therefore, for expanded memory under Windows is page-frame conflicts.

The adapters installed on a system can break up the free area in the upper memory area so that there is no 64K contiguous area to place the page frame, and hence no free expanded memory for running non-Windows applications. If this problem occurs, you might have to rearrange the adapter memory locations.

This is easiest to do on a Micro Channel machine such as the IBM Personal System/2, which allows you to change adapter memory locations by booting with the PS/2 Reference Disk and choosing Change Configuration. A similar procedure is available on most Extended Industry Standard Architecture (EISA) bus machines, such as the Compaq SystemPro and HP Vectra 486.

For Industry Standard Architecture (ISA) bus machines, such as the IBM AT and Compaq 386, you might have to open the case and flip DIP switches on the cards to change their memory addresses. Refer to your hardware manual, and use Figure 5.2 (page 232) as a reference when readdressing adapters to open a 64K page frame.

You can also disable expanded memory entirely (and 64K page frame support) in Windows 386 enhanced mode by setting **NoEMMDriver=yes** in the **[386enh]** section of SYSTEM.INI.

Other related entries in the **[386enh]** section of SYSTEM.INI are **EMMSize=** and **IgnoreInstalledEMM=**. For more information, see the descriptions of these entries in Chapter 4, “The Windows Initialization Files.”

Placing Translation Buffers in the Upper Memory Area

In 386 enhanced mode, Windows allocates buffers in the upper memory area to translate MS-DOS and network application program interface (API) calls from Windows protected mode to MS-DOS real mode. (Because the upper memory area is within the first megabyte of address space, it can be accessed by MS-DOS in real mode on an 80386 and higher processor.) Ideally, there will be enough free space in the upper memory area to place both the translation buffers and any expanded memory page frame required. But on many systems there isn't enough room, and you must choose to eliminate the expanded memory page frame or allocate the translation buffers in conventional memory (instead of in the upper memory area).

If the translation buffers are allocated in conventional memory, they take up memory in every virtual machine Windows creates, leaving less space in the virtual machines to run non-Windows applications. To compound the problem, the translation buffers can be allocated either in the upper memory area or in conventional memory, but never half-and-half.

To specify a preference, set the value for the **ReservePageFrame=** entry in the **[386enh]** section of SYSTEM.INI. If **ReservePageFrame=true** (the default), then Windows allocates the page frame first and the translation buffers second. Usually, on machines with full UMBs, the translation buffers are forced into conventional memory, but this lets you use expanded memory for non-Windows applications.

If **ReservePageFrame=false**, the translation buffers are allocated in the UMBs first, followed by the page frame if there is still room. This setting gives you the most free memory in virtual machines, but you might not have enough expanded memory for non-Windows applications.

Controlling UMB Mapping in 386 Enhanced Mode

You can control the placement of expanded memory page frames and the translation-buffer mapping with the **EMMExclude=** or **ReservedHighArea=** entries in the **[386enh]** section of SYSTEM.INI. To explicitly exclude an area of the upper memory area from mapping by Windows 386 enhanced mode, set a value for **EMMExclude=**, for example, **EMMExclude=E000-EFFF**. Because there is no standard for hardware implementation of the E000-EFFF area, it is frequently necessary to exclude this range, so that 386 enhanced mode can function properly. Windows 386 enhanced mode detects and excludes the area for most adapter cards automatically.

Any values set with the **x=** switch in the line that loads EMM386.EXE in CONFIG.SYS will override the value set for **EMMExclude=** in SYSTEM.INI.

The **ReservedHighArea=** entry provides the same support, but for 4K ranges, rather than 16K. Other entries related to UMB mapping in the **[386enh]** section of SYSTEM.INI are **EMMInclude=**, **EMMPageFrame=**, **EMMSize=**, and **UseableHighArea=**. If you suspect conflicting use of the upper memory area, use **EMMExclude=**. Because Windows will use all free pages in the upper memory area automatically, there are few uses for **EMMInclude=**, **EMMPageFrame=**, and **UseableHighArea=**. For more information, see the descriptions of these entries in Chapter 4, "The Windows Initialization Files," and see also "Troubleshooting EMS Memory Problems" in Chapter 13, "Troubleshooting Windows 3.1."

Windows/386 2.x did not use the E000-EFFF area of the adapter segment unless specifically instructed to do so. Windows 386 enhanced mode uses this segment unless the machine identifies itself as a PS/2. As a side note, most of the entries in the **[386enh]** section of SYSTEM.INI that begin with the letters "EMM" control both placement of the expanded memory page frames and the translation-buffer mapping. The letters "EMM" are used only for backward compatibility; except for the **EMMPageFrame=** entry, they no longer apply only to the expanded memory page frame. The **LastEMMSeg=** parameter used in Windows/386 2.x has been

dropped for Windows 3.x.

About 386 Expanded Memory Managers

An expanded memory manager such as EMM386.EXE can provide expanded memory for non-Windows applications on 80386 and higher machines without a physical EMS card when you aren't running Windows. For more information about EMM386.EXE, see Chapter 6, "Tips for Configuring Windows 3.1," in the *Windows Resource Kit;* see also "Freeing Expanded Memory" in Chapter 12 of the *MS-DOS 5.0 User's Guide and Reference*.

Some 386 expanded memory managers such as EMM386.EXE and CEMM.EXE allow Windows to turn them off when Windows is run. CEMM.EXE requires that no expanded memory be in use when you start Windows. (That is, set **NoEMMDriver=yes** in the **[386enh]** section of SYSTEM.INI.)

W

Both Windows 3.1 and MS-DOS 5.0 support the upper memory area mechanisms defined for the LIM 3.2 and LIM 4.0, so 386 expanded memory managers such as EMM386.EXE, 386MAX.SYS, and QEMM.SYS can load network drivers and other software devices in the upper memory area and still run with Windows 386 enhanced mode.

Virtual Memory and 386 Enhanced Mode

Virtual memory has been widely used for years with mainframes, but first came to PCs with the introduction of the IBM/Microsoft OS/2 operating system. Windows 386 enhanced mode goes beyond OS/2 to offer virtual memory using the special demand-paging capabilities of the Intel 80386 processor.

When virtual memory is used with Windows 386 enhanced mode, some of the program code and data are kept in physical memory while the rest is swapped to the hard disk in a swap file. Whenever a reference is made to a memory address, it can be used without interruption if the information is currently in physical memory. If the information isn't in physical memory, a page fault occurs and the Windows Virtual Memory Manager (VMM) takes control, pulling the required information back into physical memory and, if necessary, swapping other information to the disk. All of this activity is invisible to the user, who only sees some hard disk activity.

Windows applications can use virtual memory without being specially written to take advantage of it, because Windows handles the memory management, allocating however much memory the application requests. With Windows managing virtual memory, you will see much more memory available than is installed in your machine when you choose About Program Manager or About File Manager from the Help menu.

A major benefit of using virtual memory is that you can run more programs simultaneously than your system's physical memory would usually allow. The drawbacks are the disk space required for the virtual memory swap file and the decreased execution speed when page swapping is required. However, it's usually better to run a program slowly in virtual memory than to not be able to run it at all.

Creating Swap Files for Virtual Memory

W

In Windows 3.1, you can create a swap file for virtual memory during Setup, or you can choose the 386 Enhanced icon in Control Panel to change the swap file at any time. For details about using the 386 Enhanced Mode dialog box, see Chapter 14, "Optimizing Windows," in the *Windows User's Guide.*

Figure 5.4

Virtual Memory
dialog box

To display this dialog,
click the Virtual Memory button in the 386 Enhanced dialog

Click the Change button to display this extended dialog box

You can create either a temporary or a permanent swap file. A permanent swap file improves the speed of the Windows virtual memory system because the file is contiguous, so accessing it requires less overhead than the normal MS-DOS file created for a temporary swap file.

- A temporary swap file named WIN386.SWP is created on the hard disk while Windows is running, then deleted automatically when you exit Windows. This swap file is not a hidden or system file, and it can shrink or grow in size as necessary. The entry for **PagingFile=** in the **[386enh]** section of SYSTEM.INI defines the filename and path for the temporary swap file. You need about 1.5 MB of free hard disk space on the paging drive for a temporary swap file.

- A permanent swap file is a hidden file named 386PART.PAR, which has a system attribute and is always created in the root directory of the specified drive. Windows also creates a read-only SPART.PAR file in the WINDOWS directory that tells Windows where and how large the permanent swap file is. Because a permanent swap file must be contiguous, you cannot create a permanent swap file bigger than the largest contiguous free segment of your hard disk. You cannot create a permanent swap file if Stacker is running on your system.

You can specify the type and size of a swap file and the drive where it's located in the Virtual Memory dialog box. A permanent swap file is always created in the root directory. But you can specify a subdirectory as the location for a temporary swap file as a **PagingFile=** value in the **[386enh]** section of SYSTEM.INI.

When you install Windows, Setup checks whether your hard-disk controller is compatible with 32-bit disk access. If so, the 32-Bit Disk Access check box appears. Select this check box if your system has only a small amount of free memory and you want to increase performance for the MS-DOS Prompt. When this option is checked, you can run more instances of MS-DOS Prompt and switch between them faster. If you have multiple instances of MS-DOS Prompt running and the applications in them all access disk drives, the access time is faster with 32-bit disk access. For a technical discussion of this features see "FastDisk: An Introduction to 32-Bit Disk Access" in Appendix D, "Articles."

The Windows virtual memory utility that creates swap files supports only hard disks that use 512-byte sectors. If 512-byte sectors are not being used, this indicates a nonstandard configuration, such as a third-party driver. Never create a permanent swap file on a drive that uses a partitioning driver, with the exception of the Compaq ENHDISK.SYS utility.

Before you can create a large permanent swap file, you should compact your hard disk with a disk compacting utility. If an error message reports that your swap file is corrupted, delete the current swap file and create a new one.

The related entries for swap files in the **[386enh]** section of

SYSTEM.INI are **MaxPagingFileSize=**, **MinUserDiskSpace=**, **PagingDrive=**, and **PagingFile=**. For more information, see the descriptions of these entries in Chapter 4, "The Windows Initialization Files."

Note Do not attempt to create a swap file on a RAM disk. A swap file created on a RAM disk is self-defeating, because you sacrifice physical memory to create virtual memory.

Demand Paging and Virtual Memory Management

Windows 386 enhanced mode manages virtual memory as a demand-paged system that uses a virtual memory manager (VMM) and a pageswap device (which is built into the WIN386.EXE file). This means that pages of data are brought into physical memory when they are referenced, and the system does not try to predict which pages will be required in the future.

The Windows VMM maintains the virtual memory page table that lists the pages currently in physical memory and those swapped to disk. Because 386 enhanced mode is a multitasking environment, the VMM page table also lists which memory pages belong to which processes. When the VMM needs a page not currently in physical memory, it calls the pageswap device, which allocates virtual memory and maps pages into and out of physical memory.

Some virtual memory systems rely on program segmentation to do their work. Although the code for a Windows application is segmented as described earlier, Windows virtual memory management is not related to this segmentation. All virtual and physical memory is divided into 4K pages, and the system is managed on this basis. Page mapping starts at 0000K and works its way up.

Two kinds of pages can be allocated: physical pages and virtual pages. The possible amount of physical pages is the amount of physical memory in the machine divided by 4K. Memory allocated to an application is made up of virtual pages, and at any time a virtual page can be in physical memory or swapped to the hard disk.

The entries in the **[386enh]** section of SYSTEM.INI that control

paging for virtual memory are:

LocalLoadHigh=	**Paging=**	**SysVMEMSLimit=**
MaxBPs=	**PagingDrive=**	**SysVMEMSLocked=**
MaxPhysPage=	**PagingFile=**	**SysVMV86Locked=**
MinUnlockMem=	**PageOverCommit=**	**SysVMXMSLimit=**

For more information, see the descriptions of these entries in Chapter 4, "The Windows Initialization Files."

The LRU Algorithm for Virtual Memory Management

The Windows VMM swaps pages using a Least Recently Used (LRU) page replacement algorithm. This means pages that have not been accessed for the longest period of time are the first ones to be swapped to the disk.

The VMM page table contains flags for the Accessed and Dirty attributes of each page. *Accessed* means that a process has made a reference to the page since it was originally loaded. *Dirty* means that a "write" has been made to the page since it was loaded. Because a "write" qualifies as an "access," the Dirty attribute implies the Accessed attribute.

If physical memory space cannot be found when a process requests additional memory, Windows uses the LRU algorithm to decide which pages to swap to the hard disk to fulfill the request. This decision is a three-step process:

1. Windows scans the VMM page table looking for pages with neither an Accessed nor a Dirty attribute. During the scanning process, Windows clears the Accessed attribute from all the pages.
2. If Windows finds enough pages meeting the Not Accessed/Not Dirty requirement, it swaps those pages to the hard disk and gives the resulting free memory to the process.

3. If Windows cannot find enough pages the first time, then it repeats the scan. Theoretically, more pages will meet the requirements because the Accessed attribute was cleared in the first scan. If the second scan doesn't find the required pages, then Windows swaps pages to the hard disk regardless of their attributes.

The related entries in the **[386enh]** section of SYSTEM.INI are **LRULowRateMult=**, **LRURateChngTime=**, **LRUSweepFreq=**, **LRUSweepLen=**, **LRUSweepLowWater=**, and **LRUSweepReset=**.
For more information, see the descriptions of these entries in Chapter 4, "The Windows Initialization Files."

Swapping Pages to a Network Drive

We recommend that you do not swap pages to a network drive. Paging to a network drive is possible, but it's extremely slow. If you must page to a network drive, use a permanent swap file. Also, the directory must not have an MS-DOS read-only attribute, and you must have both create and write access to the directory. Do not set the value for **PagingDrive=** or **PagingFile=** in SYSTEM.INI to a Novell network drive, because Novell networks are not compatible with MS-Net Redirector.

Other Memory Management Issues

This section provides brief information on various memory management issues:

- DPMI and VCPI specifications
- MS-DOS and Windows 3.1
- Memory and startup requirements
- System resources and memory

DPMI and VCPI Specifications

The DOS Protected Mode Interface (DPMI) was developed by a group of industry leaders. Several members of the DPMI committee

also helped create the Virtual Control Program Interface (VCPI). DPMI is primarily a creation of Microsoft, and VCPI was formulated primarily by Phar Lap Systems. DPMI and VCPI solve two different problems.

Applications that use MS-DOS Extenders can execute code in the protected mode of the 80286 or 80386 processor. DPMI provides a standard method for such applications to switch the 80286 processor to protected mode and to allocate extended memory. Hundreds of applications use various types of MS-DOS Extenders, and those that do not already support DPMI require minor modifications to do so.

VCPI provides an interface that allows applications using MS-DOS Extenders on 80386 machines to run simultaneously with 386 expanded memory managers. For example, QEMM.EXE, 386MAX.EXE, and CEMM.EXE support the VCPI specification. Windows 3.1 supports VCPI in both standard mode and 386 enhanced mode. Windows 3.0 does not support VCPI.

MS-DOS 5.0 and Windows 3.1

Many of the changes in MS-DOS 5.0 make it a more robust platform for Microsoft Windows, enhancing its ability to make the best use of your system's memory. For the best performance from Windows 3.1 (and other applications), we suggest that you upgrade your PC's operating system to MS-DOS 5.0 if you haven't already done so.

On 80386 and 80486 PCs, with MS-DOS 5.0 you can load memory-resident programs such as device drivers, TSRs, and network software into the upper memory area, thereby freeing conventional memory. To do this, you must load HIMEM.SYS and EMM386.EXE, then load the memory-resident programs using the **devicehigh** command in CONFIG.SYS and the **loadhigh** command in AUTOEXEC.BAT.

For more information about taking advantage of MS-DOS 5.0 to optimize your system configuration, see Chapter 6, "Tips for Configuring Windows 3.1."

Running Windows Standard Mode with MS-DOS 5.0

- Use the Windows Task List to swap applications (or press ALT+TAB). Using the MS-DOS 5.0 Task Swapper while running Windows is redundant, incurring unnecessary conventional memory overhead.
- Load MS-DOS 5.0 into the HMA, regardless of whether your PC is an 80286, 80386, or 80486. This will provide more conventional memory for non-Windows applications running under Windows standard mode.

Running Windows 386 Enhanced Mode with MS-DOS 5.0

- Load MS-DOS 5.0 into the HMA, and load whatever else fits into the upper memory area. The conventional memory you free by loading MS-DOS 5.0 into the HMA is also free in every virtual machine, giving more memory to every non-Windows application you run in Windows 386 enhanced mode.
- Use the Task List in Windows 3.1 to switch between applications (or press ALT+TAB). Windows 386 enhanced mode allows multitasking, so that multiple applications can run in the background. The MS-DOS 5.0 Task Swapper can only task-switch, which means that the swapped application is suspended.

Memory and Windows Startup Requirements

Flowchart 1.1 System Requirements

Windows starts automatically in the appropriate Windows operating mode, depending on your system configuration. However, you can start Windows in a particular operating mode by using one of these command-line switches:

- **win /s** to run in standard mode
- **win /3** to run in 386 enhanced mode

Standard Mode Startup Requirements

For Windows to start automatically in standard mode, the system must have:

- 80286 or higher processor
- 256K of free conventional memory
- 192K of free extended memory
- An XMS driver such as HIMEM.SYS already loaded

Conventional and extended memory requirements are mutually dependent for standard mode and are not fixed.

386 Enhanced Mode Startup Requirements

For Windows to start in 386 enhanced mode, the system must have:

- 80386 or higher processor
- 256K of free conventional memory
- 1024K of free extended memory
- An XMS driver such as HIMEM.SYS already loaded

Windows 386 enhanced mode requires between 580K and 624K combined conventional and extended memory to run. A typical installation requires a minimum of 192K free conventional memory at the command prompt plus sufficient extended memory available to run in 386 enhanced mode. Windows can start under low memory in 386 enhanced mode because it provides virtual memory support, but it can be extremely slow because of the extra disk swapping that Windows must perform.

All numbers are approximate and can vary widely depending on the Windows device drivers present, the MS-DOS version, the display adapter, and other factors. For example, on Compaq 386 machines, 128K of extended memory is recovered from shadow RAM. Memory requirements take into account memory that can be recovered from SMARTDrive, down to the minimum cache size specified.

Windows checks for a minimum of 1 MB of free extended memory before it automatically starts in 386 enhanced mode. If it finds less, it tries to run in standard mode. On an 80386 with 2 MB or less system memory, if Windows doesn't find enough free extended memory, it runs in standard mode. To free more extended memory so that you can run in 386 enhanced mode, you might try reducing

the amount of extended memory that SMARTDrive uses by setting its **MinCacheSize** parameter to **0**.

The entries in the **[386enh]** section of SYSTEM.INI that control how Windows allocates conventional memory for startup and for use in 386 enhanced mode are **SysVMEMSRequired=**, **SysVMXMSRequired=**, **WindowKBRequired=**, and **WindowMemSize=**. For details, see the descriptions of these entries in Chapter 4, "The Windows Initialization Files."

Memory and the Windows System Resources

Flowchart 5.8 Out-of-Memory Errors

The Help About dialog box in Program Manager and File Manager (and in many other Windows applications) shows the percentage of free system resources and the amount of free memory. The system resources percentage reflects the memory used by the core Windows internal structure.

Three core files make up the part of Windows that runs Windows applications:

- The kernel file (KRNL286.EXE or KRNL386.EXE) loads and executes Windows applications and handles their memory management.
- GDI.EXE manages graphics and printing.
- USER.EXE controls user input and output, including the keyboard, mouse, sound driver, timer, communications ports, and window management.

In Windows 3.x, these files are in the Windows SYSTEM subdirectory. (In Windows 2.x these modules were linked into the files WIN200.BIN and WIN200.OVL, so you didn't see them in the WINDOWS directory.)

W

Both GDI and User have storage areas called heaps, which are limited to 64K in size. GDI has a local heap; User has a menu heap and a user heap, each with 64K storage space. The available system resources reflect the remaining free percentage after combining the GDI local heap and the user and menu heaps in User. This increases by 64K the amount of heap space that was available in Windows 3.0. In Windows 3.1, Program Manager icons are handled separately and do not use the User heap space.

To see how much of the system resources a particular application uses, note the amount of system resources available before and after the application runs. (Choose About Program Manager from the Help menu and check the amount of system resources listed in the dialog box.)

Although Windows 3.1 allows you to run many more simultaneous Windows applications than any previous Windows version, you may get an out-of-memory message that indicates your system is low on system resources. This is because every window and sub-window created requires User and GDI local heap space. The system resources can be exhausted if enough objects are created by the Windows applications.

An important element of memory management for Windows applications that is not included in the system resources percentage is the number of selectors. A selector is a memory pointer that is consumed with each memory allocation made by a Windows application. If a Windows application allocates many small data objects, it is possible to run out of selectors, resulting in an out-of-memory message.

The efficiency with which a Windows application handles its data objects can help in this situation. If you experience a chronic problem with a particular application while few other applications are loaded, contact the application vendor so the company becomes aware of the problem and corrects it.

For more information about low-memory and out-of-memory conditions, see Chapter 13, “Troubleshooting Windows 3.1.”

SMARTDrive 4.0: A Technical Discussion

W

This section describes the new MS-DOS disk cache, SMARTDrive 4.0, which replaces SMARTDrive version 3.x. The section is written for experienced technical users who want a deeper understanding of the utility.

About SMARTDrive 4.0

SMARTDrive 3.x is a read-only track cache that caches on a track basis and for read operations only. The internal data structures are tied to the logical geometry of the disk. It caches at the ROM BIOS Int 13 level and uses the BIOS specified disk geometry to decide the size of its caching element (that is, track size). This leads to problems (described below), so Microsoft chose to implement a new cache in SMARTDrive 4.0.

Figure 5.5

SMARTDrive 3.x

SMARTDrive 4.0 is designed as a block-oriented disk cache. It hooks into the system at the MS-DOS device driver level rather than the ROM BIOS Int 13 level. Each block device driver on the MS-DOS device driver chain is "front-ended" by a SMARTDrive 4.0 module that provides the actual caching. This yields the following benefits:

- **Independent of Int 13 interface.** Many device drivers do not use the Int 13 interface. This means that SMARTDrive 4.0 can cache these devices where SMARTDrive 3.x could not. Examples are Bernoulli, some hard cards, and many SCSI and WORM drives.
- **Independent of disk geometry.** Some disk managers and disk controllers use a disk geometry mapping scheme that causes the "logical" geometry (that is, what MS-DOS sees) to be different from the physical geometry. Examples are many PS/2 systems, Ontrack Disk Manager, and several disk controllers.

Int 13-based caches are sensitive to this and often have problems. For example, Ontrack Disk Manager will actually change the ROM BIOS-specified disk geometry on the fly and thus confuse SMARTDrive 3.x.

An API determines the true geometry, but this requires disk-manager detection and generally complicates matters. Often logical tracks will actually cross physical track boundaries, which then causes track caches to incur performance penalties (intertrack seeks and rotational latencies). Also, to get around the ROM BIOS 1024 cylinder limitation, disk managers and controllers will "fold" multiple tracks into one logical track. This yields the above problem, as well as forcing track caches to have a very large track buffer. In some cases, this is as large as 31.5K and must reside in low memory. The design of SMARTDrive 4.0 eliminates the geometry mismatch problem.

Figure 5.6

SMARTDrive 4.0

SMARTDrive 4.0 is also a write-behind cache. It adds significant performance improvement where files are being written and implements a "valid" bits scheme to avoid thrashing the disk in case of random access I/O.

SMARTDrive 4.0 currently uses a FIFO replacement algorithm and implements a shrink algorithm that frees memory for Windows in a way similar to SMARTDrive 3.x. The difference is that SMARTDrive 4.0 watches for the Windows startup broadcast while SMARTDrive 3.x provides an IOCTL interface. The net effect is identical, but SMARTDrive 4.0 is much simpler. When Windows quits, the process is reversed and the memory is reacquired by SMARTDrive 4.0.

The following table summarizes the SMARTDrive 3.x design problems, what other disk-cache software does, and the solutions employed in SMARTDrive 4.0.

Feature	*SMARTDrive 3.x*	*Other disk caches*	*SMARTDrive 4.0*
Read-Only Cache	Causes writes to proceed at the same or slower speed as with no cache.	All others solve this.	Lazy write caching. Valid bits to avoid thrashing.
Track Cache	Tied to "logical" disk geometry, which disk managers and some disk controllers change, causing a "single" track read to actually be multiple − performance hit.	All others have configurable cache element size.	Support configurable cache element size.
	Some disks have very large track sizes (56 on some mapping controllers), which forces a very large track buffer into low memory and thus a large low memory footprint.	Same	Same
	Internal data structures and algorithms are tied to the size of track and thus it is very hard to retrofit variable block size.	Same	Same
Int 13 based	Many popular device drivers do not go through Int 13 interface and therefore don't get cached. Bernoulli, WORM drives, and SCSI are some examples of non-Int 13 devices that SMARTDrive does not cache.	Mixed. Some cache Int 13; some cache at device driver level.	Front-end MS-DOS device drivers. This means that block device in MS-DOS will be cached.
	Disk manager drivers and some controllers cause a geometry mismatch.	Some DM drivers provide DDI, which Int 13 caches need to call (PC-KWIK).	Above solution eliminates need to know true geometry.

SMARTDrive 4.0: Frequently Asked Questions

Q: What is double buffering?

A: Certain disk controllers support a concept called bus mastering, where the actual disk controller takes over the bus to transfer data to or from system RAM. Some SCSI controllers have this feature. A problem occurs when running in the virtual 8086 mode that Windows 3.x virtual machines provide. Popular memory managers also use virtual 8086 mode. The read or write address that is passed to MS-DOS is often not the same as the actual physical memory address. This can cause data to be read from the wrong location or, worse, can cause data to be written to the wrong RAM. The result can be erratic system behavior.

Microsoft created a standard called Virtual DMA Services, which provides an interface that allows these bus master controllers to get the correct address and avoid the problem. However, some older bus master controller cards do not support this standard. So we have added a feature to SMARTDrive that provides a memory buffer with physical and virtual addresses that are the same, so we avoid the problem at the cost of 2.5K of conventional memory and a small amount of performance (the cost of moving the data to and from the buffer). This feature is used by placing the line **device=smartdrv.exe /double_buffer** in CONFIG.SYS. This only installs the double-buffer driver, not the cache. (The cache must be installed in AUTOEXEC.BAT.)

Q: Do I need double buffering?

A: The Windows 3.1 Setup program tries to determine whether your system needs double buffering, and if so, installs SMARTDRV.EXE in your CONFIG.SYS file for double buffering. Most disk controllers do not need double buffering. These include all MFM, RLL, and IDE controllers as well as many ESDI and SCSI devices. In the cases where Setup is unable to determine whether double buffering is needed, it will install the driver in CONFIG.SYS, possibly erring on the side of safety. We have added a feature to SMARTDrive to help you determine if double buffering is unneeded and to allow you to remove the driver. After your system is running with SMARTDrive loaded, type **smartdrv** at the command prompt and press ENTER. You will see something similar to the following screen.

```
Microsoft SMARTDrive Disk Cache version 4.0
Copyright 1991,1992 Microsoft Corp.
Cache size: 1,048,576 bytes
Cache size while running Windows: 1,048,576 bytes          Disk Caching Status
drive    read cache   write cache   buffering
A:           yes          no            no
B:           yes          no            no
C:           yes          yes           yes
D:           yes          yes           -
For help, type smartdrv /? at the MS-DOS prompt.
```

Notice the column labeled “buffering.” For each drive that is being cached, it can have one of three values: **Yes** to indicate that double buffering is needed. **No** to indicate that buffering is not needed, and the “**–**” symbol to indicate that SMARTDrive has not yet determined the necessity of double buffering. If the buffering

column has all No's in it, the double buffer driver is not needed.

Q: Why is SMARTDrive in both my CONFIG.SYS and AUTOEXEC.BAT files?

A: There are really two device drivers in a single file: a disk cache and a double buffer driver. See above for a description of double buffering. The cache component of SMARTDRV.EXE is installed in AUTOEXEC.BAT and the double buffer driver is installed in CONFIG.SYS.

Q: Does SMARTDrive work with Stacker?

Flowchart 1.8 Stacker

A: Yes. SMARTDrive works quite well with Stacker. SMARTDrive is aware of Stacker and will automatically cache the underlying drive that Stacker uses. This provides significantly better cache utilization by increasing the effective size of the cache by the compression ratio of Stacker. However, do not cache the actual "stacked" volume. Only the underlying (uncompressed) drive should be cached.

Q: Why doesn't my Stacker Volume show up in the SMARTDrive status screen?

A: This is because SMARTDrive is caching underneath Stacker. You should see the underlying drive letter listed.

Q: How can I make sure that data written to the disk is really on the disk and not still in the cache when I reboot my machine? Won't my data get lost when I reboot?

A: SMARTDrive goes to great lengths to avoid data loss. When it detects the CTRL+ALT+DEL reboot key sequence, SMARTDrive takes control and makes sure that all data has been written to the actual disk. You might see a box in the upper left corner of the display asking you to wait while this happens. SMARTDrive also writes all data to the disk when an application calls the MS-DOS reset disk function to make sure that all data in the MS-DOS buffers gets written to disk.

To force all data to be written to the disk, type **smartdrv /c** at the command prompt and press ENTER.

If you use a third-party program to reboot your machine from a batch file, you should make sure that you have the above line in

the batch file before the reboot program. Failure to do so may cause loss of data.