VESA®

Video Electronics Standards Association

2150 North First Street, Suite 440
San Jose, CA 95131-2029

Phone: (408) 435-0333
FAX: (408) 435-8225

VESA BIOS EXTENSION (VBE)
Core Functions
Standard

Version: 2.0
Document Revision: 1.1

Ratification Date: November 18, 1994

Purpose

To standardize a modular, software interface to display and audio devices. The VBE interface is intended to simplify and encourage the development of applications that wish to use graphics, video, and audio devices without specific knowledge of the internal operation of the evolving target hardware.

Summary

The VBE standard defines a set of extensions to the VGA ROM BIOS services. These functions can be accessed under DOS through interrupt 10h, or be called directly by high performance 32-bit applications and operating systems other than DOS.

These extensions also provide a hardware-independent mechanism to obtain vendor information, and serve as an extensible foundation for OEMs and VESA to facilitate rapid software support of emerging hardware technology without sacrificing backwards compatibility.

Intellectual Property

Trademarks

All trademarks used in this document are property of their respective owners.

- VESA, VBE, VESA DDC, VBE/AI, VBE/PM, and VBE/DDC are trademarks of Video Electronics Standards Association.
- MS-DOS and Windows are trademarks of Microsoft , Inc.
- IBM, VGA, EGA, CGA, and MDA are trademarks of International Business Machines
- RAMDAC is a trademark of Brooktree Corp.
- Hercules is a trademark of Hercules Computer Technology, Inc.

Patents

VESA proposal and standards documents are adopted by the Video Electronics Standards Association without regard to whether their adoption may involve patents on articles, materials, or processes. Such adoption does not assume any liability to any patent owner, nor does it assume any obligation whatever to parties adopting the proposal or standards document.

Support for this Specification

Clarifications and application notes to support this standard will be published as the need arises. To obtain the latest standard and support documentation, contact VESA.

If you have a product which incorporates VBE, you should ask the company that manufactured your product for assistance. If you are a display or controller manufacturer, VESA can assist you with any clarification you may require. All comments or reported errors should be submitted in writing to VESA using one of the following mechanisms:

World Wide Web:	www.vesa.org	**Mail to:**
E-mail:	techsupport@vesa.org	Video Electronics Standards Association
Fax:	408-435-8225	2150 North First Street, Suite 440
Voice:	408-435-0333	San Jose, California 95131-2029

SSC/VBE Workgroup Members

Any industry standard requires input from many sources. The people listed below were members of the VBE Workgroup of the Software Standards Committee (SSC) which was responsible for combining all of the industry input into this proposal:

CHAIRMAN

Kevin Gillett, S-MOS Systems, Inc.,
past chairman, Rod Dewell , Excalibur Solutions

MEMBERS

David Penley, AT&T Global Information Solutions
Rebecca Nolan, Chips and Technologies, Inc.
Brad Haakenson, Cirrus Logic, Inc.
Joe Rickson, Logitech, Inc.
Aaron Leatherman, LSI Logic Corporation
Jake Richter, Panacea Inc.
Raluca Iovan, Phoenix Technologies Ltd.
Kendall Bennett, SciTech Software
Tom Ryan, SciTech Software
George Bystricky, S-MOS Systems, Inc.
Jason Li, Trident Microsystems, Inc.
Chris Tsang, ULSI Systems
Greg Moore, Video Electronics Standards Association
Andy Lambrecht, VLSI Technology, Inc.
Rex Wolfe, Western Digital Imaging

Table of Contents

1.0 Introduction

This document contains the VESA BIOS Extension (VBE) specification for standard software access to graphics display controllers which support resolutions, color depths, and frame buffer organizations beyond the VGA hardware standard. It is intended for use by both applications programmers and system software developers.

System software developers may use this document to supplement the System and INT 10h ROM BIOS functions to provide the VBE services. Application developers can use this document as a guide to programming all VBE compatible devices.

To understand the VBE specification, some knowledge of 80x86 assembly language and the VGA hardware registers may be required. However, the services described in this specification may be called from any high-level programming language that provides a mechanism for generating software interrupts with the 80x86 registers set to user-specified values.

In this specification, 'VBE' and 'VBE 2.0' are synonymous with 'VBE Core Functions version 2.0'.

1.1 Scope of the VBE Standard

The primary purpose of the VESA VBE is to provide standard software support for the many unique implementations of Super VGA (SVGA) graphics controllers on the PC platform that provide features beyond the original VGA hardware standard. This is to provide a feasible mechanism by which application developers can take advantage of this nonstandard hardware in graphics applications.

The VBE specification offers an extensible software foundation which allows it to evolve as display and audio devices evolve over time, without sacrificing backward software compatibility with older implementations. New application software should be able to work with older hardware, and application software that has already shipped should work correctly on new hardware devices.

VBE services provide standard access to all resolutions and color depths provided on the display controller, and report the availability and details of all supported configurations to the application as necessary.

VBE implementations facilitate the field support of audio and display hardware by providing the application software with the manufacturer's name and the product identification of the display hardware.

Since graphics controller services on the PC are typically implemented in ROM, the VBE services are defined so that they should be implemented within the standard VGA ROM. When ROM implementations of VBE are not possible, or when field software upgrades to the onboard ROM are necessary, the VBE implementation may be also offered as a device driver or DOS Terminate and Stay Resident (TSR) program.

The standard VBE functions may be supplemented by OEM's as necessary to support custom or proprietary functions unique to the manufacturer. This mechanism enables the OEM to establish functions that may be standard to the product line, or provide access to special hardware enhancements.

Although previous VBE standards assumed that the underlying graphics architecture was a VGA device, the display services described by VBE 2.0 can be implemented on any frame buffer oriented graphics device.

The majority of VBE services facilitate the setup and configuration of the hardware, allowing applications high performance, direct access to the configured device at runtime. To further improve the performance of flat frame buffer display devices in extended resolutions, VBE 2.0 provides new memory models that do not require the traditional frame buffer "banking" mechanisms.

VBE is expected to work on all 80x86 platforms, in real and protected modes.

Since some modern display devices are designed without any VGA support, two display controllers may be present in the system. One display controller would be used for VGA compatibility, and the other used for graphic extensions to the basic VGA modes, resolutions, and frame buffer models. Therefore, VBE must be able offer the application automatic access to the appropriate device based on the mode or resolution that is requested by the application.

Currently beyond the scope of the VBE specification is the handling of hardware configuration and installation issues. It was originally considered to become part of VBE 2.0, however we have deferred the issues to the Graphics Configuration Supplemental Specification. In addition, it is also possible for an OEM to define their own extensions using the OEM Supplemental Specification if required.

1.2 Backgrounder

The IBM VGA[1] has become a de facto standard in the PC graphics world. A multitude of different VGA offerings exist in the marketplace, each one providing BIOS or register compatibility with the IBM VGA. More and more of these VGA compatible products implement various supersets of the VGA standard. These extensions range from higher resolutions and more colors to improved performance and even some graphics processing capabilities. Intense competition has dramatically improved the price/performance ratio, to the benefit of the end user.

[1] IBM and VGA are trademarks of International Business Machines Corporation.

However, several serious problems face a software developer who intends to take advantage of these "Super VGA"[2] environments. Because there is no standard hardware implementation, the developer is faced with widely different Super VGA hardware architecture. Lacking a common software interface, designing applications for these environments is costly and technically difficult. Except for applications supported by OEM-specific display drivers, very few software packages can take advantage of the power and capabilities of Super VGA products.

The VBE standard was originally conceived to enable the development of applications that wished to take advantage of display resolutions and color depths beyond the VGA definition. The need for an application or software standard was recognized by the developers of graphic hardware to encourage the use and acceptance of their rapidly advancing product families. It became obvious that the majority of software application developers did not have the resources to develop and support custom device level software for the hundreds of display boards on the market. Therefore the rich new features of these display devices were not being used outside of the relatively small CAD market, and only then after considerable effort.

Indeed, the need for a standard for SVGA display adapters became so important that the VESA organization was formed to seek out a solution. The original VBE standard was devised and agreed upon by each of the active display controller manufacturers, and has since been adopted by DOS application developers to enable use of non-VGA extended display modes.

As time went along VBE 1.1 was created to add more video modes and increased logical line length/double buffering support. VBE 1.2 was created to add modes and also added high color RAMDAC support.

In the three years since VBE 1.2 was approved we have seen the standard become widely accepted and many successful programs have embraced VBE. However, it has become obvious that the need for a more robust and extensible standard exists. Early extensions to the VGA standard continued using all of the original VGA I/O ports and frame buffer address to communicate with the controller hardware. As we've seen, the supported resolutions and color depths have grown, intelligent controllers with BITBLT and LineDraw Functions have become common, and new flat frame buffer memory models have appeared along with display controllers that are not based on VGA in any way. VBE 2.0 and future extensions will support non-VGA based controllers with new functions for reading and writing the palette and for access to the flat frame buffer models.

---

[2] The term "Super VGA" is used in this document for a graphics display controller implementing any superset of the standard IBM VGA display adapter.

VBE 2.0, as designed, offers the extensibility and the robustness that was lacking in the previous specifications, while at the same time offering backwards compatibility.

In the future, we see the need for adding supplemental specifications for issues like Multimedia; Advanced Graphics Functions; and "Plug and Play" features.

2.0 VBE Overview

This chapter outlines the various features and limitations of the VBE standard.

2.1 VBE Features

- Standard application interface to Graphics Controllers (SVGA Devices).
- Standard method of identifying products and manufacturers.
- Provision for OEM extensions through Sub-function 14h.
- Simple protected mode interface.
- Extensible interface through supplemental specifications.

2.2 VBE Affected Devices

All frame buffer-based devices in the PC platform (with the exception of Hercules, Monochrome (MDA), CGA and EGA devices) are suitable for use within the VBE standard to enable access to the device by VBE-compliant applications.

2.3 Providing Vendor Information

The VGA specification does not provide a standard mechanism to determine what graphic device it is running on. Only by knowing OEM-specific features can an application determine the presence of a particular graphics controller or display board. This often involves reading and testing registers located at I/O addresses unique to each OEM. By not knowing what hardware an application is running on, few, if any, of the extended features of this hardware can be used.

The VESA BIOS Extension provides several functions to return information about the graphics environment. These functions return system level information as well as graphics mode specific details. Function 00h returns general system level information, including an OEM identification string. The function also returns a pointer to the supported VBE and OEM modes. Function 01h may be used by an application to obtain additional information about each supported mode. Function 03h returns the current VBE mode.

3.0 VBE Mode Numbers

Standard VGA mode numbers are 7 bits wide and presently range from 00h to 13h. OEMs have defined extended display modes in the range 14h to 7Fh. Values from 80h to FFh cannot be used, since VGA BIOS Function 00h (Set video mode) interprets bit 7 as a flag to clear or preserve display memory.

Due to the limitations of 7-bit mode numbers, the optional VBE mode numbers are 14 bits wide. To initialize a VBE mode, the mode number is passed in the BX register to VBE Function 02h (Set VBE mode).

The format of VBE mode numbers is as follows:

D0-D8	=	Mode number If D8 == 0, this is not a VESA defined mode If D8 == 1, this is a VESA defined mode
D9-D13	=	Reserved by VESA for future expansion (= 0)
D14	=	Linear/Flat Frame Buffer Select If D14 == 0, Use VGA Frame Buffer If D14 == 1, Use Linear/Flat Frame Buffer
D15	=	Preserve Display Memory Select If D15 == 0, Clear display memory If D14 == 1, Preserve display memory

Thus, VBE mode numbers begin at 100h. This mode numbering scheme implements standard 7-bit mode numbers for OEM-defined modes. Standard VGA modes may be initialized through VBE Function 02h (Set VBE mode) simply by placing the mode number in BL and clearing the upper byte (BH). 7-bit OEM-defined display modes may be initialized in the same way. Note that modes may only be set if the mode exists in the VideoModeList pointed to by the VideoModePTR returned in Function 00h. The exception to this requirement is the mode number 81FFh.

To date, VESA has defined one special 7-bit mode number, 6Ah, for the 800x600, 16-color, 4-plane graphics mode. The corresponding 15-bit mode number for this mode is 102h. The following VBE mode numbers have been defined:

GRAPHICS				TEXT			
15-bit mode number	7-bit mode number	Resolution	Colors number	15-bit mode number	7-bit mode	Columns	Rows
100h	-	640x400	256	108h	-	80	60
101h	-	640x480	256	109h	-	132	25
102h	6Ah	800x600	16	10Ah	-	132	43
103h	-	800x600	256	10Bh	-	132	50
104h	-	1024x768	16	10Ch	-	132	60
105h	-	1024x768	256				
106h	-	1280x1024	16				

107h - 1280x1024 256

GRAPHICS

15-bit mode number	7-bit mode number	Resolution	Colors
10Dh	-	320x200	32K (1:5:5:5:)
10Eh	-	320x200	64K (5:6:5)
10Fh	-	320x200	16.8M (8:8:8)
110h	-	640x480	32K (1:5:5:5:)
111h	-	640x480	64K (5:6:5)
112h	-	640x480	16.8M (8:8:8)
113h	-	800x600	32K (1:5:5:5:)
114h	-	800x600	64K (5:6:5)
115h	-	800x600	16.8M (8:8:8)
116h	-	1024x768	32K (1:5:5:5:)
117h	-	1024x768	64K (5:6:5)
118h	-	1024x768	16.8M (8:8:8)
119h	-	1280x1024	32K (1:5:5:5:)
11Ah	-	1280x1024	64K (5:6:5)
11Bh	-	1280x1024	16.8M (8:8:8)
81FFh		Special Mode (see below for details)	

Note: Starting with VBE version 2.0, VESA will no longer define new VESA mode numbers and it will not longer be mandatory to support these old mode numbers. However, it is highly recommended that BIOS implementations continue to support these mode numbers for compatibility with older software. VBE 2.0-aware applications should follow the guidelines in Appendix 5 - Application Programming Considerations - for setting a desired mode.

Note: Mode 81FFh is a special mode designed to preserve the current memory contents and give access to the entire video memory. This mode is especially useful for saving the entire video memory contents before going into a state that could lose the contents (e.g., set this mode to gain access to all video memory to save it before going into a volatile power down state). This mode is required because the entire video memory contents are not always accessible in every mode. It is recommended that this mode be packed pixel in format, and a ModeInfoBlock must be defined for it. However, it should not appear in the VideoModeList. Look in the ModeInfoBlock to determine if paging is required and how paging is supported if it is. Also note that there are no implied resolutions or timings associated with this mode.

Note: Future display resolutions will be defined by VESA display vendors. The color depths will not be specified and new mode numbers will not be assigned for these resolutions. For example, if the VESA display vendors define 1600x1200 as a VESA resolution, application developers should target their display resolution for 1600x1200 rather than choosing an arbitrary resolution like 1550x1190. The VBE implementation should be queried to get the available resolutions and color depths and the

application should be flexible enough to work with this list. Appendix 5 gives a detailed summary of the way an application should go about selecting and setting modes.

4.0 VBE Functions

This chapter describes in detail each of the functions defined by the VBE standard. VBE functions are called using the INT 10h interrupt vector, passing arguments in the 80X86 registers. The INT 10h interrupt handler first determines if a VBE function has been requested, and if so, processes that request. Otherwise control is passed to the standard VGA BIOS for completion.

All VBE functions are called with the AH register set to 4Fh to distinguish them from the standard VGA BIOS functions. The AL register is used to indicate which VBE function is to be performed. For supplemental or extended functionality the BL register is used when appropriate to indicate a specific sub-function.

Functions 00h-0Fh have been reserved for Standard VBE function numbers; Functions 10h-FFh are reserved for VBE Supplemental Specifications.

In addition to the INT 10h interface, a Protected Mode Interface is available and is described below.

4.1 VBE Return Status

The AX register is used to indicate the completion status upon return from VBE functions. If VBE support for the specified function is available, the 4Fh value passed in the AH register on entry is returned in the AL register. If the VBE function completed successfully, 00h is returned in the AH register. Otherwise the AH register is set to indicate the nature of the failure.

VBE RETURN STATUS

AL == 4Fh:	Function is supported
AL != 4Fh:	Function is not supported
AH == 00h:	Function call successful
AH == 01h:	Function call failed
AH == 02h:	Function is not supported in the current hardware configuration
AH == 03h:	Function call invalid in current video mode

Note: Applications should treat any non-zero value in the AH register as a general failure condition as later versions of the VBE may define additional error codes.

4.2 Protected Mode Considerations

VBE services may be called directly from 32-bit protected mode only.

For 32-bit protected mode, 2 selector/segment descriptors for 32-bit code and the data segment are needed. These will be allocated and initialized by the caller. The segment limit fields will be set to 64k. These selectors may either be in the GDT or LDT, but must be valid whenever the VBE is called in protected mode. The caller must supply a stack large enough for use by VBE and by potential interrupt handlers. The caller's stack will be active if or when interrupts are enabled in the VBE routine, since the VBE will not switch stacks when interrupts are enabled, including NMI interrupts. The 32-bit VBE interface requires a 32-bit stack.

If the memory location is zero, then only I/O mapped ports will be used so the application does not need to do anything special. This should be the default case for ALL cards that have I/O mapped registers because it provides the best performance.

If the memory location is nonzero (there can be only one), the application will need to create a new 32-bit selector with the base address that points to the “physical” location specified with the specified limit.

When the application needs to call the 32-bit bank switch function, it must then load the ES selector with the value of the new selector that has been created. The bank switching code can then directly access its memory mapped registers as absolute offsets into the ES selector (i.e., mov [es:10],eax to put a value into the register at base+10).

It is up to the application code to save and restore the previous state of the ES selector if this is necessary (for example in flat model code).

When the VBE services are called, the current I/O permission bit map must allow access to the I/O ports that the VBE may need to access. This can be found in the Sub-Table (Ports and Memory) returned by VBE Function 0Ah.

To summarize, it is the responsibility of the calling to ensure to that it has the appropriate I/O and memory privileges, and a large enough stack and appropriate selectors allocated. It is also the responsibility of the calling application to preserve registers if necessary.

Applications must use the same registers for the Function 05h and Function 09h protected mode interface that they would use in a real mode call. This includes the AX register.

Function 07h protected mode calls have a different format.

AX	=	4F07h	
BL	=	00h	Set Display CRTC Start
	=	80h	Set Display CRTC Start during Vertical Retrace
CX	=		Bits 0-15 of display start address
DX	=		Bits 16-31 of display start address

The protected mode application must keep track of the color depth and scan line length to calculate the new start address. If a value that is out of range is programmed, unpredictable results will occur.

4.3 Function 00h - Return VBE Controller Information

This required function returns the capabilities of the display controller, the revision level of the VBE implementation, and vendor specific information to assist in supporting all display controllers in the field.

The purpose of this function is to provide information to the calling program about the general capabilities of the installed VBE software and hardware. This function fills an information block structure at the address specified by the caller. The VbeInfoBlock information block size is 256 bytes for VBE 1.x, and 512 bytes for VBE 2.0.

Input:	AX	= 4F00h	Return VBE Controller Information
	ES:DI	=	Pointer to buffer in which to place VbeInfoBlock structure (VbeSignature should be set to 'VBE2' when function is called to indicate VBE 2.0 information is desired and the information block is 512 bytes in size.)
Output:	AX	=	VBE Return Status

Note: All other registers are preserved.

The information block has the following structure:

```
VbeInfoBlock struc
VbeSignature        db  'VESA' ; VBE Signature
VbeVersion          dw  0200h       ; VBE Version
OemStringPtr        dd  ?           ; Pointer to OEM String
Capabilities        db  4 dup (?)   ; Capabilities of graphics controller
VideoModePtr    dd  ?           ; Pointer to VideoModeList
TotalMemory         dw  ?           ; Number of 64kb memory blocks
                                    ; Added for VBE 2.0
OemSoftwareRev      dw  ?           ; VBE implementation Software revision
OemVendorNamePtr    dd  ?           ; Pointer to Vendor Name String
OemProductNamePtr   dd  ?           ; Pointer to Product Name String
OemProductRevPtr    dd  ?           ; Pointer to Product Revision String
Reserved            db  222 dup (?) ; Reserved for VBE implementation scratch
                                    ;  area
OemData             db  256 dup (?) ; Data Area for OEM Strings
VbeInfoBlock ends
```

Note: All data in this structure is subject to change by the VBE implementation when VBE Function 00h is called. Therefore, it should not be used by the application to store data of any kind.

Description of the **VbeInfoBlock** structure fields:

The **VbeSignature** field is filled with the ASCII characters 'VESA' by the VBE implementation.VBE 2.0 applications should preset this field with the ASCII characters 'VBE2' to indicate to the VBE implementation that the VBE 2.0 extended information is desired, and the VbeInfoBlock is 512 bytes in size. Upon return from VBE Function 00h, this field should always be set to 'VESA' by the VBE implementation.

The **VbeVersion** is a BCD value which specifies what level of the VBE standard is implemented in the software. The higher byte specifies the major version number. The lower byte specifies the minor version number.

Note: The BCD value for VBE 2.0 is 0200h and the BCD value for VBE 1.2 is 0102h. In the past we have had some applications misinterpreting these BCD values. For example, BCD 0102h was interpreted as 1.02, which is incorrect.

The **OemStringPtr** is a Real Mode far pointer to a null terminated OEM-defined string. This string may be used to identify the graphics controller chip or OEM product family for hardware specific display drivers. There are no restrictions on the format of the string. This pointer may point into either ROM or RAM, depending on the specific implementation. VBE 2.0 BIOS implementations must place this string in the OemData area within the VbeInfoBlock if 'VBE2' is preset in the VbeSignature field on entry to Function 00h. This makes it possible to convert the RealMode address to an offset within the VbeInfoBlock for Protected mode applications.

Note: The length of the OEMString is not defined, but for space considerations, we recommend a string length of less than 256 bytes.

The **Capabilities** field indicates the support of specific features in the graphics environment. The bits are defined as follows:

D0	= 0	DAC is fixed width, with 6 bits per primary color
	= 1	DAC width is switchable to 8 bits per primary color
D1	= 0	Controller is VGA compatible
	= 1	Controller is not VGA compatible
D2	= 0	Normal RAMDAC operation
	= 1	When programming large blocks of information to the RAMDAC, use the blank bit in Function 09h.
D3-31	= Reserved	

BIOS Implementation Note: The DAC must always be restored to 6 bits per primary as default upon a mode set. If the DAC has been switched to 8 bits per primary, the mode set must restore the DAC to 6 bits per primary to ensure the application developer that he does not have to reset it.

Application Developer's Note: If a DAC is switchable, you can assume that the DAC will be restored to 6 bits per primary upon a mode set. For an application to use a DAC, the application program is responsible for setting the DAC to 8 bits per primary mode using Function 08h.

VGA compatibility is defined as supporting all standard IBM VGA modes, fonts and I/O ports; however, VGA compatibility doesn't guarantee that all modes which can be set are VGA compatible, or that the 8x14 font is available.

The need for D2 = 1 "program the RAMDAC using the blank bit in Function 09h" is for older style RAMDACs, where programming the RAM values during display time causes a "snow-like" effect on the screen. Newer style RAMDACs don't have this limitation and can easily be programmed at any time, but older RAMDACs require that they be blanked so as not to display the snow while values change during display time. This bit informs the software that it should make the function call with BL=80h rather than BL=00h to ensure the minimization of the "snow-like" effect.

The **VideoModePtr** points to a list of mode numbers for all display modes supported by the VBE implementation. Each mode number occupies one word (16 bits). The list of mode numbers is terminated by a -1 (0FFFFh). The mode numbers in this list represent all of the potentially supported modes by the display controller. Refer to Chapter 3 for a description of VESA VBE mode numbers. VBE 2.0 BIOS implementations must place this mode list in the Reserved area of the VbeInfoBlock or have it statically stored within the VBE implementation if 'VBE2' is preset in the VbeSignature field on entry to Function 00h.

Note: It is responsibility of the application to verify the actual availability of any mode returned by this function through the Return VBE Mode Information (VBE Function 01h) call. Some of the returned modes may not be available due to the actual amount of memory physically installed on the display board or due to the capabilities of the attached monitor.

Note: If a VideoModeList is found to contain no entries (starts with 0FFFFh), it can be assumed that the VBE implementation is a "stub" implementation where only Function 00h is supported for diagnostic or "Plug and Play" reasons. These stub implementations are not VBE 2.0 compliant and should only be implemented in cases where no space is available to implement the entire VBE.

The **TotalMemory** field indicates the maximum amount of memory physically installed and available to the frame buffer in 64KB units. (e.g. 256KB = 4, 512KB = 8) Not all video modes can address all this memory, see the ModeInfoBlock for detailed information about the addressable memory for a given mode.

The **OemSoftwareRev** field is a BCD value which specifies the OEM revision level of the VBE software. The higher byte specifies the major version number. The lower byte specifies the minor

version number. This field can be used to identify the OEM's VBE software release. This field is only filled in when 'VBE2' is preset in the VbeSignature field on entry to Function 00h.

The **OemVendorNamePtr** is a pointer to a null-terminated string containing the name of the vendor which produced the display controller board product. (This string may be contained in the VbeInfoBlock or the VBE implementation.) This field is only filled in when 'VBE2' is preset in the VbeSignature field on entry to Function 00h. (**Note:** the length of the strings OemProductRev, OemProductName and OemVendorName (including terminators) summed, must fit within a 256 byte buffer; this is to allow for return in the OemData field if necessary.)

The **OemProductNamePtr** is a pointer to a null-terminated string containing the product name of the display controller board. (This string may be contained in the VbeInfoBlock or the VBE implementation.) This field is only filled in when 'VBE2' is preset in the VbeSignature field on entry to Function 00h. (**Note:** the length of the strings OemProductRev, OemProductName and OemVendorName (including terminators) summed, must fit within a 256 byte buffer; this is to allow for return in the OemData field if necessary.)

The **OemProductRevPtr** is a pointer to a null-terminated string containing the revision or manufacturing level of the display controller board product. (This string may be contained in the VbeInfoBlock or the VBE implementation.) This field can be used to determine which production revision of the display controller board is installed. This field is only filled in when 'VBE2' is preset in the VbeSignature field on entry to Function 00h. (**Note:** the length of the strings OemProductRev, OemProductName and OemVendorName (including terminators) summed, must fit within a 256 byte buffer; this is to allow for return in the OemData field if necessary.)

The **Reserved** field is a space reserved for dynamically building the VideoModeList if necessary if the VideoModeList is not statically stored within the VBE implementation. This field should not be used for anything else, and may be reassigned in the future. Application software should not assume that information in this field is valid.

The **OemData** field is a 256 byte data area that is used to return OEM information returned by VBE Function 00h when 'VBE2' is preset in the VbeSignature field. The OemVendorName string, OemProductName string and OemProductRev string are copied into this area by the VBE implementation. This area will only be used by VBE implementations 2.0 and above when 'VBE2' is preset in the VbeSignature field.

4.4 Function 01h - Return VBE Mode Information

This required function returns extended information about a specific VBE display mode from the mode list returned by VBE Function 00h. This function fills the mode information block, ModeInfoBlock, structure with technical details on the requested mode. The ModeInfoBlock structure is provided by the application with a fixed size of 256 bytes.

Information can be obtained for all listed modes in the VideoModeList returned in Function 00h. If the requested mode cannot be used or is unavailable, a bit will be set in the ModeAttributes field to indicate that the mode is not supported in the current configuration.

Input:	AX	= 4F01h	Return VBE mode information
	CX	=	Mode number
	ES:DI	=	Pointer to ModeInfoBlock structure
Output:	AX	=	VBE Return Status

Note: All other registers are preserved.

The mode information block has the following structure:

ModeInfoBlock struc

```
; Mandatory information for all VBE revisions
ModeAttributes      dw  ?   ; mode attributes
WinAAttributes      db  ?   ; window A attributes
WinBAttributes  db  ?   ; window B attributes
WinGranularity  dw  ?   ; window granularity
WinSize             dw  ?   ; window size
WinASegment     dw  ?   ; window A start segment
WinBSegment     dw  ?   ; window B start segment
WinFuncPtr          dd  ?   ; pointer to window function
BytesPerScanLine    dw  ?   ; bytes per scan line

; Mandatory information for VBE 1.2 and above
XResolution         dw  ?   ; horizontal resolution in pixels or characters[3]
YResolution         dw  ?   ; vertical resolution in pixels or characters
XCharSize           db  ?   ; character cell width in pixels
YCharSize           db  ?   ; character cell height in pixels
NumberOfPlanes      db  ?   ; number of memory planes
```

[3]Pixels in graphics modes, characters in text modes.

```
BitsPerPixel                db      ?       ; bits per pixel
NumberOfBanks               db      ?       ; number of banks
MemoryModel         db      ?       ; memory model type
BankSize                    db      ?       ; bank size in KB
NumberOfImagePages  db      ?       ; number of images
Reserved                    db      1       ; reserved for page function

; Direct Color fields (required for direct/6 and YUV/7 memory models)
RedMaskSize                 db      ?       ; size of direct color red mask in bits
RedFieldPosition            db      ?       ; bit position of lsb of red mask
GreenMaskSize               db      ?       ; size of direct color green mask in bits
GreenFieldPosition          db      ?       ; bit position of lsb of green mask
BlueMaskSize        db      ?       ; size of direct color blue mask in bits
BlueFieldPosition           db      ?       ; bit position of lsb of blue mask
RsvdMaskSize        db      ?       ; size of direct color reserved mask in bits
RsvdFieldPosition           db      ?       ; bit position of lsb of reserved mask
DirectColorModeInfo db      ?       ; direct color mode attributes

; Mandatory information for VBE 2.0 and above
PhysBasePtr                 dd      ?       ; physical address for flat memory frame buffer
OffScreenMemOffset  dd      ?       ; pointer to start of off screen memory
OffScreenMemSize            dw      ?       ; amount of off screen memory in 1k units
Reserved                    db      206 dup (?) ; remainder of ModeInfoBlock
ModeInfoBlock ends
```

The ModeAttributes field describes certain important characteristics of the graphics mode.

The ModeAttributes field is defined as follows:

```
D0      =       Mode supported by hardware configuration
        0 =     Mode not supported in hardware
        1 =     Mode supported in hardware
D1      =       1 (Reserved)
D2      =       TTY Output functions supported by BIOS
        0 =     TTY Output functions not supported by BIOS
        1 =     TTY Output functions supported by BIOS
D3      =       Monochrome/color mode (see note below)
        0 =     Monochrome mode
        1 =     Color mode
D4      =       Mode type
        0 =     Text mode
        1 =     Graphics mode
D5      =       VGA compatible mode
        0 =     Yes
```

1 = No

D6 = VGA compatible windowed memory mode is available
0 = Yes
1 = No
D7 = Linear frame buffer mode is available
0 = No
1 = Yes
D8-D15 = Reserved

Bit D0 is set to indicate that this mode can be initialized in the present hardware configuration. This bit is reset to indicate the unavailability of a graphics mode if it requires a certain monitor type, more memory than is physically installed, etc.

Bit D1 was used by VBE 1.0 and 1.1 to indicate that the optional information following the BytesPerScanLine field were present in the data structure. This information became mandatory with VBE version 1.2 and above, so D1 is no longer used and should be set to 1. The Direct Color fields are valid only if the MemoryModel field is set to a 6 (Direct Color) or 7 (YUV).

Bit D2 indicates whether the video BIOS has support for output functions like TTY output, scroll, etc. in this mode. TTY support is recommended but not required for all extended text and graphic modes. If bit D2 is set to 1, then the INT 10h BIOS must support all of the standard output functions listed below.

All of the following TTY functions must be supported when this bit is set:

01	Set Cursor Size
02	Set Cursor Position
06	Scroll TTY window up or Blank Window
07	Scroll TTY window down or Blank Window
09	Write character and attribute at cursor position
0A	Write character only at cursor position
0E	Write character and advance cursor

Bit D3 is set to indicate color modes, and cleared for monochrome modes.

Bit D4 is set to indicate graphics modes, and cleared for text modes.

Note: Monochrome modes map their CRTC address at 3B4h. Color modes map their CRTC address at 3D4h. Monochrome modes have attributes in which only bit 3 (video) and bit 4 (intensity) of the attribute controller output are significant. Therefore, monochrome text modes have attributes of off, video, high intensity, blink, etc. Monochrome graphics modes are two plane graphics modes and have attributes of off, video, high intensity, and blink. Extended two color modes that have their CRTC address at 3D4h, are color modes with one bit per pixel and one plane. The standard VGA modes, 06h and 11h, would be classified as color modes, while the standard VGA modes 07h and 0Fh would be classified as monochrome modes.

Bit D5 is used to indicate if the mode is compatible with the VGA hardware registers and I/O ports. If this bit is set, then the mode is NOT VGA compatible and no assumptions should be made about the availability of any VGA registers. If clear, then the standard VGA I/O ports and frame buffer address defined in WinASegment and/or WinBSegment can be assumed.

Bit D6 is used to indicate if the mode provides Windowing or Banking of the frame buffer into the frame buffer memory region specified by WinASegment and WinBSegment. If set, then Windowing of the frame buffer is NOT possible. If clear, then the device is capable of mapping the frame buffer into the segment specified in WinASegment and/or WinBSegment. (This bit is used in conjunction with bit D7, see table following D7 for usage).

Bit D7 indicates the presence of a Linear Frame Buffer memory model. If this bit is set, the display controller can be put into a flat memory model by setting the mode (VBE Function 02h) with the Flat Memory Model bit set. (This bit is used in conjunction with bit D6, see following table for usage)

	D7	D6
Windowed frame buffer only	0	0
n/a	0	1
Both Windowed and Linear[4]	1	0
Linear frame buffer only	1	1

The **BytesPerScanLine** field specifies how many full bytes are in each logical scanline. Thelogical scanline could be equal to or larger than the displayed scanline.

The **WinAAttributes** and **WinBAttributes** describe the characteristics of the CPU windowing scheme such as whether the windows exist and are read/writeable, as follows:

D0 = Relocatable window(s) supported
0 = Single non-relocatable window only
1 = Relocatable window(s) are supported
D1 = Window readable
0 = Window is not readable
1 = Window is readable
D2 = Window writeable
0 = Window is not writeable
1 = Window is writeable
D3-D7 = Reserved

[4]Use D14 of the Mode Number to select the Linear Buffer on a mode set (Function 02h).

Even if windowing is not supported (bit D0 = 0 for both Window A and Window B), then an application can assume that the display memory buffer resides at the location specified by WinASegment and/or WinBSegment.

WinGranularity specifies the smallest boundary, in KB, on which the window can be placed in the frame buffer memory. The value of this field is undefined if Bit D0 of the appropriate WinAttributes field is not set.

WinSize specifies the size of the window in KB.

WinASegment and **WinBSegment** address specify the segment addresses where the windows are located in the CPU address space.

WinFuncPtr specifies the segment:offset of the VBE memory windowing function. The windowing function can be invoked either through VBE Function 05h, or by calling the function directly. A direct call will provide faster access to the hardware paging registers than using VBE Function 05h, and is intended to be used by high performance applications. If this field is NULL, then VBE Function 05h must be used to set the memory window when paging is supported. This direct call method uses the same parameters as VBE Function 05h including AX and for VBE 2.0 implementations will return the correct Return Status. VBE 1.2 implementations and earlier, did not require the Return Status information to be returned. For more information on the direct call method, see the notes in VBE Function 05h and the sample code in Appendix 5.

The **XResolution** and **YResolution** specify the width and height in pixel elements or characters for this display mode. In graphics modes, these fields indicate the number of horizontal and vertical pixels that may be displayed. In text modes, these fields indicate the number of horizontal and vertical character positions. The number of pixel positions for text modes may be calculated by multiplying the returned XResolution and YResolution values by the character cell width and height indicated in the XCharSize and YCharSize fields described below.

The **XCharSize** and **YCharSize** specify the size of the character cell in pixels. This value is not zero based (e.g. XCharSize for Mode 3 using the 9 point font will have a value of 9).

The **NumberOfPlanes** field specifies the number of memory planes available to software in that mode. For standard 16-color VGA graphics, this would be set to 4. For standard packed pixel modes, the field would be set to 1. For 256-color non-chain-4 modes, where you need to do banking to address all pixels, this value should be set to the number of banks required to get to all the pixels (typically this would be 4 or 8).

The **BitsPerPixel** field specifies the total number of bits allocated to one pixel. For example, a standard VGA 4 Plane 16-color graphics mode would have a 4 in this field and a packed pixel 256-color graphics mode would specify 8 in this field. The number of bits per pixel per plane can normally be derived by dividing the BitsPerPixel field by the NumberOfPlanes field.

The **MemoryModel** field specifies the general type of memory organization used in this mode. The following models have been defined:

00h	=	Text mode
01h	=	CGA graphics
02h	=	Hercules graphics
03h	=	Planar
04h	=	Packed pixel
05h	=	Non-chain 4, 256 color
06h	=	Direct Color
07h	=	YUV
08h-0Fh	=	Reserved, to be defined by VESA
10h-FFh	=	To be defined by OEM

VBE Version 1.1 and earlier defined Direct Color graphics modes with pixel formats 1:5:5:5, 8:8:8, and 8:8:8:8 as a Packed Pixel model with 16, 24, and 32-bits per pixel, respectively. In VBE Version 1.2 and later, the Direct Color modes use the Direct Color memory model and use the MaskSize and FieldPosition fields of the ModeInfoBlock to describe the pixel format. BitsPerPixel is always defined to be the total memory size of the pixel, in bits.

NumberOfBanks. This is the number of banks in which the scan lines are grouped. The quotient from dividing the scan line number by the number of banks is the bank that contains the scan line and the remainder is the scan line number within the bank. For example, CGA graphics modes have two banks and Hercules graphics mode has four banks. For modes that don't have scanline banks (such as VGA modes 0Dh-13h), this field should be set to 1.

The **BankSize** field specifies the size of a bank (group of scan lines) in units of 1 KB. For CGA and Hercules graphics modes this is 8, as each bank is 8192 bytes in length. For modes that do not have scanline banks (such as VGA modes 0Dh-13h), this field should be set to 0.

The **NumberOfImagePages** field specifies the "total number minus one (-1)"of complete display images that will fit into the frame buffer memory. The application may load more than one image into the frame buffer memory if this field is non-zero, and move the display window within each of those pages. This should only be used for determining the additional display pages which are available to the application; to determine the available off screen memory, use the OffScreenMemOffset and OffScreenMemSize information.

Note: If the ModeInfoBlock is for an IBM Standard VGA mode and the NumberOfImagePages field contains more pages than would be found in a 256KB implementation, the TTY support described in the ModeAttributes must be accurate. i.e., if the TTY functions are claimed to be supported, they must be supported in all pages, not just the pages normally found in the 256KB implementation.

The **Reserved** field has been defined to support a future VBE feature and will always be set to one in this version.

The **RedMaskSize, GreenMaskSize, BlueMaskSize, and RsvdMaskSize** fields define the size, in bits, of the red, green, and blue components of a direct color pixel. A bit mask can be constructed from the MaskSize fields using simple shift arithmetic. For example, the MaskSize values for a Direct Color 5:6:5 mode would be 5, 6, 5, and 0, for the red, green, blue, and reserved fields, respectively. Note that in the YUV MemoryModel, the red field is used for V, the green field is used for Y, and the blue field is used for U. The MaskSize fields should be set to 0 in modes using a memory model that does not have pixels with component fields.

The **RedFieldPosition, GreenFieldPosition, BlueFieldPosition, and RsvdFieldPosition** fields define the bit position within the direct color pixel or YUV pixel of the least significant bit of the respective color component. A color value can be aligned with its pixel field by shifting the value left by the FieldPosition. For example, the FieldPosition values for a Direct Color 5:6:5 mode would be 11, 5, 0, and 0, for the red, green, blue, and reserved fields, respectively. Note that in the YUV MemoryModel, the red field is used for V, the green field is used for Y, and the blue field is used for U. The FieldPosition fields should be set to 0 in modes using a memory model that does not have pixels with component fields.

The **DirectColorModeInfo** field describes important characteristics of direct color modes. Bit D0 specifies whether the color ramp of the DAC is fixed or programmable. If the color ramp is fixed, then it can not be changed. If the color ramp is programmable, it is assumed that the red, green, and blue lookup tables can be loaded by using VBE Function 09h. Bit D1 specifies whether the bits in the Rsvd field of the direct color pixel can be used by the application or are reserved, and thus unusable.

D0	=	Color ramp is fixed/programmable
	0 =	Color ramp is fixed
	1 =	Color ramp is programmable
D1	=	Bits in Rsvd field are usable/reserved
	0 =	Bits in Rsvd field are reserved
	1 =	Bits in Rsvd field are usable by the application

The **PhysBasePtr** is a 32-bit physical address of the start of frame buffer memory when the controller is in flat frame buffer memory mode. If this mode is not available, then this field will be zero.

The **OffScreenMemOffset** is a 32-bit offset from the start of the frame buffer memory. Extra off-screen memory that is needed by the controller may be located either before or after this off screen memory, be sure to check OffScreenMemSize to determine the amount of off-screen memory which is available to the application.

The **OffScreenMemSize** contains the amount of available, contiguous off-screen memory in 1k units, which can be used by the application.

Note: Version 1.1 and later VBE will zero out all unused fields in the Mode Information Block, always returning exactly 256 bytes. This facilitates upward compatibility with future versions of the standard, as any newly added fields will be designed such that values of zero will indicate nominal defaults or non-implementation of optional features. (For example, a field containing a bit-mask of extended capabilities would reflect the absence of all such capabilities.) Applications that wish to be backwards compatible to Version 1.0 VBE should pre-initialize the 256 byte buffer before calling the Return VBE Mode Information function.

4.5 Function 02h - Set VBE Mode

This required function initializes the controller and sets a VBE mode. The format of VESA VBE mode numbers is described earlier in this document. If the mode cannot be set, the BIOS should leave the graphics environment unchanged and return a failure error code.

Input:	AX	= 4F02h		Set VBE Mode
	BX	=		Desired Mode to set
		D0-D8=		Mode number
		D9-D13 =		Reserved (must be 0)
		D14	= 0	Use windowed frame buffer model
			= 1	Use linear/flat frame buffer model
		D15	= 0	Clear display memory
			= 1	Don't clear display memory

Output: AX = VBE Return Status

Note: All other registers are preserved.

If the requested mode number is not available, then the call will fail, returning AH=01h to indicate the failure to the application.

If bit D14 is set, the mode will be initialized for use with a flat frame buffer model. The base address of the frame buffer can be determined from the extended mode information returned by VBE Function 01h. If D14 is set, and a linear frame buffer model is not available then the call will fail.

If bit D15 is not set, all reported image pages, based on Function 00h returned information NumberOfImagePages, will be cleared to 00h in graphics mode, and 20 07 in text mode. Memory over and above the reported image pages will not be changed. If bit D15 is set, then the contents of the frame buffer after the mode change is undefined. Note, the 1-byte mode numbers used in Function 00h of an IBM VGA compatible BIOS use D7 to signify the same thing as D15 does in this function. If function call D7 is set and the application assumes it is similar to the IBM compatible mode set using VBE Function 02h, the implementation will fail. VBE aware applications must use the memory clear bit in D15.

Note: VBE BIOS 2.0 implementations should also update the BIOS Data Area 40:87 memory clear bit so that VBE Function 03h can return this flag. VBE BIOS 1.2 and earlier implementations ignore the memory clear bit.

Note: This call should not set modes not listed in the list of supported modes. In addition all modes (including IBM standard VGA modes), if listed as supported, must have ModeInfoBlock structures associated with them. Required ModeInfoBlock values for the IBM Standard Modes are listed in Appendix 2.

4.6 Function 03h - Return Current VBE Mode

This required function returns the current VBE mode. The format of VBE mode numbers is described earlier in this document.

Input:	AX	= 4F03h		Return current VBE Mode
Output:	AX	=		VBE Return Status
	BX	=		Current VBE mode
		D0-D13 =		Mode number
		D14	= 0	Windowed frame buffer model
			= 1	Linear/flat frame buffer model
		D15	= 0	Memory cleared at last mode set
			= 1	Memory not cleared at last mode set

Note: All other registers are preserved.

Version 1.x Note: In a standard VGA BIOS, Function 0Fh (Read current video state) returns the current graphics mode in the AL register. In D7 of AL, it also returns the status of the memory clear bit (D7 of 40:87). This bit is set if the mode was set without clearing memory. In this VBE function, the memory clear bit will not be returned in BX since the purpose of the function is to return the video mode only. If an application wants to obtain the memory clear bit, it should call the standard VGA BIOS Function 0Fh.

Version 2.x Note: Unlike version 1.x VBE implementations, the memory clear flag will be returned. The application should NOT call the standard VGA BIOS Function 0Fh if the mode was set with VBE Function 02h.

Note: The mode number returned must be the same mode number used in the VBE Function 02h mode set.

Note: This function is not guaranteed to return an accurate mode value if the mode set was not done with VBE Function 02h.

4.7 Function 04h - Save/Restore State

This required function provides a complete mechanism to save and restore the display controller hardware state. The functions are a superset of the three subfunctions under the standard VGA BIOS Function 1Ch (Save/restore state) which does not guarantee that the extended registers of the video device are saved or restored. The complete hardware state (except frame buffer memory) should be saveable/restorable by setting the requested states mask (in the CX register) to 000Fh.

Input:	AX	= 4F04h	Save/Restore State
	DL	= 00h	Return Save/Restore State buffer size
		= 01h	Save state
		= 02h	Restore state
	CX	=	Requested states
		D0=	Save/Restore controller hardware state
		D1=	Save/Restore BIOS data state
		D2=	Save/Restore DAC state
		D3=	Save/Restore Register state
	ES:BX	=	Pointer to buffer (if DL <> 00h)
Output:	AX	=	VBE Return Status
	BX	=	Number of 64-byte blocks to hold the state buffer (if DL=00h)

Note: All other registers are preserved.

4.8 Function 05h - Display Window Control

This required function sets or gets the position of the specified display window or page in the frame buffer memory by adjusting the necessary hardware paging registers. To use this function properly, the software should first use VBE Function 01h (Return VBE Mode information) to determine the size, location and granularity of the windows.

For performance reasons, it may be more efficient to call this function directly, without incurring the INT 10h overhead. VBE Function 01h returns the segment:offset of this windowing function that may be called directly for this reason. Note that a different entry point may be returned based upon the selected mode. Therefore, it is necessary to retrieve this segment:offset specifically for each desired mode.

Input:	AX	= 4F05h	VBE Display Window Control
	BH	= 00h	Set memory window
		= 01h	Get memory window
	BL	=	Window number
		= 00h	Window A
		= 01h	Window B
	DX	=	Window number in video memory in window granularity units (Set Memory Window only)

Output:	AX	=	VBE Return Status
	DX	=	Window number in window granularity units (Get Memory Window only)

Note: In VBE 1.2 implementations, the direct far call version returns no Return Status information to the application. Also, in the far call version, the AX and DX registers will be destroyed. Therefore if AX and/or DX must be preserved, the application must do so prior to making the far call. The application must still load the input arguments in BH, BL, and DX (for Set Window). In VBE 2.0 implementations, the BIOS will return the correct Return Status, and therefore the application must assume that AX and DX will be destroyed.

Application Developer's Note: This function is not intended for use in a linear frame buffer mode, if this function is requested, the function call will fail with the VBE Completion code AH=03h.

VBE BIOS Implementation Note: If this function is called while in a linear frame buffer memory model, this function must fail with completion code AH=03h.

4.9 Function 06h - Set/Get Logical Scan Line Length

This required function sets or gets the length of a logical scan line. This allows an application to set up a logical display memory buffer that is wider than the displayed area. VBE Function 07h (Set/Get Display Start) then allows the application to set the starting position that is to be displayed.

Input:	AX	= 4F06h	VBE Set/Get Logical Scan Line Length
	BL	= 00h	Set Scan Line Length in Pixels
		= 01h	Get Scan Line Length
		= 02h	Set Scan Line Length in Bytes
		= 03h	Get Maximum Scan Line Length
	CX	=	If BL=00h Desired Width in Pixels If BL=02h Desired Width in Bytes (Ignored for Get Functions
Output:	AX	=	VBE Return Status
	BX	=	Bytes Per Scan Line
	CX	=	Actual Pixels Per Scan Line (truncated to nearest complete pixel)
	DX	=	Maximum Number of Scan Lines

Note: The desired width in pixels or bytes may not be achievable because of hardware considerations. The next larger value will be selected that will accommodate the desired number of pixels or bytes, and the actual number of pixels will be returned in CX. BX returns a value that when added to a pointer into display memory will point to the next scan line. For example, in VGA mode 13h this would be 320, but in mode 12h this would be 80. DX returns the number of logical scan lines based upon the new scan line length and the total memory installed and usable in this display mode.

Note: This function is also valid in VBE supported text modes. In VBE supported text modes the application should convert the character line length to pixel line length by getting the current character cell width through the XCharSize field returned in ModeInfoBlock, multiplying that times the desired number of characters per line, and passing that value in the CX register. In addition, this function will only work if the line length is specified in character granularity. i.e. in 8 dot modes only multiples of 8 will work. Any value which is not in character granularity will result in a function call failure.

Note: On a failure to set scan line length by setting a CX value too large, the function will fail with error code 02h.

Note: The value returned when BL=03h is the lesser of either the maximum line length that the hardware can support, or the longest scan line length that would support the number of lines in the current video mode.

4.10 Function 07h - Set/Get Display Start

This required function selects the pixel to be displayed in the upper left corner of the display. This function can be used to pan and scroll around logical screens that are larger than the displayed screen. This function can also be used to rapidly switch between two different displayed screens for double buffered animation effects.

Input:	AX	= 4F07h	VBE Set/Get Display Start Control
	BH	= 00h	Reserved and must be 00h
	BL	= 00h	Set Display Start
		= 01h	Get Display Start
		= 80h	Set Display Start during Vertical Retrace
	CX	=	First Displayed Pixel In Scan Line (Set Display Start only)
	DX	=	First Displayed Scan Line (Set Display Start only)
Output:	AX	=	VBE Return Status
	BH	=	00h Reserved and will be 0 (Get Display Start only)
	CX	=	First Displayed Pixel In Scan Line (Get Display Start only)
	DX	=	First Displayed Scan Line (Get Display Start only)

Note: This function is also valid in text modes. To use this function in text mode, the application should convert the character coordinates to pixel coordinates by using XCharSize and YCharSize returned in the ModeInfoBlock. If the requested Display Start coordinates do not allow for a full page of video memory or the hardware does not support memory wrapping, the Function call should fail and no changes should be made. As a general case, if a requested Display Start is not available, fail the Function call and make no changes.

Note: CX and DX, for both input and output values, will be zero-based.

4.11 Function 08h - Set/Get DAC Palette Format

This required function manipulates the operating mode or format of the DAC palette. Some DACs are configurable to provide 6 bits, 8 bits, or more of color definition per red, green, and blue primary colors. The DAC palette width is assumed to be reset to the standard VGA value of 6 bits per primary color during any mode set.

Input:	AX	= 4F08h	VBE Set/Get Palette Format
	BL	= 00h	Set DAC Palette Format
		= 01h	Get DAC Palette Format
	BH	=	Desired bits of color per primary (Set DAC Palette Format only)
Output:	AX	=	VBE Return Status
	BH	=	Current number of bits of color per primary

An application can determine if DAC switching is available by querying Bit D0 of the Capabilities field of the VbeInfoBlock structure returned by VBE Function 00h (Return Controller Information). The application can then attempt to set the DAC palette width to the desired value. If the display controller hardware is not capable of selecting the requested palette width, then the next lower value that the hardware is capable of will be selected. The resulting palette width is returned.

This function will return failure code AH=03h if called in a direct color or YUV mode.

4.12 Function 09h - Set/Get Palette Data

This required function is very important for RAMDAC's which are larger than a standard VGA RAMDAC. The standard INT 10h BIOS Palette function calls assume standard VGA ports and VGA palette widths. This function offers a palette interface that is independent of the VGA assumptions.

Input:	AX	= 4F09h	VBE Load/Unload Palette Data
	BL	= 00h	Set Palette Data
		= 01h	Get Palette Data
		= 02h	Set Secondary Palette Data
		= 03h	Get Secondary Palette Data
		= 80h	Set Palette Data during Vertical Retrace with Blank Bit on
	CX	=	Number of palette registers to update (to a maximum of 256)
	DX	=	First of the palette registers to update (start)
	ES:DI	=	Table of palette values (see below for format)
Output:	AX	=	VBE Return Status

Format of Palette Values:Alignment byte, Red byte, Green byte, Blue byte

Note: The need for BL= 80h is for older style RAMDAC's where programming the RAM values during display time causes a "snow-like" effect on the screen. Newer style RAMDAC's don't have this limitation and can easily be programmed at any time, but older RAMDAC's require that they be programmed during a non-display time only to stop the snow like effect seen when changing the DAC values. When this is requested the VBE implementation will program the DAC with blanking on. Check D2 of the Capabilities field returned by VBE Function 00h to determine if 80h should be used instead of 00h.

Note: The need for the secondary palette is for anticipated future palette extensions, if a secondary palette does not exist in a implementation and these calls are made, the VBE implementation will return error code 02h.

Note: When in 6 bit mode, the format of the 6 bits is LSB, this is done for speed reasons, as the application can typically shift the data faster than the BIOS can.

Note: All application should assume the DAC is defaulted to 6 bit mode. The application is responsible for switching the DAC to higher color modes using Function 08h.

Note: Query VBE Function 08h to determine the RAMDAC width before loading a new palette.

4.13 Function 0Ah - Return VBE Protected Mode Interface

This required function call returns a pointer to a table that contains code for a 32-bit protected mode interface that can either be copied into local 32-bit memory space or can be executed from ROM providing the calling application sets all required selectors and I/O access correctly. This function returns a pointer (in real mode space) with offsets to the code fragments, and additionally returns an offset to a table which contains Non-VGA Port and Memory locations which an Application may have to have I/O access to.

Input:	AX	= 4F0Ah	VBE 2.0 Protected Mode Interface
	BL	= 00h	Return protected mode table

Output:	AX	=	Status
	ES	=	Real Mode Segment of Table
	DI	=	Offset of Table
	CX	=	Length of Table including protected mode code in bytes (for copying purposes)

The format of the table is as follows:

ES:DI + 00h	Word Offset in table of Protected mode code for Function 5 for Set Window Call
ES:DI + 02h	Word Offset in table of Protected mode code for Function 7 for set Display Start
ES:DI + 04h	Word Offset in table of Protected mode code for Function 9 for set Primary Palette data
ES:DI + 06h	Word Offset in table of Ports and Memory Locations that the application may need I/O privilege for. (Optional: if unsupported this must be 0000h) (See Sub-table for format)
ES:DI + ?	Variable remainder of Table including Code

The format of the Sub-Table (Ports and Memory locations)

Port, Port, ... , Port, Terminate Port List with FF FF, Memory locations (4 bytes), Length (2 bytes), Terminate Memory List with FF FF.

Example 1. For Port/Index combination 3DE/Fh and Memory locations DE800-DEA00h (length = 200h) the table would look like this:

```
DE 03 DF 03 FF FF 00 E8 0D 00 00 02 FF FF
```

Example 2. For only the ports it would look like:

```
DE 03 DF 03 FF FF FF FF
```

Example 3. For only the memory locations it would look like

```
FF FF 00 E8 0D 00 00 02 FF FF
```

Note:. All protected mode functions should end with a near RET (as opposed to FAR RET) to allow the application software to CALL the code from within the ROM.

Note: The Port and Memory location Sub-table does not include the Frame Buffer Memory location. The Frame Buffer Memory location is contained within the ModeInfoBlock returned by VBE Function 01h.

Note: The protected mode code is assembled for a 32-bit code segment, when copying it, the application must copy the code to a 32-bit code segment.

Note: It is the responsibility of the application to ensure that the selectors and segments are set up correctly.

If the memory location is zero, then only I/O mapped ports will be used so the application does not need to do anything special. This should be the default case for ALL cards that have I/O mapped registers because it provides the best performance.

If the memory location is nonzero (there can be only one), the application will need to create a new 32-bit selector with the base address that points to the "physical" location specified with the specified limit.

When the application needs to call the 32-bit bank switch function, it must then load the ES selector with the value of the new selector that has been created. The bank switching code can then directly access its memory mapped registers as absolute offsets into the ES selector (i.e., mov [es:10],eax to put a value into the register at base+10).

It is up to the application code to save and restore the previous state of the ES selector if this is necessary (for example in flat model code).

Note: Currently undefined registers may be destroyed with the exception of ESI, EBP, DS and SS.

Note: Applications must use the same registers for the Function 05h and Function 09h protected mode interface that it would use in a real mode call. This includes the AX register.

Note: Function 07h protected mode calls have a different format.

AX	=	4F07h	
BL	=	00h	Set Display CRTC Start
	=	80h	Set Display CRTC Start during Vertical Retrace
CX	=		Bits 0-15 of display start address
DX	=		Bits 16-31 of display start address

The protected mode application must keep track of the color depth and scan line length to calculate the new start address. If a value that is out of range is programmed, unpredictable results will occur.

Note: Refer to Section 4.2 for information on protected mode considerations.

5.0 VBE Supplemental Specifications

This chapter details VBE Supplemental Specifications.

5.1 Purpose of Supplemental Specifications

The VBE was originally designed to provide a device-independent interface between application software and SVGA hardware. In the last few years, the personal computing environment has grown much more complex and there have been numerous requests to provide interfaces similar to the VBE to service these new requirements. The VBE supplemental specification architecture provides a way to extend the basic VBE specification without making it too unwieldy or having to revise the VBE specification itself.

The supplemental specifications are implemented using VBE function numbers starting at AL=10h. This leaves the first sixteen functions available for eventual VBE growth. Individual calls for each supplemental specification are made through a subfunction number via the BL register. This function/subfunction architecture is compatible with the VBE and provides each VBE Supplemental Specification with 64 potential subfunctions. Subfunction 00h for each supplemental specification is reserved for a 'Return VBE Supplemental Specification Information' call. It is based on the VBE Function 00h and returns basic information on the VBE Supplemental Specification implementation.

5.2 Obtaining Supplemental VBE Function Numbers

VBE Supplemental Specifications can only be created by VESA committees. Once a need for a new software specification has been identified, the group working on it needs to contact the VESA Software Standards Committee (SSC) to discuss the requirements. The SSC will assign a function number and name to the supplemental specification. The name assigned to a supplemental specification will be in the form of 'VBE/???' where the '???' is a two or three letter acronym for its function. Two letter acronyms will be padded with FFh for the third letter. In some cases the committee that is working on the supplemental specification may have another name that they will use for promotional purposes, however the VBE/??? will continue to be the signature.

The VBE specification will be revised periodically to update the list of supplemental specifications. To obtain the actual specifications, contact the VESA office.

5.3 Required VBE Supplemental Specification Components

5.3.1 VBE Supplemental Specification Functions

All VBE Supplemental Specification functions are called with the AH register set to 4Fh to identify them as VBE function calls. The AL register is used to specify which VESA function the supplemental specification is using. The BL register will contain the subfunction number for the call being made, BH will contain the sub-subfunction number if necessary.

e.g., Input:	AX	= 4FXXh	VESA Supplemental VBE Specification ('XX' represents the function number for the supplemental specification)
	BL	= ?	Subfunction
	BH	= ?	Sub-subfunction

5.3.2 Return Status

All VBE Supplemental Specifications will use the VBE Completion codes as documented in Section 4.1 of the VBE specification.

5.3.3 Subfunction 00h - Return VBE Supplemental Specification Information

This subfunction returns the capabilities, revision level, and vendor specific information of the supplemental specification, and is a required function for any VBE 2.x Supplemental Specification.

The purpose of this subfunction is to provide information to the calling program about the general capabilities of the installed VBE software and hardware. This subfunction fills an information block structure at the address specified by the caller. The SupVBEInfoBlock information block size is 256 bytes.

Input:	AX	= 4FXXh	Return Supplemental VBE Specification Information ('XX' represents the function number for the supplemental specification)
	BL	= 00h	Subfunction '0'
	ES:DI	=	Pointer to buffer in which to place SupVBEInfoBlock structure
Output: AX	=		VBE Return Status

Other registers may be defined for input and output based upon the particular requirements of supplemental specifications

When writing supplemental functions, explicitly state which registers are preserved and which are destroyed. Refrain from preserving all registers as this tends to limit expandability in the future. An example of the note is:

Note: Currently undefined registers may be destroyed with the exception of SI,BP,DS and SS.
The information block has the following structure:

```
SupVbeInfoBlock struc
SupVbeSignature     db      'VBE/???'       ; Supplemental VBE Signature
SupVbeVersion       dw      ?               ; Supplemental VBE Version
SupVbeSubFunc       db      8 dup (?)       ; Bitfield of supported subfunctions
OemSoftwareRev      dw      ?               ; OEM Software revision
OemVendorNamePtr    dd      ?               ; Pointer to Vendor Name String
OemProductNamePtr   dd      ?               ; Pointer to Product Name String
OemProductRevPtr    dd      ?               ; Pointer to Product Revision String
OemStringPtr        dd      ?               ; Pointer to OEM String
Reserved            db      221 dup (?)     ; Reserved for description strings and future
                                            ; expansion
SupVbeInfoBlock ends
```

Note: All data in this structure is subject to change by the VBE implementation when any VBE Subfunction 00h is called. Therefore it should not be used by the application to store data of any kind.

Description of the **SupVbeInfoBlock** structure fields:

The **SupVbeSignature** field is filled with the ASCII characters 'VBE/' followed by the two or three letter acronym that represents the supplemental specification. This field is filled by the supplementary VBE implementation. In the event that the acronym is only two letters, the third letter must be filled with FFh.

The **SupVbeVersion** is a BCD value which specifies what level of the VBE supplementary specification is implemented in the software. The higher byte specifies the major version number. The lower byte specifies the minor version number.

Note: The BCD value for 2.0 is 0200h and the BCD value for 1.2 is 0102h. In the past we have had some applications misinterpreting these BCD values. For example, BCD 0102h was interpreted as 1.02, which is incorrect.

The **SupVbeSubFunc** is a bitfield that represents the subfunctions available for the supplementary specification. If the bit representing a particular subfunction is set, then that subfunction is supported. Subfunction '0' is represented by the LSB of the first byte and the other subfunctions follow. Only bits for subfunctions defined in the specification need to be set.

The **OemStringPtr** is a Real Mode far pointer to a null-terminated OEM-defined string. This string may used to identify the graphics controller chip or OEM product family for hardware specific display

drivers. There are no restrictions on the format of the string. This pointer may point into either ROM or RAM, depending on the specific implementation.

The **OemSoftwareRev** field is a BCD value which specifies the OEM revision level of the Supplemental Specification software. The higher byte specifies the major version number. The lower byte specifies the minor version number. This field can be used to identify the OEM's VBE software release.

The **OemVendorNamePtr** is a Real Mode far pointer to a null-terminated OEM-defined string containing the name of the vendor who produced the display controller board product.

The **OemProductNamePtr** is a Real Mode far pointer to a null-terminated OEM-defined string containing the product name of the display controller board.

The **OemProductRevPtr** is a Real Mode far pointer to a null-terminated OEM-defined string containing the revision or manufacturing level of the display controller board product. This field can be used to determine which production revision of the display controller board is installed.

5.4 Supplemental Specification Protected Mode Guidelines

VBE Supplemental Specifications may wish to incorporate 32-bit protected mode interfaces based upon the 32-bit protected mode interface in VBE 2.0. The guidelines for this are simple.

Input:	AX	= 4FXXh	Supplemental Specification Function Number
	BL	= XX	Return 32-bit protected mode interface table

Output:	AX	=	Status
	ES	=	Real Mode Segment of Table
	DI	=	Offset of Table
	CX	=	Length of Table including protected mode code (for copying purposes)

The format of the table should be as follows:

ES:DI + 00h	Word Offset in table of Protected mode code for first function
ES:DI + (n*2)	Word Offset in table of Protected mode code for nth function
ES:DI + ?	Word Offset in table of Ports and Memory Locations that the application may need I/O privilege for (Optional: if unsupported this must be 0000h) (See Sub-table for format)
ES:DI + ?	Variable remainder of Table including Code

The format of the Sub-Table (Ports and Memory locations)

> Port, Port, ... , Port, Terminate Port List with FF FF, Memory locations (4 bytes), Length (2 bytes), Terminate Memory List with FF FF.

Example 1. For Port/Index combination 3DE/Fh and Memory locations DE800-DEA00h (length = 200h) the table would look like this:

```
DE 03 DF 03 FF FF 00 E8 0D 00 00 02 FF FF
```

Example 2. For only the ports it would look like:

```
DE 03 DF 03 FF FF FF FF
```

Example 3. For only the memory locations it would look like

```
FF FF 00 E8 0D 00 00 02 FF FF
```

Note:. All protected mode functions should end with a near RET (as opposed to FAR RET) to allow the application software to CALL the code from within the ROM.

Note: The protected mode code should be assembled for a 32-bit code segment, when copying it, the application must copy the code to a 32-bit code segment.

Note: It is the responsibility of the application to ensure that the selectors and segments are set up correctly.

Note: Currently undefined registers may be destroyed with the exception of ESI, EBP, DS and SS.

In developing a supplemental specification, ensure that both the application developer and the VBE/XXX implementors are aware of which portion of the function is supported, i.e., if a function supports both a Get and a Set function, spell out which is supported, the Get, the Set or both.

5.5 Loading Supplemental Drivers

VBE Supplemental Specifications can be implemented in ROM, TSR programs or as device drivers. The specific requirements will vary depending on the individual supplementary specification. If there are any specific requirements, they should be detailed in the supplementary specification.

5.6 Implementation Questions

When developing a new supplemental specification, implementation questions whether they are covered in this guideline or not, should be referred to the VESA Software Standards Committee for clarification. The chairman of the SSC can be contacted through VESA office.

5.7 Known Supplemental Specifications

5.7.1 Function 10h - Power Management Extensions (PM)

This optional function controls the power state of the attached display device or monitor.
Refer to the VBE/PM Standard for specifics.

5.7.2 Function 11h - Flat Panel Interface Extensions (FP)

This proposed optional supplemental specification allows access to the special features incorporated in Flat Panel controllers. There is no reference specification at the time of this standard's approval. Contact the VESA office for more information.

5.7.3 Function 12h - Cursor Interface Extensions (CI)

This proposed optional function provides services for Hardware Cursors and Pointing Devices. At this time, this is in development. There is no reference specification at the time of this standard's approval, contact the VESA office for more information.

5.7.4 Function 13h - Audio Interface Extensions (AI)

This optional function provides standard Audio services.
Refer to the VBE/AI Standard for specifics.

5.7.5 Function 14h - OEM Extensions

This optional supplemental function provides OEM's with a code dispatch area that falls under the VESA 4Fh functions. An OEM may use this area at their own risk. VESA states no warranties or guarantees about the function calls contained within this area.

5.7.6 Function 15h - Display Data Channel (DDC)

This optional function provides a mechanism to extract data from attached display devices on the VESA communication channel. Refer to the VBE/DDC Standard for specifics.

5.7.7 Function 16h - Graphics System Configuration (GC)

This proposed supplemental function provides a mechanism for system level services to set up Monitor Timings, Linear Frame Buffer addresses etc., i.e. system level calls that applications should not deal with . There is no reference specification at the time of this standard's approval. Contact the VESA office for more information.

Appendix 1 - VBE Quick Reference

Input:	AX	= 4F00h	Return VBE Controller Information
	ES:DI	=	Pointer to buffer in which to place VbeInfoBlock structure (VbeSignature should be set to 'VBE2' when function is called to indicate VBE 2.0 information is desired and the information block is 512 bytes in size.)

Output: AX = VBE Return Status
Note: All other registers are preserved.

Input:	AX	= 4F01h	Return VBE mode information
	CX	=	Mode number
	ES:DI	=	Pointer to ModeInfoBlock structure

Output: AX = VBE Return Status
Note: All other registers are preserved.

Input:	AX	= 4F02h		Set VBE Mode
	BX	=		Desired Mode to set
		D0-D8=		Mode number
		D9-D13 =		Reserved (must be 0)
		D14	= 0	Use windowed frame buffer model
			= 1	Use linear/flat frame buffer model
		D15	= 0	Clear display memory
			= 1	Don't clear display memory

Output: AX = VBE Return Status
Note: All other registers are preserved.

Input:	AX	= 4F03h	Return current VBE Mode
Output: AX	=		VBE Return Status
	BX	=	Current VBE mode
		D0-D13 =	Mode number
		D14 = 0	Windowed frame buffer model
		= 1	Linear/flat frame buffer model
		D15 = 0	Memory cleared at last mode set
		= 1	Memory not cleared at last mode set

Note: All other registers are preserved.

Input:	AX	= 4F04h	Save/Restore State
	DL	= 00h	Return Save/Restore State buffer size
		= 01h	Save state
		= 02h	Restore state
	CX	=	Requested states
		D0=	Save/Restore controller hardware state
		D1=	Save/Restore BIOS data state
		D2=	Save/Restore DAC state
		D3=	Save/Restore Register state
	ES:BX	=	Pointer to buffer (if DL <> 00h)
Output: AX	=		VBE Return Status
	BX	=	Number of 64-byte blocks to hold the state buffer (if DL=00h)

Note: All other registers are preserved.

Input:	AX	= 4F05h	VBE Display Window Control
	BH	= 00h	Set memory window
		= 01h	Get memory window
	BL	=	Window number
		= 00h	Window A
		= 01h	Window B
	DX	=	Window number in video memory in window granularity units (Set Memory Window only)
Output: AX	=		VBE Return Status
	DX	=	Window number in window granularity units

Input:	AX	= 4F06h	VBE Set/Get Logical Scan Line Length
	BL	= 00h	Set Scan Line Length in Pixels
		= 01h	Get Scan Line Length
		= 02h	Set Scan Line Length in Bytes
		= 03h	Get Maximum Scan Line Length
	CX	=	If BL=00h Desired Width in Pixels
			If BL=02h Desired Width in Bytes
			(Ignored for Get Functions)
Output: AX	=		VBE Return Status
	BX	=	Bytes Per Scan Line
	CX	=	Actual Pixels Per Scan Line
			(truncated to nearest complete pixel)
	DX	=	Maximum Number of Scan Lines

Input:	AX	= 4F07h	VBE Set/Get Display Start Control
	BH	= 00h	Reserved and must be 00h
	BL	= 00h	Set Display Start
		= 01h	Get Display Start
		= 80h	Set Display Start during Vertical Retrace
	CX	=	First Displayed Pixel In Scan Line
			(Set Display Start only)
	DX	=	First Displayed Scan Line (Set Display Start only)
Output: AX	=		VBE Return Status
	BH	=	00h Reserved and will be 0 (Get Display Start only)
	CX	=	First Displayed Pixel In Scan Line (Get Display Start only)
	DX	=	First Displayed Scan Line (Get Display Start only)

Input:	AX	= 4F08h	VBE Set/Get Palette Format
	BL	= 00h	Set DAC Palette Format
		= 01h	Get DAC Palette Format
	BH	=	Desired bits of color per primary
			(Set DAC Palette Format only)
Output: AX	=		VBE Return Status
	BH	=	Current number of bits of color per primary

Input:	AX	= 4F09h	VBE Load/Unload Palette Data
	BL	= 00h	Set Palette Data
		= 01h	Get Palette Data
		= 02h	Set Secondary Palette Data
		= 03h	Get Secondary Palette Data
		= 80h	Set Palette Data during Vertical Retrace with Blank Bit on
	CX	=	Number of palette registers to update
	DX	=	First palette register to update
	ES:DI	=	Table of palette values (see below for format)

Output: AX = VBE Return Status

Format of Palette Values: Alignment byte, Red byte, Green byte, Blue byte

---

Input:	AX	= 4F0Ah	VBE 2.0 Protected Mode Interface
	BL	= 00h	Return protected mode table
Output:	AX	=	Status
	ES	=	Real Mode Segment of Table
	DI	=	Offset of Table
	CX	=	Length of Table including protected mode code in bytes (for copying purposes)

Appendix 2 - VBE Data Structures

```
VbeInfoBlock struc
VbeSignature        db      'VESA'          ; VBE Signature
VbeVersion          dw      0200h           ; VBE Version
OemStringPtr        dd      ?               ; Pointer to OEM String
Capabilities        db      4 dup (?)       ; Capabilities of graphics controller
VideoModePtr    dd  ?           ; Pointer to VideoModeList
TotalMemory         dw      ?               ; Number of 64kb memory blocks
                                            ; Added for VBE 2.0
OemSoftwareRev      dw      ?               ; VBE implementation Software revision
OemVendorNamePtr    dd      ?               ; Pointer to Vendor Name String
OemProductNamePtr   dd      ?               ; Pointer to Product Name String
OemProductRevPtr    dd      ?               ; Pointer to Product Revision String
Reserved            db      222 dup (?)     ; Reserved for VBE implementation scratch
                                            ;  area
OemData             db      256 dup (?)     ; Data Area for OEM Strings
VbeInfoBlock ends
```

```
ModeInfoBlock struc
; Mandatory information for all VBE revisions
ModeAttributes      dw ?    ; mode attributes
WinAAttributes      db ?    ; window A attributes
WinBAttributes      db ?    ; window B attributes
WinGranularity      dw ?    ; window granularity
WinSize             dw ?    ; window size
WinASegment         dw ?    ; window A start segment
WinBSegment         dw ?    ; window B start segment
WinFuncPtr          dd ?    ; pointer to window function
BytesPerScanLine    dw ?    ; bytes per scan line
; Mandatory information for VBE 1.2 and above
XResolution         dw ?    ; horizontal resolution in pixels or characters
YResolution         dw ?    ; vertical resolution in pixels or characters
XCharSize           db ?    ; character cell width in pixels
YCharSize           db ?    ; character cell height in pixels
NumberOfPlanes      db ?    ; number of memory planes
BitsPerPixel        db ?    ; bits per pixel
NumberOfBanks       db ?    ; number of banks
MemoryModel         db ?    ; memory model type
BankSize            db ?    ; bank size in KB
NumberOfImagePages  db ?    ; number of images
Reserved            db 1    ; reserved for page function
; Direct Color fields (required for direct/6 and YUV/7 memory models)
RedMaskSize         db ?    ; size of direct color red mask in bits
RedFieldPosition    db ?    ; bit position of lsb of red mask
GreenMaskSize       db ?    ; size of direct color green mask in bits
GreenFieldPosition  db ?    ; bit position of lsb of green mask
BlueMaskSize        db ?    ; size of direct color blue mask in bits
BlueFieldPosition   db ?    ; bit position of lsb of blue mask
RsvdMaskSize        db ?    ; size of direct color reserved mask in bits
RsvdFieldPosition   db ?    ; bit position of lsb of reserved mask
DirectColorModeInfo db ?    ; direct color mode attributes
; Mandatory information for VBE 2.0 and above
PhysBasePtr         dd ?    ; physical address for flat memory frame buffer
OffScreenMemOffset  dd ?    ; pointer to start of off screen memory
OffScreenMemSize    dw ?    ; amount of off screen memory in 1k units
Reserved            db 206 dup (?) ; remainder of ModeInfoBlock
ModeInfoBlock ends
```

```
SupVbeInfoBlock struc
SupVbeSignature     db    'VBE/???'     ; Supplemental VBE Signature
SupVbeVersion       dw    ?             ; Supplemental VBE Version
SupVbeSubFunc       db    8 dup (?)     ; Bitfield of supported subfunctions
OemSoftwareRev      dw    ?             ; OEM Software revision
OemVendorNamePtr    dd    ?             ; Pointer to Vendor Name String
OemProductNamePtr   dd    ?             ; Pointer to Product Name String
OemProductRevPtr    dd    ?             ; Pointer to Product Revision String
OemStringPtr        dd    ?             ; Pointer to OEM String
Reserved            db    221 dup (?)   ; Reserved for description strings and future
                                        ;   expansion
SupVbeInfoBlock ends
```

Function 0Ah Table Formats

The format of the table is as follows:

ES:DI + 00h	Word Offset in table of Protected mode code for Function 5 for Set Window Call
ES:DI + 02h	Word Offset in table of Protected mode code for Function 7 for set Display Start
ES:DI + 04h	Word Offset in table of Protected mode code for Function 9 for set Primary Palette data
ES:DI + 06h	Word Offset in table of Ports and Memory Locations that the application may need I/O privilege for (Optional: if unsupported this must be 0000h) (See Sub-table for format)
ES:DI + ?	Variable remainder of Table including Code

The format of the Sub-Table (Ports and Memory locations)

Port, Port, ... , Port, Terminate Port List with FF FF, Memory locations (4 bytes), Length (2 bytes), Terminate Memory List with FF FF.

Required ModeInfoBlock Information for VGA Standard Modes

The VGA Standard modes are not required to be supported by the VESA set mode Function 02h, however, if 4F02h can set the mode, then the mode must have a ModeInfoBlock structure associated with it. These are the required ModeInfoBlock formats for VGA standard modes if they are supported.

Note: The NumberOfImagePages field is defined as
[Available Memory/(YResolution*BytesPerScanLine)] -1

The Available Memory has been calculated for a 256KB VGA implementation; implementations >256KB may have more memory available to an IBM Standard Mode and therefore this number may vary. All other values are fixed.

```
; IBM Mode 00 VBE Support
; Text 40x25

        DW      0000Eh          ; ModeAttributes
        DB      06              ; WinAAttributes
        DB      00              ; WinBAttributes
        DW      32              ; WinGranularity
        DW      32              ; WinSize
        DW      0B800h          ; WinASegment
        DW      00000h          ; WinBSegment
        DW      0,0             ; WinFuncPtr
        DW      80              ; BytesPerScanLine

        DW      40              ; XResolution
        DW      25              ; YResolution
        DB      9               ; XCharSize  (This may be 8 for flat panels)
        DB      16              ; YCharSize
        DB      1               ; NumberOfPlanes
        DB      4               ; BitsPerPixel
        DB      1               ; NumberOfBanks
        DB      0               ; MemoryModel(text)
        DB      0               ; BankSize
        DB      15              ; NumberOfImagePages
        DB      1               ; Reserved
```

```
; IBM Mode 01 VBE Support
; Text 40x25

        DW      0000Eh          ; ModeAttributes
        DB      06              ; WinAAttributes
        DB      00              ; WinBAttributes
        DW      32              ; WinGranularity
        DW      32              ; WinSize
        DW      0B800h          ; WinASegment
        DW      00000h          ; WinBSegment
        DW      0,0             ; WinFuncPtr
        DW      80              ; BytesPerScanLine

        DW      40              ; XResolution
        DW      25              ; YResolution
        DB      9               ; XCharSize  (This may be 8 for flat panels)
        DB      16              ; YCharSize
        DB      1               ; NumberOfPlanes
        DB      4               ; BitsPerPixel
        DB      1               ; NumberOfBanks
        DB      0               ; MemoryModel(text)
        DB      0               ; BankSize
        DB      15              ; NumberOfImagePages
        DB      1               ; Reserved

;IBM Mode 02 VBE Support
; Text 80x25

        DW      0000Eh          ; ModeAttributes
        DB      06              ; WinAAttributes
        DB      00              ; WinBAttributes
        DW      32              ; WinGranularity
        DW      32              ; WinSize
        DW      0B800h          ; WinASegment
        DW      00000h          ; WinBSegment
        DW      0,0             ; WinFuncPtr
        DW      160             ; BytesPerScanLine
```

```
        DW      80                      ; XResolution
        DW      25                      ; YResolution
        DB      9                       ; XCharSize  (This may be 8 for flat panels)
        DB      16                      ; YCharSize
        DB      1                       ; NumberOfPlanes
        DB      4                       ; BitsPerPixel
        DB      1                       ; NumberOfBanks
        DB      0                       ; MemoryModel(text)
        DB      0                       ; BankSize
        DB      7                       ; NumberOfImagePages
        DB      1                       ; Reserved

;IBM Mode 03 VBE Support
; Text 80x25

        DW      0000Eh                  ; ModeAttributes
        DB      06                      ; WinAAttributes
        DB      00                      ; WinBAttributes
        DW      32                      ; WinGranularity
        DW      32                      ; WinSize
        DW      0B800h                  ; WinASegment
        DW      00000h                  ; WinBSegment
        DW      0,0                     ; WinFuncPtr
        DW      160                     ; BytesPerScanLine

        DW      80                      ; XResolution
        DW      25                      ; YResolution
        DB      9                       ; XCharSize  (This may be 8 for flat panels)
        DB      16                      ; YCharSize
        DB      1                       ; NumberOfPlanes
        DB      4                       ; BitsPerPixel
        DB      1                       ; NumberOfBanks
        DB      0                       ; MemoryModel(text)
        DB      0                       ; BankSize
        DB      7                       ; NumberOfImagePages
        DB      1                       ; Reserved
```

```
;IBM Mode 04 VBE Support
; 320x200x4

        DW      0001Eh                  ; ModeAttributes
        DB      06                      ; WinAAttributes
        DB      00                      ; WinBAttributes
        DW      32                      ; WinGranularity
        DW      32                      ; WinSize
        DW      0B800h                  ; WinASegment
        DW      00000h                  ; WinBSegment
        DD      0                       ; WinFuncPtr
        DW      80                      ; BytesPerScanLine

        DW      320                     ; XResolution
        DW      200                     ; YResolution
        DB      8                       ; XCharSize
        DB      8                       ; YCharSize
        DB      1                       ; NumberOfPlanes
        DB      2                       ; BitsPerPixel
        DB      2                       ; NumberOfBanks
        DB      1  ;CGA Graphics        ; MemoryModel
        DB      8                       ; BankSize
        DB      1                       ; NumberOfImagePages
        DB      1                       ; Reserved

; IBM Mode 05 VBE Support
; 320x200x4

        DW      0001Eh                  ; ModeAttributes
        DB      06                      ; WinAAttributes
        DB      00                      ; WinBAttributes
        DW      32                      ; WinGranularity
        DW      32                      ; WinSize
        DW      0B800h                  ; WinASegment
        DW      00000h                  ; WinBSegment
        DD      0                       ; WinFuncPtr
        DW      80                      ; BytesPerScanLine
```

```
        DW      320                     ; XResolution
        DW      200                     ; YResolution
        DB      8                       ; XCharSize
        DB      8                       ; YCharSize
        DB      1                       ; NumberOfPlanes
        DB      2                       ; BitsPerPixel
        DB      2                       ; NumberOfBanks
        DB      1  ;CGA Graphics        ; MemoryModel
        DB      8                       ; BankSize
        DB      1                       ; NumberOfImagePages
        DB      1                       ; Reserved

; IBM Mode 06 VBE Support
; 640x200x2

        DW      0001Eh                  ; ModeAttributes
        DB      06                      ; WinAAttributes
        DB      00                      ; WinBAttributes
        DW      32                      ; WinGranularity
        DW      32                      ; WinSize
        DW      0B800h                  ; WinASegment
        DW      00000h                  ; WinBSegment
        DD      0                       ; WinFuncPtr
        DW      80                      ; BytesPerScanLine

        DW      640                     ; XResolution
        DW      200                     ; YResolution
        DB      8                       ; XCharSize
        DB      8                       ; YCharSize
        DB      1                       ; NumberOfPlanes
        DB      1                       ; BitsPerPixel
        DB      2                       ; NumberOfBanks
        DB      1  ;CGA Graphics        ; MemoryModel
        DB      8                       ; BankSize
        DB      1                       ; NumberOfImagePages
        DB      1                       ; Reserved
```

```
; IBM Mode 07 VBE Support
; Text 80x25

        DW      00006h          ; ModeAttributes
        DB      06              ; WinAAttributes
        DB      00              ; WinBAttributes
        DW      32              ; WinGranularity
        DW      32              ; WinSize
        DW      0B000h          ; WinASegment
        DW      00000h          ; WinBSegment
        DW      0,0             ; WinFuncPtr
        DW      160             ; BytesPerScanLine

        DW      80              ; XResolution
        DW      25              ; YResolution
        DB      9               ; XCharSize  (This may be 8 for flat panels)
        DB      16              ; YCharSize
        DB      1               ; NumberOfPlanes
        DB      2               ; BitsPerPixel
        DB      1               ; NumberOfBanks
        DB      0               ; MemoryModel(text)
        DB      0               ; BankSize
        DB      7               ; NumberOfImagePages
        DB      1               ; Reserved

; IBM Mode 0D VBE Support
; 320x200x16

        DW      0001Eh          ; ModeAttributes
        DB      06              ; WinAAttributes
        DB      00              ; WinBAttributes
        DW      64              ; WinGranularity
        DW      64              ; WinSize
        DW      0A000h          ; WinASegment
        DW      00000h          ; WinBSegment
        DD      0               ; WinFuncPtr
        DW      40              ; BytesPerScanLine
```

```
        DW      320             ; XResolution
        DW      200             ; YResolution
        DB      8               ; XCharSize
        DB      8               ; YCharSize
        DB      4               ; NumberOfPlanes
        DB      4               ; BitsPerPixel
        DB      1               ; NumberOfBanks
        DB      3               ; MemoryModel
        DB      0               ; BankSize
        DB      7               ; NumberOfImagePages
        DB      1               ; Reserved

; IBM Mode 0E VBE Support
; 640x200x16

        DW      0001Eh          ; ModeAttributes
        DB      06              ; WinAAttributes
        DB      00              ; WinBAttributes
        DW      64              ; WinGranularity
        DW      64              ; WinSize
        DW      0A000h          ; WinASegment
        DW      00000h          ; WinBSegment
        DD      0               ; WinFuncPtr
        DW      80              ; BytesPerScanLine

        DW      640             ; XResolution
        DW      200             ; YResolution
        DB      8               ; XCharSize
        DB      8               ; YCharSize
        DB      4               ; NumberOfPlanes
        DB      4               ; BitsPerPixel
        DB      1               ; NumberOfBanks
        DB      3               ; MemoryModel
        DB      0               ; BankSize
        DB      3               ; NumberOfImagePages
        DB      1               ; Reserved
```

```
; IBM Mode 0F VBE Support
; 640x350x2

        DW      00016h          ; ModeAttributes
        DB      06              ; WinAAttributes
        DB      00              ; WinBAttributes
        DW      64              ; WinGranularity
        DW      64              ; WinSize
        DW      0A000h          ; WinASegment
        DW      00000h          ; WinBSegment
        DD      0               ; WinFuncPtr
        DW      80              ; BytesPerScanLine

        DW      640             ; XResolution
        DW      350             ; YResolution
        DB      8               ; XCharSize
        DB      14              ; YCharSize
        DB      4               ; NumberOfPlanes
        DB      4               ; BitsPerPixel
        DB      1               ; NumberOfBanks
        DB      3               ; MemoryModel
        DB      0               ; BankSize
        DB      1               ; NumberOfImagePages
        DB      1               ; Reserved

; IBM Mode 10 VBE Support
; 640x350x16

        DW      0001Eh          ; ModeAttributes
        DB      06              ; WinAAttributes
        DB      00              ; WinBAttributes
        DW      64              ; WinGranularity
        DW      64              ; WinSize
        DW      0A000h          ; WinASegment
        DW      00000h          ; WinBSegment
        DD      0               ; WinFuncPtr
        DW      80              ; BytesPerScanLine
```

```
        DW      640             ; XResolution
        DW      350             ; YResolution
        DB      8               ; XCharSize
        DB      14              ; YCharSize
        DB      4               ; NumberOfPlanes
        DB      4               ; BitsPerPixel
        DB      1               ; NumberOfBanks
        DB      3               ; MemoryModel
        DB      0               ; BankSize
        DB      1               ; NumberOfImagePages
        DB      1               ; Reserved

; IBM Mode 11 VBE Support
; 640x480x2

        DW      0001Eh          ; ModeAttributes
        DB      06              ; WinAAttributes
        DB      00              ; WinBAttributes
        DW      64              ; WinGranularity
        DW      64              ; WinSize
        DW      0A000h          ; WinASegment
        DW      00000h          ; WinBSegment
        DD      0               ; WinFuncPtr
        DW      80              ; BytesPerScanLine

        DW      640             ; XResolution
        DW      480             ; YResolution
        DB      8               ; XCharSize
        DB      16              ; YCharSize
        DB      4               ; NumberOfPlanes
        DB      4               ; BitsPerPixel
        DB      1               ; NumberOfBanks
        DB      3               ; MemoryModel
        DB      0               ; BankSize
        DB      0               ; NumberOfImagePages
        DB      1               ; Reserved
```

```
; IBM Mode 12 VBE Support
; 640x480x16

        DW      0001Eh          ; ModeAttributes
        DB      06              ; WinAAttributes
        DB      00              ; WinBAttributes
        DW      64              ; WinGranularity
        DW      64              ; WinSize
        DW      0A000h          ; WinASegment
        DW      00000h          ; WinBSegment
        DD      0               ; WinFuncPtr
        DW      80              ; BytesPerScanLine

        DW      640             ; XResolution
        DW      480             ; YResolution
        DB      8               ; XCharSize
        DB      16              ; YCharSize
        DB      4               ; NumberOfPlanes
        DB      4               ; BitsPerPixel
        DB      1               ; NumberOfBanks
        DB      3               ; MemoryModel
        DB      0               ; BankSize
        DB      0               ; NumberOfImagePages
        DB      1               ; Reserved

; IBM Mode 13 VBE Support
; 320x200x256

        DW      0001Eh          ; ModeAttributes
        DB      06              ; WinAAttributes
        DB      00              ; WinBAttributes
        DW      64              ; WinGranularity
        DW      64              ; WinSize
        DW      0A000h          ; WinASegment
        DW      00000h          ; WinBSegment
        DD      0               ; WinFuncPtr
        DW      320             ; BytesPerScanLine
```

```
DW      320             ; XResolution
DW      200             ; YResolution
DB      8               ; XCharSize
DB      8               ; YCharSize
DB      1               ; NumberOfPlanes
DB      8               ; BitsPerPixel
DB      1               ; NumberOfBanks
DB      4               ; MemoryModel
DB      0               ; BankSize
DB      0               ; NumberOfImagePages
DB      1               ; Reserved
```

Appendix 3 - VBE Supplemental Specs.

VESA Power Management (VBE/PM 1.0) Function Summary

(VBE/PM Function 4F10h)
00h - Return VBE/PM Information
01h - Set Display Power State
02h - Get Display Power State

VESA Audio Interface (VBE/AI 1.0) Function Summary

(VBE/AI Function 4F13h)
00h - Return VBE/AI Information
01h - Get Next Device Handle
02h - Get Device Class Information
03h - Open Device
04h - Close Device
05h - Driver Unload Request
06h - Driver Chaining
07h - Load 32-bit Interface

WAVE Audio Services

wsDeviceCheck()
wsPCMInfo()
wsPlayBlock()
wsRecordBlock()
wsPlayCont()
wsRecordCont()
wsPauseIO()
wsResumeIO()
wsStopIO()
wsTimerTick()
wsGetLastError()

MIDI Audio Services

msDeviceCheck()
msGlobalReset()
msMIDImsg()
msPreLoadPatch()
msUnloadPatch()
msTimerTick()
msGetLastError()

VOLUME Control Services

vsDeviceCheck()
vsSetVolume()
vsSetFieldVol()
vsToneControl()
vsFilterControl()
vsOutputPath()
vsGetLastError()

VESA BIOS Extensions / Display Data Channel (VBE/DDC 1.0) Function Summary

(VBE/DDC Function 4F15h)
00h - Report VBE/DDC Capabilities
01h - Read EDID
02h - Read VDIF

Appendix 4 - VBE Implementation Considerations

This appendix discusses required features of VBE 2.0. implementations, and offers some suggestions for consideration by BIOS developers. Some issues raised here apply only to adding VBE 2.0 to an existing VGA BIOS, while other issues are more generally relevant.

A4.1 Minimum Functionality Requirements

A4.1.1 Required VBE Services

VBE Functions 00h-0Ah are required; all other functions are optional. There are no absolutely required modes, or mode capabilities, since these will vary according to the hardware and applications.

A4.1.2 Minimum ROM Implementation

For compliance certification, Functions 00-0Ah must be implemented in the ROM. In the case of ROM space limitations that do not allow full implementations, VESA strongly recommends that VBE Get Controller Information Function 00h be implemented in the ROM so applications will be able to find information about the controller type and capabilities. This 'Stub' implementation can be supplemented by a TSR which will provide full VBE Core functionality. These stub implementations are not VBE 2.0 compliant and should only be implemented in cases where no space is available to implement the whole VBE. In the event that a stub is implemented a TSR must be available to complete VBE 2.0 functionality.

In a stub implementation, the VideoModeList will contain no entries (starts with 0FFFFh). This is the indicator to application software that the VBE Core implementation is in fact only a stub and that other functions and modes do not exist.

A4.1.3 TSR Implementations

TSR based implementations of VBE must not assume that a compatible graphics controller is present! They must first attempt to detect the presence of a compatible device before chaining into INT 10h and completing the load process. If no compatible hardware is detected they must exit without chaining INT 10h. On failure to load, the TSR should display an appropriate message to the screen, identifying both the installed and expected hardware and displaying the OEM strings from Function 00h, if available. The software version number and identifying information for the TSR should also be shown.

TSRs which are meant to work in a variety of hardware and BIOS environments should check to see if the ROM supports some version of VBE Function 00h, Get Controller Information. The information which is returned from this function can then be passed on to calling applications or displayed on the screen, reducing the burden of supporting different display hardware. If a stub or incomplete version of VBE exists in ROM, it is the responsibility of the TSR to supplement all missing functions and replace Function 00h.

VESA recommends that VBE 2.0 TSRs be given names which contain some identifier for the OEM and/or product, as well as a '2' to indicate the VBE version supported. This will help users make sure they have the correct version of software for their hardware, and may prevent a few phone calls for software support. It is also required that a help screen be included, which can be activated by typing "/h", "/?", or any unrecognized parameter on the command line. The help screen should contain all pertinent information about the source and version number of the TSR and the hardware on which it is designed to work.

A4.2 VGA BIOS Implications

A primary design goal with the VESA VBE is to minimize the effects on the standard VGA BIOS. Standard VGA BIOS functions should need to be modified as little as possible. However, two standard VGA functions are affected by the VBE. These are VGA Function 00h (Set VGA Mode) and VGA Function 0Fh (Get Current VGA Mode).

VBE-unaware applications (such as old Pop-Up programs and other TSRs, or the CLS command of MS-DOS), may use VGA BIOS Function 0Fh to get the current display mode and later call VGA BIOS Function 00h to restore/reinitialize the old graphics mode. To make such applications work, the 8-bit value returned by VGA BIOS Function 0Fh (it is up to the OEM to define this number), must correctly reinitialize the graphics mode through VGA BIOS Function 00h.

However, VBE-aware applications should not set the VBE mode using VGA BIOS Function 00h, or get the current mode with VGA BIOS Function 0Fh. VBE Functions 02h (Set VBE mode) and 03h (Get VBE Mode) should be used instead. The mode number must be from the mode list returned by VBE Function 00h, and Function 03h must return the same mode number used to set the mode in Function 02h.

Given these requirements, and the fact that many BIOS manufacturers will need to support at least some of the VESA-defined 14-bit mode numbers for backwards compatibility, it is clear that the BIOS must keep track of the last mode that was set with VBE Function 02h. There are various ways that this could be accomplished without the use of scratch registers or non-volatile RAM, which is not always available. One method is to use the mode number byte in the BIOS Data Area to store the index into the mode list returned in VBE Function 00h, which is always stored in ROM. Another method is to store a small translation table for the 14-bit mode numbers (probably necessary for using duplicate mode numbers anyway) and to use an obsolete or unused bit in the BIOS Data Area to indicate a 14-bit mode in effect.

If a BIOS offers only the flat frame buffer version of one of the modes which have VESA-defined numbers, it may be advisable to use an OEM-defined number for that mode instead. Since VBE 1.2 and earlier versions assume standard VGA windowing of the frame buffer, older VBE-aware applications may recognize the mode number and attempt to use windowed memory without properly checking with Function 01h.

A4.3 ROM Space Limitations

Since standard VGA BIOS is currently confined to 32K ROM images, space is likely to be critical in implementing even the minimum VBE 2.0 functionality. Most VGA BIOSs have already been compressed many times as new features and modes have been added over time. Clearly, older VGA BIOS features may have to be sacrificed to make room.

A4.3.1 Data Storage

To allow for ROM based execution of the VBE functions, each VBE function must be implemented without the use of any local data. When possible, the BIOS data area, non-volatile RAM, or OEM specific scratch registers can be used to place scratch data during execution. All VBE data structures are allocated and provided to VBE by the calling application.

A4.3.2 Removal of Unused VGA Fonts

VESA strongly recommends that removal of the 8x14 VGA font become a standard way of freeing up space for VBE 2.0 implementations. The removal of this font leaves 3.5K bytes of ROM space for new functions, and is probably the least painful way to free up such a large amount of space while preserving as much backwards compatibility as possible. The 8x14 font is normally used for VGA Modes 0*, 3* and Mode 10h, which are 350-line or EGA compatible modes. When these modes are selected the 8x16 font may be chopped and used instead. When chopping a 16 point font to replace the 14 point, there are several characters (ones with descenders) that should be special cased.

Some applications which use the 8x14 font obtain a pointer to the font table through the standard VGA functions and then use this table directly. In such cases, no workaround using the 8x16 font is possible and a TSR with the 8x14 font is unavoidable. Some OEMs may find this situation unacceptable because of the potential for an inexperienced user to encounter "garbage" on the screen if the TSR is not present. However, OEMs may also find eventually that demand for VBE 2.0 services is great enough to justify the inconvenience associated with an 8x14 font TSR. To date, no compatibility problems are known to be caused by the use of such a TSR. VESA will make available a TSR that replaces the 8x14 font, please contact VESA for more information.

Another option with the fonts in Turn-Key systems (such as Laptops, Notebooks etc.) is to move the fonts to another location in the System ROM. In fact VBE functions could even be relocated. This however is not an acceptable solution for most desktop systems, where they are expandable.

A4.3.3 Deleting VGA Parameter Tables

One way to create more ROM space for the VBE is to delete some of the VGA parameter tables by deleting modes which are outdated and little used. Many of the standard VGA modes are now almost entirely obsolete and should probably be phased out of existence. How quickly this might happen depends on which applications are still using the older modes and on how tolerant OEMs and users will be to using TSR programs for these modes when necessary. Some mode groups which might be candidates for removal are modes 4, 5, and 6, all CGA modes, or all 200 line modes.

It must be emphasized, however, that it is absolutely necessary to preserve the size and positions of all the standard mode VGA parameter tables! Failure to do so will cause a lot of problems with diagnostics and older VGA applications. If a table is removed, fill the space with an equal number of bytes of code or data.

A4.3.4 Increasing ROM Space

In the PC environment, VGA BIOS developers have traditionally been limited to a 32K ROM image located at C0000h-C7FFFh. The C8000h-CBFFFh area was originally reserved for the XT hard disk BIOS, which is of little current concern. However, SCSI CD ROM controllers have now begun to use this area, and the possibility exists that other devices may use this area also. It is unlikely that VBE developers will be able to expand into the C8000h-CFFFFh region without creating potential conflicts.

4.3.5 Support of VGA TTY Functions

The support of VGA TTY functions is recommended, but not mandatory, for graphic modes beyond VGA. TTY support for all modes is desirable to allow basic text operations such as reading and writing characters to the screen. Some operating systems will revert to using TTY functions when a hardware error occurs, since the graphics environment may no longer be operational.

Support of TTY functions for all modes will, of course, increase the size of the BIOS. One possible solution is to provide TTY function support for extended modes as part of a TSR rather than in the ROM.

Bit D2 in the Mode Attributes field in the ModeInfoBlock structure returned by VBE Function 01h indicates the presence of support for TTY functions for each VBE mode. Refer to the VBE Function 01h description for details on which TTY functions must be supported when this bit is set.

A4.4 Implementation Notes by Function

A4.4.1 General Notes

Starting with VBE version 2.0 VESA will no longer define new VESA mode numbers and it will not longer be mandatory to support these old mode numbers. However, it is highly recommended that BIOS implementations continue to support these mode numbers for compatibility with old software. VBE 2.0 aware applications should follow the guidelines in Appendix 5 - Application Programming Considerations - for setting a desired mode.

Applications should treat any non-zero value in the AH register as a general failure condition as later versions of the VBE may define additional error codes. BIOS developers should refrain from defining their own return codes, which may conflict with future VESA-defined return codes.

VESA strongly recommends the preservation of the Graphics Controller indexes.

A4.4.2 Function 00h - Return VBE Controller Information

All data in the structure is subject to change by the VBE implementation when VBE Function 00h is called. Therefore it should not be used by the application to store data of any kind. VBE should fill any unused portion of the structure with zeros.

The BCD value for VBE 2.0 is 0200h, The BCD value for VBE 1.2 is 0102h. In the past we have had some applications misinterpreting these BCD values. For example, BCD 0102h was interpreted as 1.02, which is incorrect.

The length of the OEMString is not defined, but for space considerations, we recommend a string length of less than 256 bytes.

The DAC must always be restored to 6 bits per primary as default upon a mode set. If the DAC has been switched to 8 bits per primary, the mode set must restore the DAC to 6 bits per primary to ensure the application developer that he does not have to reset it.

If the RAMDAC is an older style RAMDAC with the possibility of "snow" during programming, the VBE 2.0 implementation must place a 1 in bit 2 of the Capabilities field.

If a VideoModeList is found to contain no entries (starts with 0FFFFh), it can be assumed that the VBE implementation is a "stub" implementation where only Function 00h is supported for diagnostic or "Plug and Play" reasons. These stub implementations are not VBE 2.0 compliant and should only be implemented in cases where no space is available to implement the whole VBE.

The length of the strings OemProductRev, OemProductName and OemVendorName (including terminators), summed, must fit within a 256 byte buffer. This is to allow for return in the OemData field if necessary.

A4.4.3 Function 01h - Return VBE Mode Information

Monochrome modes map their CRTC address at 3B4h. Color modes map their CRTC address at 3D4h. Monochrome modes have attributes in which only bit 3 (video) and bit 4 (intensity) of the attribute controller output are significant. Therefore, monochrome text modes have attributes of off, video, high intensity, blink, etc. Monochrome graphics modes are two plane graphics modes and have attributes of off, video, high intensity, and blink. Extended two color modes that have their CRTC address at 3D4h, are color modes with one bit per pixel and one plane. The standard VGA modes, 06h and 11h would be classified as color modes, while the standard VGA modes 07h and 0Fh would be classified as monochrome modes.

Version 1.1 and later VBE will zero out all unused fields in the Mode Information Block, always returning exactly 256 bytes. This facilitates upward compatibility with future versions of the standard, as any newly added fields will be designed such that values of zero will indicate nominal defaults or non-implementation of optional features. (For example, a field containing a bit-mask of extended capabilities would reflect the absence of all such capabilities.) Applications that wish to be backwards compatible to Version 1.0 VBE should pre-initialize the 256 byte buffer before calling the Return VBE Mode Information function.

If the ModeInfoBlock is for an IBM Standard VGA mode and the NumberOfImagePages field contains more pages than would be found in a 256KB implementation, the TTY support described in the ModeAttributes must be accurate, i.e., if the TTY functions are claimed to be supported, they must be supported in all pages, not just the pages normally found in the 256KB implementation.

A4.4.4 Function 02h - Set VBE Mode

VBE BIOS 2.0 implementations should also update the BIOS Data Area 40:87 (memory clear bit) so that Function 03h can return this flag. VBE BIOS 1.2 and earlier BIOS implementations ignore the memory clear bit.

Function 00h of an IBM VGA compatible BIOS uses D7 to signify the same thing as D15 does in this function. If D7 is set for an IBM compatible mode when calling this function, this mode set should fail. VBE aware applications must set the memory clear bit in D15.

This call should not set modes not listed in the list of supported modes. All modes (including IBM standard VGA modes), if listed as supported, must have ModeInfoBlock structures associated with them.

Mode 81FFh is a special mode designed to preserve the current memory contents and to give access to the entire video memory. This mode is especially useful for saving the entire video memory contents before going into a state that could lose the contents (e.g. set this mode to gain access to all video memory to save it before going into a volatile power down state). This mode is required as the entire video memory contents are not always accessible in every mode. It is recommended that this mode be packed pixel in format, and a ModeInfoBlock must be defined for it. However, it should not appear in the VideoModeList. Look in the ModeInfoBlock to determine if paging is required and, if it is required, how it is supported. Also note that there are no implied resolutions or timings associated with this mode.

A4.4.5 Function 03h - Return Current VBE Mode

Version 1.x Note: In a standard VGA BIOS, Function 0Fh (Read current video state) returns the current graphics mode in the AL register. In D7 of AL, it also returns the status of the memory clear bit (D7 of 40:87). This bit is set if the mode was set without clearing memory. In this VBE function, the memory clear bit will not be returned in BX since the purpose of the function is to return the video mode only. If an application wants to obtain the memory clear bit, it should call the standard VGA BIOS Function 0Fh.

Version 2.x Note: Unlike version 1.x VBE implementations, the memory clear flag will be returned. The application should NOT call the standard VGA BIOS Function 0Fh if the mode was set with VBE Function 02h.

The mode number returned must be the same mode number used in the VBE Function 02h mode set.

This function is not guaranteed to return an accurate mode value if the mode set was not done with VBE Function 02h. In that case, the results are unspecified.

A4.4.6 Function 05h - Display Window Control

In VBE 1.2 implementations, the direct far call version returns no Return Status information to the application. Also, in the far call version, the AX and DX registers will be destroyed. Therefore if AX and/or DX must be preserved, the application must do so prior to making the far call. The application must still load the input arguments in BH, BL, and DX (for Set Window). In VBE 2.0 implementations, the BIOS will return the correct Return Status, and therefore the application must assume that AX and DX will be destroyed.

If this function is called while in a linear frame buffer memory model, this function must fail with completion code AH=03h.

A4.4.7 Function 06h - Get/Set Logical Scan Line Length

The desired width in pixels may not be achievable because of hardware considerations. The next larger value will be selected that will accommodate the desired number of pixels, and the actual number of pixels will be returned in CX. BX returns a value that when added to a pointer into display memory will point to the next scan line. For example, in VGA mode 13h this would be 320, but in mode 12h this would be 80. DX returns the number of logical scan lines based upon the new scan line length and the total memory installed and usable in this display mode.

On a failure to set scan line length by setting a CX value too large, the function will fail with error code 02h.

The value returned when BL=03h is the lesser of either the maximum line length that the hardware can support, or the longest scan line length that would support the number of lines in the current video mode.

This function is also valid in text modes. In text modes the application should convert the character line length to pixel line length by getting the current character cell width through the XCharSize field returned in ModeInfoBlock, multiplying that times the desired number of characters per line, and passing that value in the CX register.

In text modes, this function will only work if the line length is specified in character granularity. i.e. in 8 dot modes only multiples of 8 will work. Any value which is not in character granularity will result in a function call failure.

A4.4.8 Function 07h - Get/Set Display Start

This function is also valid in text modes. To use this function in text mode, the application should convert the character coordinates to pixel coordinates by using XCharSize and YCharSize returned in the ModeInfoBlock. If the requested Display Start coordinates do not allow for a full page of video memory or the hardware does not support memory wrapping, the Function call should fail and no changes should be made. As a general case, if a requested Display Start is not available, fail the Function call and make no changes.

A4.4.9 Function 08h - Set/Get DAC Palette Format

This function will return failure code 03h if called in a direct color or YUV mode.

A4.4.10 Function 09h - Set/Get Palette Data

The need for BL= 80h is for older style RAMDAC's where programming the RAM values during display time causes a "snow-like" effect on the screen. Newer style RAMDAC's don't have this limitation and can easily be programmed at any time, but older RAMDAC's require that they be programmed during a non-display time only to stop the snow like effect seen when changing the DAC values. When this is requested the VBE implementation will program the DAC with blanking on. Check D2 of the Capabilities field returned by VBE Function 00h to determine if 80h should be used instead of 00h.

The need for the secondary palette is for anticipated future palette extensions, if a secondary palette does not exist in a implementation and these calls are made, the VBE implementation will return error code 02h.

A4.4.11 Function 0Ah - Return VBE Function Information

All protected mode functions should end with a near RET (as opposed to FAR RET) to allow the application software to CALL the code from within the ROM.

The Port and Memory location Sub-table does not include the Frame Buffer Memory location. The Frame Buffer Memory location is contained within the ModeInfoBlock returned by VBE Function 01h.

The protected mode code must be assembled for a 32-bit code segment, when copying it, the application must copy the code to a 32-bit code segment.

A4.5 Plug and Play Issues

Plug and Play information may be used to fill in the VBE Function 00h (Return VBE Controller Information) data structures. Since VBE Function 00h returns information such as product name, or product revision, which must be hard-coded in the ROM BIOS, Plug and Play may help to avoid the need for the display board manufacturer to alter or customize the ROM binary image.

A4.6 Supporting Multiple Controllers

It is sometimes necessary for more than one display controller to be present in the system for several reasons. For example, OEMs may choose to implement a dual-controller design with VGA functionality provided by one controller, and SVGA or VBE functionality provided by a second controller. In some cases, it may be desirable to install more than one display adapter in the system for simultaneous support of multiple display monitors.

A4.6.1 Dual-Controller Designs

VBE 2.0 supports the dual-controller design by assuming that since both controllers are typically provided by the same OEM, under control of a single BIOS ROM, it is possible to hide the fact that two controllers are indeed present from the application. This has the limitation of preventing simultaneous use of the independent controllers, but allows applications released before VBE 2.0 to operate normally. The VBE Function 00h (Return Controller Information) returns the combined information of both controllers, including the combined list of available modes. When the application selects a mode, the appropriate controller is activated. Each of the remaining VBE functions then operates on the active controller.

A4.6.2 Provision for Multiple Independent Controllers

There are no provisions for multiple independent controllers under VBE at this time. If it ever becomes necessary, support of additional display controllers can be provided under the "Supplemental Specification" guidelines.

A4.7 Display Refresh Rates and Interlacing

Display refresh rates, interlacing, and other timing parameters are automatically implied for each graphics mode. Application programs should not be concerned with these hardware details, and therefore should assume that the most desirable timings are selected for each graphics mode.

The VGA standard defines the timing details for each VGA mode, and all VGA monitors support these timing requirements. Additional graphics modes implemented under VBE should operate at the maximum frequency possible for the display controller and installed monitor. This presents a problem since the manufacturer of the display controller may be unaware of what display monitor hardware is installed, and what its capabilities are.

VESA is in the process of standardizing a mechanism by which the display controller can automatically determine the capabilities of the installed display monitor. The maximum refresh rate and need for interlacing for each graphics mode can then be determined based on the display controller timing logic and the hardware profile of the installed monitor. See the VESA Display Data Channel and Graphics Configuration documents for more information.

If the VESA Display Data Channel capability is absent, the display controller manufacturer must provide a configuration utility, or other suitable process, to select the best available timing for each mode. It is permissible for the display controller OEM to extend the VBE implementation with private functions to assist with the configuration of the display hardware in a standard way across the product line (see section A4.8). This allows support of all controllers in the product line with a single configuration utility that makes use of these private functions. When an automatic method is not available, the end user can run this utility when a new monitor is installed to select the refresh rates and adjust other parameters such as centering for each graphics mode. Monitor specific information must then be stored, either in scratch registers, non-volatile RAM, or, more likely, through the use of a configuration file and associated TSR.

A utility accessing these proprietary functions must read the VbeInfoBlock returned by VBE Function 00h to determine if the firmware is of the proper type and revision level before making any Function 14h calls. Failure to do so will render the calling utility incompatible with VBE 2.0 and may cause unpredictable results.

A4.8 OEM Extensions to VBE

The VBE specification allows the OEM to extend its functionality for support of nonstandard, or private features known only to the OEM and custom applications that are aware of these OEM extensions.

VBE Function 14h is reserved for use by OEMs wishing to add VBE subfunctions of their own. This function number is provided so that the OEM may add custom services without fear of conflict with other VBE services. These subfunctions must use the AX register in the same manner as all other standard VBE functions and return the standard VBE completion codes.

Normally, these extended functions are used by the OEM to aid in the setup and configuration of the controller hardware. For example, during installation it may be necessary to set the physical frame buffer address, maximum monitor refresh frequency, default graphics mode, default power state, etc. A single setup and installation program can be used by the OEM with the entire product line if the same OEM extensions are implemented on each product.

A utility accessing these proprietary functions must read the VbeInfoBlock returned by VBE Function 00h to determine if the firmware is of the proper type and revision level before making any Function 14h calls. Failure to do so will render the calling utility incompatible with VBE 2.0 and may cause unpredictable results.

A4.9 Certification Requirements

Perhaps one of the key differences in VBE 2.0 over earlier revisions, is the certification requirements of VBE 2.0. There is only one type of certification that can be done through VESA, this is "Compliance". For Compliance with VBE 2.0, an implementation must pass a certification process. Compliance testing will require VBE functions in ROM, and will benefit the end user, the video vendor and the application developers. Vendors who pass compliance testing, may license the VESA VBE 2.0 logo, and may market their products as VBE 2.0 Compliant.

Another term that can be associated with VBE 2.0 implementations is "Compatible". VBE 2.0 compatible systems cannot be marketed as VBE 2.0 Compliant, VBE 2.0 Compatible nor can they use the VESA VBE 2.0 logo. VBE Compatible means that a system implements VBE 2.0 correctly but does not have the features in ROM. VBE Compatibility is not the desired method of VBE implementation because of the Plug and Play advantages of the ROM implementation. VBE Compatibility is targeted for the upgrading of older video systems.

A4.9.1 VBETest Utility

A Test program will be developed which must be passed in order to claim your system is VBE Compliant. It is extremely important that even VBE Compatible implementations be tested with this test. The test will be made available to VESA Members only, however discussion on the merits of distributing this test to the general public is going on at this time.

A4.9.2 Communication with VESA Office

To find out more about VBE 2.0 Compliant Logo licensing and testing, please contact the VESA office.

Appendix 5 - Application Programming Considerations

A5.1 Application Developer's Sample Source

The certification process is only for the BIOS implementations, this should be enough to ensure that the applications fall in line with the VESA standard. If it doesn't work on a VBE Compliant card, then the application is wrong and should be changed. To help ensure that the application developer will work on VBE Compliant systems, sample source for application developer's will be provided by the VESA office.

A simple example of how to set a Video Mode, and how to use it to put something up on the screen, is found below. This is not intended to be a complete SDK or source example, but it only demonstrates what we are trying to achieve.

C Language Module

(This has been compiled and tested under Microsoft C 6.0. Conversion for the direct banking method to inline assembly may be required for Borland C.)

```
/****************************************************************************
*
*                               Hello VBE!
*
*
* Language:     C (Keyword far is by definition not ANSI, therefore
*                  to make it true ANSI remove all far references and
*                  compile under MEDIUM model.)
*
* Environment:  IBM PC (MSDOS) 16 bit Real Mode
* Original code contributed by:     - Kendall Bennett, SciTech Software
* Conversion to Microsoft C by:     - Rex Wolfe, Western Digital Imaging
*                                   - George Bystricky, S-MOS Systems
*
* Description:  Simple 'Hello World' program to initialize a user
*               specified 256 color graphics mode, and display a simple
*               moire pattern. Tested with VBE 1.2 and above.
*
*               This code does not have any hard-coded VBE mode numbers,
*               but will use the VBE 2.0 aware method of searching for
*               available video modes, so will work with any new extended
*               video modes defined by a particular OEM VBE 2.0 version.
*
*               For brevity we don't check for failure conditions returned
*               by the VBE (but we shouldn't get any).
*
****************************************************************************/

#include <stdio.h>
#include <stdlib.h>
```

```
#include <dos.h>
#include <conio.h>
```

```
/* Comment out the following #define to disable direct bank switching.
 * The code will then use Int 10h software interrupt method for banking. */

#define  DIRECT_BANKING

#ifdef  DIRECT_BANKING
/* only needed to setup registers BX,DX prior to the direct call.. */
extern far setbxdx(int, int);
#endif

/*---------------------- Macro and type definitions -----------------------*/

/* SuperVGA information block */

struct
{
    char    VESASignature[4];       /* 'VESA' 4 byte signature         */
    short   VESAVersion;            /* VBE version number              */
    char    far *OEMStringPtr;      /* Pointer to OEM string           */
    long    Capabilities;           /* Capabilities of video card      */
    unsigned far *VideoModePtr;     /* Pointer to supported modes      */
    short   TotalMemory;            /* Number of 64kb memory blocks    */
    char    reserved[236];          /* Pad to 256 byte block size      */
} VbeInfoBlock;

/* SuperVGA mode information block */

struct
{
    unsigned short ModeAttributes;      /* Mode attributes               */
    unsigned char  WinAAttributes;      /* Window A attributes           */
    unsigned char  WinBAttributes;      /* Window B attributes           */
    unsigned short WinGranularity;      /* Window granularity in k       */
    unsigned short WinSize;             /* Window size in k              */
    unsigned short WinASegment;         /* Window A segment              */
    unsigned short WinBSegment;         /* Window B segment              */
    void (far *WinFuncPtr)(void);       /* Pointer to window function    */
    unsigned short BytesPerScanLine;    /* Bytes per scanline            */
    unsigned short XResolution;         /* Horizontal resolution         */
    unsigned short YResolution;         /* Vertical resolution           */
    unsigned char  XCharSize;           /* Character cell width          */
    unsigned char  YCharSize;           /* Character cell height         */
    unsigned char  NumberOfPlanes;      /* Number of memory planes       */
    unsigned char  BitsPerPixel;        /* Bits per pixel                */
    unsigned char  NumberOfBanks;       /* Number of CGA style banks     */
    unsigned char  MemoryModel;         /* Memory model type             */
    unsigned char  BankSize;            /* Size of CGA style banks       */
    unsigned char  NumberOfImagePages;  /* Number of images pages        */
    unsigned char  res1;                /* Reserved                      */
    unsigned char  RedMaskSize;         /* Size of direct color red mask */
    unsigned char  RedFieldPosition;    /* Bit posn of lsb of red mask   */
    unsigned char  GreenMaskSize;       /* Size of direct color green mask */
```

```
   unsigned char  GreenFieldPosition;  /* Bit posn of lsb of green mask   */
   unsigned char  BlueMaskSize;       /* Size of direct color blue mask  */
   unsigned char  BlueFieldPosition;   /* Bit posn of lsb of blue mask    */
   unsigned char  RsvdMaskSize;       /* Size of direct color res mask   */
   unsigned char  RsvdFieldPosition;   /* Bit posn of lsb of res mask    */
   unsigned char  DirectColorModeInfo; /* Direct color mode attributes   */
   unsigned char  res2[216];          /* Pad to 256 byte block size     */
} ModeInfoBlock;

typedef enum
{
    memPL       = 3,               /* Planar memory model              */
    memPK       = 4,               /* Packed pixel memory model        */
    memRGB      = 6,               /* Direct color RGB memory model    */
    memYUV      = 7,               /* Direct color YUV memory model    */
} memModels;

/*------------------------- Global Variables ----------------------------*/
char mystr[256];
char *get_str();

int     xres,yres;                 /* Resolution of video mode used    */
int     bytesperline;              /* Logical CRT scanline length      */
int     curBank;                   /* Current read/write bank          */
unsigned int bankShift;            /* Bank granularity adjust factor   */
int     oldMode;                   /* Old video mode number            */
char    far *screenPtr;            /* Pointer to start of video memory */
void   (far *bankSwitch)(void);    /* Direct bank switching function   */
/*----------------------- VBE Interface Functions ----------------------*/

/* Get SuperVGA information, returning true if VBE found */

int getVbeInfo()
{
   union REGS in,out;
   struct SREGS segs;
   char far *VbeInfo = (char far *)&VbeInfoBlock;
   in.x.ax = 0x4F00;
   in.x.di = FP_OFF(VbeInfo);
   segs.es = FP_SEG(VbeInfo);
   int86x(0x10, &in, &out, &segs);
   return (out.x.ax == 0x4F);
}

/* Get video mode information given a VBE mode number. We return 0 if
 * if the mode is not available, or if it is not a 256 color packed
 * pixel mode.
 */

int getModeInfo(int mode)
{
   union REGS in,out;
   struct SREGS segs;
   char far *modeInfo = (char far *)&ModeInfoBlock;
```

```
    if (mode < 0x100) return 0;     /* Ignore non-VBE modes           */
    in.x.ax = 0x4F01;
    in.x.cx = mode;
    in.x.di = FP_OFF(modeInfo);
    segs.es = FP_SEG(modeInfo);
    int86x(0x10, &in, &out, &segs);
    if (out.x.ax != 0x4F) return 0;
    if ((ModeInfoBlock.ModeAttributes & 0x1)
            && ModeInfoBlock.MemoryModel == memPK
            && ModeInfoBlock.BitsPerPixel == 8
            && ModeInfoBlock.NumberOfPlanes == 1)
        return 1;
    return 0;
}

/* Set a VBE video mode */

void setVBEMode(int mode)
{
    union REGS in,out;
    in.x.ax = 0x4F02; in.x.bx = mode;
    int86(0x10,&in,&out);
}

/* Return the current VBE video mode */

int getVBEMode(void)
{
    union REGS in,out;
    in.x.ax = 0x4F03;
    int86(0x10,&in,&out);
    return out.x.bx;
}

/* Set new read/write bank. We must set both Window A and Window B, as
 * many VBE's have these set as separately available read and write
 * windows. We also use a simple (but very effective) optimization of
 * checking if the requested bank is currently active.
 */

void setBank(int bank)
{
    union REGS  in,out;
    if (bank == curBank) return;    /* Bank is already active         */
    curBank = bank;                 /* Save current bank number       */
    bank <<= bankShift;             /* Adjust to window granularity   */
#ifdef  DIRECT_BANKING
    setbxdx(0,bank);
    bankSwitch();
    setbxdx(1,bank);
    bankSwitch();
#else
```

```
   in.x.ax = 0x4F05; in.x.bx = 0;  in.x.dx = bank;
   int86(0x10, &in, &out);
   in.x.ax = 0x4F05; in.x.bx = 1;  in.x.dx = bank;
   int86(0x10, &in, &out);
#endif
}

/*------------------------- Application Functions ------------------------*/

/* Plot a pixel at location (x,y) in specified color (8 bit modes only) */

void putPixel(int x,int y,int color)
{
   long addr = (long)y * bytesperline + x;
   setBank((int)(addr >> 16));
   *(screenPtr + (addr & 0xFFFF)) = (char)color;
}

/* Draw a line from (x1,y1) to (x2,y2) in specified color */

void line(int x1,int y1,int x2,int y2,int color)
{
    int     d;                      /* Decision variable               */
    int     dx,dy;                  /* Dx and Dy values for the line   */
    int     Eincr,NEincr;           /* Decision variable increments    */
    int     yincr;                  /* Increment for y values          */
    int     t;                      /* Counters etc.                   */

#define ABS(a)   ((a) >= 0 ? (a) : -(a))

    dx = ABS(x2 - x1);
    dy = ABS(y2 - y1);
    if (dy <= dx)
    {
        /* We have a line with a slope between -1 and 1
          *
         * Ensure that we are always scan converting the line from left to
         * right to ensure that we produce the same line from P1 to P0 as the
         * line from P0 to P1.
          */
        if (x2 < x1)
        {
            t = x2; x2 = x1; x1 = t;    /* Swap X coordinates          */
            t = y2; y2 = y1; y1 = t;    /* Swap Y coordinates          */
        }
        if (y2 > y1)
            yincr = 1;
        else
            yincr = -1;
        d = 2*dy - dx;              /* Initial decision variable value  */
        Eincr = 2*dy;               /* Increment to move to E pixel     */
        NEincr = 2*(dy - dx);       /* Increment to move to NE pixel    */
        putPixel(x1,y1,color);      /* Draw the first point at (x1,y1)  */
```

```
       /* Incrementally determine the positions of the remaining pixels */
       for (x1++; x1 <= x2; x1++)
       {
           if (d < 0)
              d += Eincr;         /* Choose the Eastern Pixel         */
           else
           {
              d += NEincr;        /* Choose the North Eastern Pixel   */
              y1 += yincr;        /* (or SE pixel for dx/dy < 0!)     */
           }
           putPixel(x1,y1,color);  /* Draw the point                  */
       }
    }
    else
    {
       /* We have a line with a slope between -1 and 1 (ie: includes
        * vertical lines). We must swap our x and y coordinates for this.
        *
        * Ensure that we are always scan converting the line from left to
        * right to ensure that we produce the same line from P1 to P0 as the
        * line from P0 to P1.
        */
       if (y2 < y1)
       {
           t = x2; x2 = x1; x1 = t;    /* Swap X coordinates          */
           t = y2; y2 = y1; y1 = t;    /* Swap Y coordinates          */
       }
       if (x2 > x1)
           yincr = 1;
       else
           yincr = -1;
       d = 2*dx - dy;             /* Initial decision variable value  */
       Eincr = 2*dx;              /* Increment to move to E pixel     */
       NEincr = 2*(dx - dy);      /* Increment to move to NE pixel    */
       putPixel(x1,y1,color);     /* Draw the first point at (x1,y1)  */

       /* Incrementally determine the positions of the remaining pixels */
       for (y1++; y1 <= y2; y1++)
       {
           if (d < 0)
              d += Eincr;         /* Choose the Eastern Pixel         */
           else
           {
              d += NEincr;        /* Choose the North Eastern Pixel   */
              x1 += yincr;        /* (or SE pixel for dx/dy < 0!)     */
           }
           putPixel(x1,y1,color);  /* Draw the point                  */
       }
    }
}
```

```
/* Draw a simple moire pattern of lines on the display */

void drawMoire(void)
{
    int     i;

    for (i = 0; i < xres; i += 5)
    {
       line(xres/2,yres/2,i,0,i % 0xFF);
       line(xres/2,yres/2,i,yres,(i+1) % 0xFF);
    }
    for (i = 0; i < yres; i += 5)
    {
       line(xres/2,yres/2,0,i,(i+2) % 0xFF);
       line(xres/2,yres/2,xres,i,(i+3) % 0xFF);
    }
   line(0,0,xres-1,0,15);
   line(0,0,0,yres-1,15);
   line(xres-1,0,xres-1,yres-1,15);
   line(0,yres-1,xres-1,yres-1,15);
}

/* Return NEAR pointer to FAR string pointer*/

char *get_str(char far *p)
{
    int i;
    char *q=mystr;

   for(i=0;i<255;i++)
    {
       if(*p) *q++ = *p++;
       else break;
    }
    *q = '\0';
    return(mystr);
}

/* Display a list of available resolutions. Be careful with calls to
 * function 00h to get SuperVGA mode information. Many VBE's build the
 * list of video modes directly in this information block, so if you
 * are using a common buffer (which we aren't here, but in protected
 * mode you will), then you will need to make a local copy of this list
 * of available modes.
 */

void availableModes(void)
{
    unsigned far    *p;

   if (!getVbeInfo())
    {
       printf("No VESA VBE detected\n");
        exit(1);
    }
```

```
   printf("VESA VBE Version %d.%d detected (%s)\n\n",
      VbeInfoBlock.VESAVersion >> 8, VbeInfoBlock.VESAVersion & 0xF,
      get_str(VbeInfoBlock.OEMStringPtr));
   printf("Available 256 color video modes:\n");
   for (p = VbeInfoBlock.VideoModePtr; *p !=(unsigned)-1; p++)
    {
       if (getModeInfo(*p))
        {
           printf("    %4d x %4d %d bits per pixel\n",
              ModeInfoBlock.XResolution, ModeInfoBlock.YResolution,
               ModeInfoBlock.BitsPerPixel);
        }
    }
   printf("\nUsage: hellovbe <xres> <yres>\n");
    exit(1);
}

/* Initialize the specified video mode. Notice how we determine a shift
 * factor for adjusting the Window granularity for bank switching. This
 * is much faster than doing it with a multiply (especially with direct
 * banking enabled).
 */
void initGraphics(unsigned int x, unsigned int y)
{
    unsigned far    *p;

   if (!getVbeInfo())
    {
       printf("No VESA VBE detected\n");
        exit(1);
    }
   for (p = VbeInfoBlock.VideoModePtr; *p != (unsigned)-1; p++)
    {
      if (getModeInfo(*p) && ModeInfoBlock.XResolution == x
              && ModeInfoBlock.YResolution == y)
        {
           xres = x;   yres = y;
         bytesperline = ModeInfoBlock.BytesPerScanLine;
          bankShift = 0;
         while ((unsigned)(64 >> bankShift) != ModeInfoBlock.WinGranularity)
              bankShift++;
          bankSwitch = ModeInfoBlock.WinFuncPtr;
           curBank = -1;
          screenPtr = (char far *)( ((long)0xA000)<<16 | 0);
          oldMode = getVBEMode();
          setVBEMode(*p);
           return;
        }
    }
   printf("Valid video mode not found\n");
    exit(1);
}
```

```
/* Main routine. Expects the x & y resolution of the desired video mode
 * to be passed on the command line. Will print out a list of available
 * video modes if no command line is present.
 */

void main(int argc,char *argv[])
{
    int x,y;

    if (argc != 3)
        availableModes();       /* Display list of available modes     */

    x = atoi(argv[1]);          /* Get requested resolution            */
    y = atoi(argv[2]);
    initGraphics(x,y);          /* Start requested video mode          */
    drawMoire();                /* Draw a moire pattern                */
    getch();                    /* Wait for keypress                   */
    setVBEMode(oldMode);        /* Restore previous mode               */
}

/*--------------------------------------------------------------------------*/
/* The following commented-out routines are for Planar modes                */
/* outpw() is for word output, outp() is for byte output                    */
/*--------------------------------------------------------------------------*/

/* Initialize Planar (Write mode 2)
 * Should be Called from initGraphics

void initPlanar()
{
  outpw(0x3C4,0x0F02);
  outpw(0x3CE,0x0003);
  outpw(0x3CE,0x0205);
}
*/

/* Reset to Write Mode 0
 * for BIOS default draw text

void setWriteMode0()
{
  outpw(0x3CE,0xFF08);
  outpw(0x3CE,0x0005);
}
*/

/* Plot a pixel in Planar mode

void putPixelP(int x, int y, int color)
{
    char dummy_read;

    long addr = (long)y * bytesperline + (x/8);
    setBank((int)(addr >> 16));
```

```
    outp(0x3CE,8);
    outp(0x3CF,0x80 >> (x & 7));
    dummy_read = *(screenPtr + (addr & 0xFFFF));
    *(screenPtr + (addr & 0xFFFF)) =  (char)color;
}
*/
```

Assembly Language Module

Below is the Assembly Language module required for the direct bank switching. In Borland C or other C compilers, this can be converted to in-line assembly code.

```
public _setbxdx
.MODEL SMALL           ;whatever
.CODE
set_struc      struc
        dw      ?       ;old bp
        dd      ?       ;return addr (always far call)
p_bx    dw      ?       ;reg bx value
p_dx    dw      ?       ;reg dx value
set_struc      ends

_setbxdx       proc far     ; must be FAR
        push    bp
        mov     bp,sp
        mov     bx,[bp]+p_bx
        mov     dx,[bp]+p_dx
        pop     bp
        ret
_setbxdx       endp
END
```

A5.2 Implementation Notes by Function

A5.2.1 General Notes

Starting with VBE version 2.0 VESA will no longer define new VESA mode numbers and it will not longer be mandatory to support these old mode numbers. VBE 2.0 aware applications should follow the guidelines in the sample code above for setting a desired mode.

Applications should treat any non-zero value in the AH register as a general failure condition as later versions of the VBE may define additional error codes. BIOS developers should refrain from defining their own return codes, which may conflict with future VESA-defined return codes.

A5.2.2 Function 00h - Return VBE Controller Information

All data in the structure is subject to change by the VBE implementation when VBE Function 00h is called. Therefore it should not be used by the application to store data of any kind. VBE should fill any unused portion of the structure with zeros.

The BCD value for VBE 2.0 is 0200h, The BCD value for VBE 1.2 is 0102h. In the past we have had some applications misinterpreting these BCD values. For example, BCD 0102h was interpreted as 1.02, which is incorrect.

If the RAMDAC is an older style RAMDAC with the possibility of "snow" during programming, the VBE 2.0 implementation must place a 1 in bit 2 of the Capabilities field.

It is the responsibility of the application to verify the actual availability of any mode returned by this Function by using the Return VBE Mode Information (VBE Function 01h) call. Some of the returned modes may not be available due to the actual amount of memory physically installed on the display board or to the capabilities of the attached monitor.

If a VideoModeList is found to contain no entries (starts with 0FFFFh), it can be assumed that the VBE implementation is a "stub" implementation where only Function 00h is supported for diagnostic or "Plug and Play" reasons. These stub implementations are not VBE 2.0 compliant and should only be implemented in cases where no space is available to implement the whole VBE. If a DAC is switchable, you can assume that the DAC will be restored to 6 bits per primary upon a mode set. For an application to use a DAC the application program is responsible for setting the DAC to 8 bits per primary mode using Function 08h.

If a DAC is switchable, you can assume that the DAC will be restored to 6 bits per primary upon a mode set. For an application to use a DAC the application program is responsible for setting the DAC to 8 bits per primary mode using Function 08h.

A5.2.3 Function 01h - Return VBE Mode Information

Version 1.1 and later VBE will zero out all unused fields in the Mode Information Block, always returning exactly 256 bytes. This facilitates upward compatibility with future versions of the standard, as any newly added fields will be designed such that values of zero will indicate nominal defaults or non-implementation of optional features. (For example, a field containing a bit-mask of extended capabilities would reflect the absence of all such capabilities.) Applications that wish to be backwards compatible to Version 1.0 VBE should pre-initialize the 256 byte buffer before calling the Return VBE Mode Information function.

Since this specification encompasses non-VGA hardware as well as VGA hardware, applications should not assume VGA properties, e.g., WinASegment and WinBSegment are not limited to the VGA frame buffer region A000-BFFFh, they may exist elsewhere.

A5.2.4 Function 02h - Set VBE Mode

Function 00h of an IBM VGA compatible BIOS uses D7 to signify the same thing as D15 does in this function. If D7 is set for an IBM compatible mode when calling this function, this mode set should fail. VBE aware applications must set the memory clear bit in D15.

This call should not set modes not listed in the list of supported modes. All modes (including IBM standard VGA modes), if listed as supported, must have ModeInfoBlock structures associated with them.

Mode 81FFh is a special mode designed to preserve the current memory contents and to give access to the entire video memory. This mode is especially useful for saving the entire video memory contents before going into a state that could lose the contents (e.g. set this mode to gain access to all video memory to save it before going into a volatile power down state). This mode is required as the entire video memory contents are not always accessible in every mode. It is recommended that this mode be packed pixel in format, and a ModeInfoBlock must be defined for it. Also note that there are no implied resolutions or timings associated with this mode.

A5.2.5 Function 03h - Return Current VBE Mode

Version 1.x Note: In a standard VGA BIOS, Function 0Fh (Read current video state) returns the current graphics mode in the AL register. In D7 of AL, it also returns the status of the memory clear bit (D7 of 40:87). This bit is set if the mode was set without clearing memory. In this VBE function, the memory clear bit will not be returned in BX since the purpose of the function is to return the video mode only. If an application wants to obtain the memory clear bit, it should call the standard VGA BIOS Function 0Fh.

Version 2.x Note: Unlike version 1.x VBE implementations, the memory clear flag will be returned. The application should NOT call the standard VGA BIOS Function 0Fh if the mode was set with VBE Function 02h.

This function is not guaranteed to return an accurate mode value if the mode set was not done with VBE Function 02h. In that case, the results are unspecified.

A5.2.6 Function 05h - Display Window Control

In VBE 1.2 implementations, the direct far call version returns no Return Status information to the application. Also, in the far call version, the AX and DX registers will be destroyed. Therefore if AX and/or DX must be preserved, the application must do so prior to making the far call. The application must still load the input arguments in BH, BL, and DX (for Set Window). In VBE 2.0 implementations, the BIOS will return the correct Return Status, and therefore the application must assume that AX and DX will be destroyed.

This function is not intended for use in a linear frame buffer mode, if this function is requested, the function call will fail with the VBE Completion code AH=03h.

A5.2.7 Function 06h - Get/Set Logical Scan Line Length

The desired width in pixels may not be achievable because of hardware considerations. The next larger value will be selected that will accommodate the desired number of pixels, and the actual number of

pixels will be returned in CX. BX returns a value that when added to a pointer into display memory will point to the next scan line. For example, in VGA mode 13h this would be 320, but in mode 12h this would be 80. DX returns the number of logical scan lines based upon the new scan line length and the total memory installed and usable in this display mode.

This function is also valid in text modes. In text modes the application should convert the character line length to pixel line length by getting the current character cell width through the XCharSize field returned in ModeInfoBlock, multiplying that times the desired number of characters per line, and passing that value in the CX register.

In text modes, this function will only work if the line length is specified in character granularity. i.e. in 8 dot modes only multiples of 8 will work. Any value which is not in character granularity will result in a function call failure.

A5.2.8 Function 07h - Get/Set Display Start

This function is also valid in text modes. To use this function in text mode, the application should convert the character coordinates to pixel coordinates by using XCharSize and YCharSize returned in the ModeInfoBlock. If the requested Display Start coordinates do not allow for a full page of video memory or the hardware does not support memory wrapping, the Function call should fail and no changes should be made. As a general case, if a requested Display Start is not available, fail the Function call and make no changes.

A5.2.9 Function 08h - Set/Get DAC Palette Format

An application can determine if DAC switching is available by querying Bit D0 of the Capabilities field of the VbeInfoBlock structure returned by VBE Function 00h (Return Controller Information). The application can then attempt to set the DAC palette width to the desired value. If the display controller hardware is not capable of selecting the requested palette width, then the next lower value that the hardware is capable of will be selected. The resulting palette width is returned.

This function is not intended for direct color modes, it will return failure code 03h if called in a direct color or YUV mode.

A5.2.10 Function 09h - Set/Get Palette Data

The need for BL= 80h is for older style RAMDAC's where programming the RAM values during display time causes a "snow-like" effect on the screen. Newer style RAMDAC's don't have this limitation and can easily be programmed at any time, but older RAMDAC's require that they be programmed during a non-display time only to stop the snow like effect seen when changing the DAC values. When this is requested the VBE implementation will program the DAC with blanking on. Check D2 of the Capabilities field returned by VBE Function 00h to determine if 80h should be used instead of 00h.

When in 6 bit mode, the format of the 6 bits is LSB, this is done for speed reasons, as the application can typically shift the data faster than the BIOS can.

All application should assume the DAC is defaulted to 6 bit mode. The application is responsible for switching the DAC to higher color modes using Function 08h.

Query VBE Function 08h to determine the RAMDAC width before loading a new palette.

A5.2.11 Function 0Ah - Return VBE Function Information

The Port and Memory location Sub-table does not include the Frame Buffer Memory location. The Frame Buffer Memory location is contained within the ModeInfoBlock returned by VBE Function 01h.

The protected mode code is assembled for a 32-bit code segment, when copying it, the application must copy the code to a 32-bit code segment.

It is the responsibility of the application to ensure that the selectors and segments are set up correctly.

If the memory location is zero, then only I/O mapped ports will be used so the application does not need to do anything special. This should be the default case for ALL cards that have I/O mapped registers because it provides the best performance.

If the memory location is nonzero (there can be only one), the application will need to create a new 32-bit selector with the base address that points to the "physical" location specified with the specified limit.

Applications must use the same registers for the Function 05h and Function 09h protected mode interface that it would use in a real mode call. This includes the AX register.

Function 07h protected mode calls have a different format.

AX	=	4F07h	
BL	=	00h	Set Display CRTC Start
	=	80h	Set Display CRTC Start during Vertical Retrace
CX	=		Bits 0-15 of display start address
DX	=		Bits 16-31 of display start address

The protected mode application must keep track of the color depth and scan line length to calculate the new start address. If a value that is out of range is programmed, unpredictable results will occur.

Currently undefined registers may be destroyed with the exception of ESI, EBP, DS and SS.

Appendix 6 - Differences Between VBE Revisions

6.1 VBE 1.0

Initial implementation: Implemented Functions 00-05h
Defined modes 100-107h

6.2 VBE 1.1

Second implementation:
- Added Functions 06h and 07h.
- Added modes 108-10Ch
- Added TotalMemory to VbeInfoBlock
- Added NumberOfImagePages and Reserved fields to ModeInfoBlock

6.3 VBE 1.2

Third implementation: Added Function 08h
- Added Hi-color modes 10D-11Bh
- Added Reserved field to VbeInfoBlock
- Added New Direct color fields to ModeInfoBlock
- Changed optional fields to mandatory in ModeInfoBlock
- Added Capabilities bit definition in VbeInfoBlock

6.4 VBE 2.0

Fourth implementation:
- Added Flat Frame Buffer support in Function 02h (D14)
- Added protected mode support (Function 0Ah)
- Added new DAC services for palette operations (Function 09h)
- Added new completion codes 02h and 03h
- Added OEM information to VbeInfoBlock
- Added two new definitions to Capabilities in VbeInfoBlock
- Added new fields to ModeInfoBlock
- Certification and ROM requirements for Compliance
- Clarified Memory Clear bit in Function 02h (D15)
- Clarified Memory Clear bit in Function 03h (D15)
- Added new return field in Function 06h
- Added Supplemental Functions definition and defined Supplemental Functions 10-16h
- Added new mode to access all of video memory
- Added wait for vertical retrace in Function 07h
- Clarified and removed ambiguities in the earlier

specifications
Added new mode to access all video memory.

6.5 VBE 2.0, Rev. 1.1

Current implementation:

Page 6, Section 3 - Revised sentence to read: Note that modes may only be set if the mode exists in the VideoModeList pointed to by the VideoModePTR returned in Function 00h. The exception to this requirement is the mode number 81ffh.

Page 11, Section 4.2 - Added: If the memory location is zero, then only I/O mapped ports will be used so the application does not need to do anything special. This should be the default case for ALL cards that have I/O mapped registers because it provides the best performance.
and
If the memory location is nonzero (there can be only one), the application will need to create a new 32-bit selector with the base address that points to the "physical" location specified with the specified limit.
and
When the application needs to call the 32-bit bank switch function, it must then load the ES selector with the value of the new selector that has been created. The bank switching code can then directly access its memory mapped registers as absolute offsets into the ES selector
(i.e., mov [es:10],eax to put a value into the register at base+10).

It is up to the application code to save and restore the previous state of the ES selector if this is necessary (for example in flat model code)

Page 25, Section 4.5 - Revised sentence to read: If function call D7 is set and the application assumes it is similar to the IBM compatible mode set using VBE Function 02h, the implementation will fail.

Page 29 - Added: **Note:** CX and DX, for both input and output values, will be zero based.

Page 33 - Added: If the memory location is zero, then only I/O mapped ports will be used so the application does not need to do anything special. This should be the default case for ALL cards that have I/O mapped registers because it provides the best performance.
and
If the memory location is nonzero (there can be only one), the application will need to create a new 32-bit selector with the base address that points to the "physical" location specified with the specified limit.
and
When the application needs to call the 32-bit bank switch function, it must then load the ES selector with the value of the new selector that has been created. The bank switching code can then directly access its memory mapped registers as absolute offsets into the ES selector (i.e., mov [es:10],eax to put a value into the register at base+10).

It is up to the application code to save and restore the previous state of the ES selector if this is necessary (for example in flat model code).

Page 69 - Added to first paragraph: However, it should not appear in the VideoModeList. Look in the ModeInfoBlock to determine if paging is required and, if it is required, how it is supported.

Page 88, Section A5.2.11 - Added: If the memory location is zero, then only I/O mapped ports will be used so the application does not need to do anything special. This should be the default case for ALL cards that have I/O mapped registers because it provides the best performance.
and
If the memory location is nonzero (there can be only one), the application will need to create a new 32-bit selector with the base address that points to the "physical" location specified with the specified limit.

Corrected typographical errors and style.

Modified copyright notice; modified Support section; added missing paragraphs regarding protected mode to function 0Ah and section on protected mode considerations; corrected typo

in function 09h – 255 should have read 256; corrected cast of 'color' in C example.

Appendix 7 - Related Documents

- VGA Reference Manual(s)
- Graphic Controller Data Sheets
- VESA Monitor Timings
- VBE/PM Monitor Power Management Standard
- VBE/AI VESA Audio Interface
- VBE/DDC VESA Display Data Channel Software Interface Standard
- VESA DDC Hardware Specification
- VESA DPMS Hardware Specification