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

Version: 2.0

NOTE: This is _NOT_ an official version of the VBE/Core 2.0 standard, but is provided as is by SciTech Software until the official specification is made available electronically. This document contains a summary of the important portions of the standard related to developing working code for VBE 2.0.

For more information on the official specification, please contact VESA at the address listed above.

1. Introduction

This document contains the VESA BIOS Extension (VBE) specification for standard software access to graphics display controllers that 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 & 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

[1]

The IBM VGA 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.

However, several serious problems face a software developer who intends to take advantage of these "Super VGA" environments. Because there is no standard hardware implementation, the developer is faced with widely disparate 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.

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.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 IBM 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 hardware 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 the underlying 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 the application to obtain additional information about each supported mode. Function 03h returns the current VBE mode.

3.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 in the range 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 Video Mode List returned in Function 0.

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 old 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 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 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. 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.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 00-0Fh have been reserved for Standard VBE function numbers; Functions 10-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: Software supports this function, but the hardware does not
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.

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 calling application's responsibility to ensure to that it has the appropriate I/O and memory privileges, a large enough stack and appropriate selectors allocated. It is also the calling application's responsibility to preserve registers if it needs them.

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 cont.
VideoModePtr        dd   ?         ; Pointer to Video Mode List
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 blank bit in Function 09h. i.e. RAMDAC
               recommends programming during blank period only.
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 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 the that they be blanked so as not to display the snow while values change during display time.. This bit informs the software that they should make the function call with 80h rather than 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. Please 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 in 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 the application's responsibility 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 whole 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 addressible 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 who 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 Video Mode List 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 chars
YResolution         dw ?      ; vertical resolution in pixels or chars
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 ?      ; 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 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 4 Linear	1	0
Linear frame buffer only	1	1

The BytesPerScanLine field specifies how many full bytes are in each logical scanline. The logical 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
```

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.

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

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 will 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 don't 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, returning AH=01h to the application.

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 D7 is set for an IBM compatible mode set using this Function (02), this mode set 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.

Note: If there is a failure when selecting an unsupported D14 value, the error return should be 02h.

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 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 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.

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
          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

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 (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 application's responsibility 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.

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

---------- END OF DOCUMENT ----------