

*unDOS*TM / *unWin*TM

PRODUCT BRIEF

unDOS and *unWin* serve to bridge code written for other environments to SMX[®]. *unDOS* emulates many of the most-common DOS int 21h routines and some BIOS routines, including file i/o, time and date, interrupt vector, memory management, internationalization, environment strings, and VGA. Similarly, *unWin* emulates many Win32 routines, in these same categories. *unWin* is for use in 32-bit systems and includes *unDOS*.

One purpose of these products is to allow use of the Borland and Microsoft C Run Time Libraries (RTL's) without DOS or Windows being present. Many of the functions in these libraries is operating system dependent. This is especially true of the 32-bit RTL's, which make many calls to Windows DLL's. The real mode and 16-bit protected mode RTL's likewise depend on DOS services.

In real mode, it is possible to run SMX on DOS in order to use DOS services, but DOS requires payment of royalties and is not well-suited for a multitasking environment. DOS calls must be protected by a single semaphore to prevent reentering them, whereas *unDOS* services are reentrant. *unDOS* avoids royalties and allows using a multitasking environment. Furthermore, *unDOS* and our Real Mode Bootloader make it possible to use additional memory above the 640KB limit imposed by DOS. See the "Real Mode Availability in the First Megabyte" chart on page 4.

In 16-bit protected mode, it is not possible to directly use DOS services, since DOS runs in real mode and it would be necessary to switch to real mode to make these calls. This is what DOS extenders do, but these are not suitable for most embedded systems because: 1) they greatly increase interrupt latency, 2) they cannot be preempted and thus delay task switching, and 3) they complicate debugging. *unDOS* is better because it

emulates certain DOS and BIOS services in protected mode without requiring switching to real mode or virtual 86 mode, as DOS extenders do.

In 32-bit protected mode, the Borland and Microsoft RTL's depend heavily on Windows services (i.e. kernel32.dll). Without *unWin*, SMX would have to run under Windows in order for the application to use the RTL. Although we support this as a development convenience, it is not suitable for most embedded systems since it lacks real-time responsiveness and requires payment of Windows royalties. *unWin* provides an alternative, by emulating some of the Win32 API calls needed by common RTL functions so that the linkages are satisfied and operations are performed using corresponding SMX services, or they are stubbed off, if inappropriate.

There is so much reliance on Win32 services by the Borland and Microsoft RTL's that *unWin* should be purchased by all SMX users who intend to use either of these compilers and expect to use the RTL functions. (Note that currently, *unWin* does not make the entire RTL available. New Win32 emulation routines are periodically added. Please consult with us about the functions you wish to use.) *unWin* implements the critical section functions necessary for support of the Borland multithread RTL. (*unWin* support of the Microsoft multithread RTL coming soon.)

Another use of *unDOS* and *unWin* is to support third-party libraries. Just as with the C RTL, many libraries depend upon BIOS, DOS, or Win32 services. *unDOS* and *unWin* implement these services eliminating the need to modify third-party library code. (Note that many third-party libraries were written for use with DOS extenders and additionally call DPMI functions. *pmEasy*[®] services these DPMI requests.)

unDOS also permits migrating legacy application code used in real-mode, DOS-dependent environments to protected mode environments, without introducing DOS extenders. Companies frequently come to us who have run out of memory and need to port their applications to protected mode. *unDOS* is a big help. We offer a free TSR called *DOSTap* which can be used to determine DOS and BIOS dependencies in existing applications before purchasing *unDOS*.

unDOS and *unWin* log unsupported calls to the screen or disk, or trap them in the debugger.

If you are porting a legacy application or expect to rely on certain C library services, please discuss your needs with us in advance. You can implement additional

services following our example, or we will contract to add them.

unDOS supports real mode, 16-bit protected mode, and 32-bit protected mode. *unWin* is for use in 32-bit protected mode, only, and includes *unDOS*. Both are compatible with all Micro Digital products. Depending upon the application, other Micro Digital products such as *smxFile* and *pmEasy* may be required. Free support and updates are included for 6 months. Source code is included. Royalty free.

Sizes

	<u>Code</u>	<u>Data</u>
16-bit	5 KB	0.5 KB
32-bit	9 KB	1 KB

Summary of Win32 Services Provided by *unWin*

Below is a list of the Win32 functions emulated by *unWin*. Note that some Win32 services are fairly broad in scope, and *unWin* implements only part of their functionality. Please consult the *unDOS/unWin* Developer's Guide for discussion of any limitations of our implementations of these functions.

CloseHandle	GetVolumeInformationA
CreateFileA	HeapAlloc
DeleteCriticalSection	HeapCreate
DeleteFileA	HeapDestroy
EnterCriticalSection	HeapFree
FileTimeToLocalFileTime	InitializeCriticalSection
FileTimeToSystemTime	LCMapStringA
FindClose	LeaveCriticalSection
FindFirstFileA	MulDiv
FindNextFileA	RaiseException
FlushFileBuffers	ReadFile
GetACP	SetFilePointer
GetCPIInfo	SetHandleCount
GetCurrentDirectoryA	SetLastError
GetCurrentProcess	SetLocalTime
GetFileAttributesA	SetStdHandle
GetFileType	SetTimeZoneInformation
GetFullPathNameA	TerminateCurrentProcess
GetLastError	TlsAlloc
GetLocalTime	TlsFree
GetStartupInfoA	TlsGetValue
GetStdHandle	TlsSetValue
GetTimeZoneInformation	VirtualQuery
GetVersionExA	WriteFile

Summary of BIOS and DOS Services Provided by *unDOS*

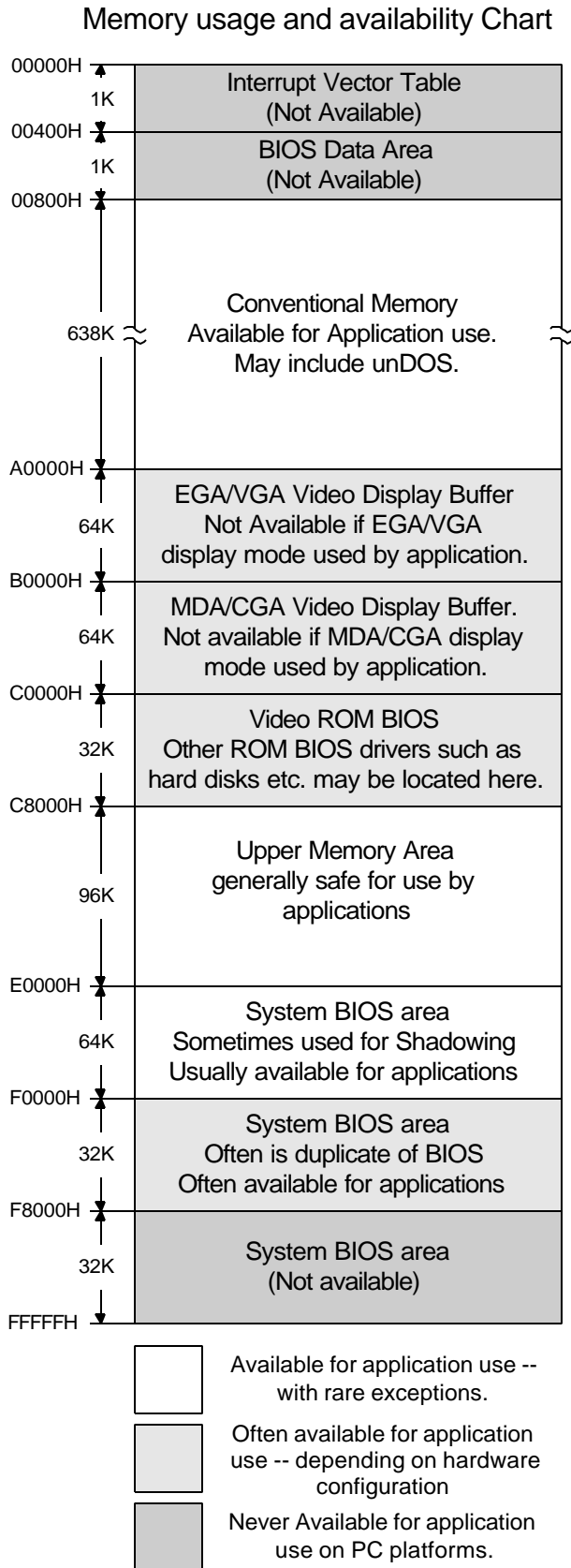
Interrupt 10h Services (BIOS)

00h	Set Graphics Mode (by switch to real mode using DPMI 300h)
02h	Set Cursor Position
0Fh	Read Current Video State
10:00h	Set Individual Palette Registers (EGA/VGA)
11h	Character generation has been stubbed out; forces graphics packages to use its own fonts.
12:00h	Return Video Information (EGA/VGA)
1A:00	Read Display Combination Code (VGA)

Interrupt 21h Services (DOS)

0Eh	Set Default Drive
19h	Get Default Drive
1Ah	Set DTA address
25h	Set Interrupt Vector
2Ah	Get Date
2Bh	Set Date
2Ch	Get Time
2Dh	Set Time
2Fh	Get DTA address
30h	Get Version Number (6.20)
35h	Get Interrupt Vector
38h	Get Country information (set not supported)
39h	Create Directory
3Ah	Remove Directory
3Bh	Change Current Directory
3Ch	Create File with Handle (volume label not supported)
3Dh	Open File with Handle
3Eh	Close File with Handle
3Fh	Read File
40h	Write File
41h	Delete File
42h	Move File Pointer
43:00h	Get File Attributes
44:00h	Get Device Data (for files only, no DOS devices supported)
47h	Get Current Directory
48h	Allocate Memory
49h	Free Allocated Memory
4Ah	Set Memory Block Size (stub)
4Ch	End Program
4Eh	Find First File
4Fh	Find Next File
50h	Set PSP Address
51h	Get PSP Address
56h	Rename File
57h	Get/Set File Date and Time
5Bh	Create New File
62h	Get PSP Address
66:01h	Get Global Code Page (returns US code 437)

Real Mode Memory Available in the First Megabyte



Using *unDOS* with our Real Mode Bootloader, it is possible for a real mode application to utilize memory in what is usually called the Upper Memory Area (UMA) from 640k (A0000h) to 1024k (100000h). *unDOS* is not limited to 640k the way DOS is. However, the UMA is generally used by ROM BIOS drivers and video display buffers as well as for Shadow ROM. Hence, the user must be cautious when utilizing memory in UMA for a real mode application.

The amount of memory available for your application will depend on the hardware and BIOS configuration of your system. For PC targets you can use Microsoft's Diagnostic utility (MSD.EXE) to determine what portions of your memory space are being used for ROM BIOS's etc. For other targets, the user must determine what memory is available based on the hardware configuration.

These notes are intended to be used as a planning tool to determine how much memory might be available to your real mode application when running under *unDOS* with its real mode loader.

For a new project, consider using the Turbo186 technology from VAutomation which supports a 16MB address space in real mode.

Below is a discussion of the various memory areas and their availability.

0 to 3FFH:

Used for Interrupt Vector Table, never available.

400H to 7FFH:

BIOS data area, used by BIOS and *unDOS*.

800H to 9FFFFH:

Conventional memory available for application.

A0000H to AFFFFH:

EGA/VGA video display buffer. Not available if an EGA/VGA/SVGA card is installed in your system and you are using it. Is available if VGA/EGA video modes are not used.

B0000H to BFFFFH:

Used for text video and CGA video display. Many EGA/VGA cards will use this memory for multiple text video pages. Not safe to use this memory if text video or CGA display modes are used. Otherwise, this area may be available for use by the application.

C0000H to C7FFFH:

Generally not available. The display card BIOS is located here and other extended BIOS's may be located here for disk drives etc. If no extended BIOS is present in system or programmer knows that no BIOS is present in this region then this area is available for application usage.

C8000H to DFFFFH:

This area is usually available for use by the application. If EMS is used then the 64k EMS page buffer is usually placed in this area. It is also possible that some devices may place their extended BIOS code or data areas in this region although this is not common. This area should be used with care.

E0000H to EFFFFH:

This area is usually available for use by application. On some systems this area may contain system BIOS and on others this may be used to Shadow the system BIOS. We suggest that the user turn off BIOS ROM Shadowing features to maximize the memory available.

F0000H to F7FFFH:

This area is often available for use. On many systems this area contains a duplicate copy of the system BIOS and is not actually required by the system BIOS. The best way to verify this is to compare this area with the system BIOS area starting at F8000H.

F8000H to FFFFFH:

Unavailable, used by system BIOS ROM.

Conclusion:

For a real mode application using *unDOS* with our real mode loader, as much as 192k of memory within the 1MB memory space becomes available in addition to the standard 638k (830k total). Additional memory up to 160k may be usable depending on what video modes, if any, are being used by the application. This presents a potentially large improvement (up to 990k total) in memory availability when compared to the maximum of 638k available to a standard real mode DOS application. *unDOS* uses approximately 50k of this memory.

Here are some ways this memory can be used:

1. In the best case where no video is being used, you may compile and link normally to use the full 990k of contiguous memory.
2. When using only text video mode or CGA, then you will have about 894k of memory split into two large blocks. In this case, you may link a dummy assembly language module with “dw 8000h DUP (?)” type statements in it to allocate space for the video buffer. These segments should be ORG’ed at 0B0000h to cover the video display buffer and BIOS. Then, compile and link normally.
3. If you are putting your code into ROM, you can locate some of your segments in the available area(s) of upper memory, arranging them to use the memory as efficiently as possible.
4. If there is a large block of contiguous memory, you might consider making that your far heap space (*smx* allows setting the heap boundaries to be anywhere in memory).
5. If your available upper memory is somewhat fragmented, you may wish to store large data structures in the blocks. To do this, you could simply create far pointers using the MK_FP() macro and use those pointers to access the locations explicitly.
6. If the upper memory space is fragmented, you may still consider using it as a far heap. This approach will require some effort. If using *smx*, you would need to understand the structure of the far heap and make modifications to the heap initialization routines. One approach would be to mark the areas of unavailable memory as fixed, permanently allocated memory blocks in the far heap.

smx and *pmEasy* are registered trademarks of Micro Digital, Inc.

smxFile, *unDOS*, and *unWin* are trademarks of Micro Digital, Inc.

Other brand and product names used above are the properties of their respective owners.