Getting VBE Mode Info

From OSDev Wiki

Contents

VESA Functions

You'll want to look in the VESA VBE docs for these functions: All VESA functions return 0x4F in AL if they are supported and use AH as a status flag, with 0x00 being success. This means that you should check that AX is 0x004F after each VESA call to see if it succeeded.

INT 0x10, AX=0x4F00
: Get Controller Info. This is the one that returns the array of all supported video modes.

```
struct VbeInfoBlock {
   char VbeSignature[4];             // == "VESA"
   uint16_t VbeVersion;                 // == 0x0300 for VBE 3.0
   uint16_t OemStringPtr[2];            // isa vbeFarPtr
   uint8_t Capabilities[4];
   uint16_t VideoModePtr[2];         // isa vbeFarPtr
   uint16_t TotalMemory;             // as # of 64KB blocks
};

VbeInfoBlock *vib = dos_alloc(512);
v86_bios(0x10, {ax:0x4f00, es:SEG(vib), di:OFF(vib)}, &out);
if (out.ax!=0x004f) die("Something wrong with VBE get info");
```

INT 0x10, AX=0x4F01, CX=mode
: Get Mode Info. Call this for each member of the mode array to find out the details of that mode.

```
struct ModeInfoBlock {
  uint16_t attributes;
  uint8_t winA,winB;
  uint16_t granularity;
  uint16_t winsize;
  uint16_t segmentA, segmentB;
  VBE_FAR(realFctPtr);
  uint16_t pitch; // bytes per scanline

  uint16_t Xres, Yres;
  uint8_t Wchar, Ychar, planes, bpp, banks;
  uint8_t memory_model, bank_size, image_pages;
  uint8_t reserved0;

  uint8_t red_mask, red_position;
  uint8_t green_mask, green_position;
  uint8_t blue_mask, blue_position;
  uint8_t rsv_mask, rsv_position;
  uint8_t directcolor_attributes;

  uint32_t physbase;  // your LFB (Linear Framebuffer) address ;)
  uint32_t reserved1;
  uint16_t reserved2;
};
```

INT 0x10, AX=0x4F02, BX=mode
: Set Video Mode. Call this with the mode number you decide to use. If you choose a mode that makes use of a linear framebuffer, you should OR the mode number with 0x4000. This sets the "Use LFB" bit in the mode number.

Will it work with Bochs?

For VBE to work in Bochs you need the "VGABIOS-lgpl" BIOS and have a version of Bochs that was compiled with the `--enable-vbe` option... See Vesa Information in Bochs thread for more info. Also read about Bochs Graphics Adaptor.

How to pick the mode I wish ?

VESA stopped assigning codes for video modes long ago -- instead they standardized a much better solution: you can query the video card for what modes it supports, and query it about the attributes of each mode. In your OS, you can have a function that you call with a desired width, height, and depth, and it returns the video mode number for it (or the closest match). Then, just set that mode

Here's a sample code, assuming you have a VirtualMonitor already ... Basically, you will scan the 'modes list' referenced by the VbeInfoBlock.videomodes[] and then call 'get mode info' for each mode. You can then compare width, height and colordepth of each mode with the desired one.

```
uint16_t findMode(int x, int y, int d)
```

```
{
  struct VbeInfoBlock *ctrl = (VbeInfoBlock *)0x2000;
  struct ModeInfoBlock *inf = (ModeInfoBlock *)0x3000;
  uint16_t *modes;
  int i;
  uint16_t best = 0x13;
  int pixdiff, bestpixdiff = DIFF(320 * 200, x * y);
  int depthdiff, bestdepthdiff = 8 >= d ? 8 - d : (d - 8) * 2;

  strncpy(ctrl->VbeSignature, "VBE2", 4);
  intV86(0x10, "ax,es:di", 0x4F00, 0, ctrl); // Get Controller Info
  if ( (uint16_t)v86.tss.eax != 0x004F ) return best;

  modes = (uint16_t*)REALPTR(ctrl->VideoModePtr);
  for ( i = 0 ; modes[i] != 0xFFFF ; ++i ) {
      intV86(0x10, "ax,cx,es:di", 0x4F01, modes[i], 0, inf); // Get Mode

      if ( (uint16_t)v86.tss.eax != 0x004F ) continue;

      // Check if this is a graphics mode with linear frame buffer suppor
      if ( (inf->attributes & 0x90) != 0x90 ) continue;

      // Check if this is a packed pixel or direct color mode
      if ( inf->memory_model != 4 && inf->memory_model != 6 ) continue;

      // Check if this is exactly the mode we're looking for
      if ( x == inf->XResolution && y == inf->YResolution &&
          d == inf->BitsPerPixel ) return modes[i];

      // Otherwise, compare to the closest match so far, remember if best
      pixdiff = DIFF(inf->Xres * inf->Yres, x * y);
      depthdiff = (inf->bpp >= d)? inf->bpp - d : (d - inf->bpp) * 2;
      if ( bestpixdiff > pixdiff ||
          (bestpixdiff == pixdiff && bestdepthdiff > depthdiff) ) {
        best = modes[i];
        bestpixdiff = pixdiff;
        bestdepthdiff = depthdiff;
      }
  }
  if ( x == 640 && y == 480 && d == 1 ) return 0x11;
  return best;
}
```

Common Mistakes

There's some mistakes that beginners seem to make fairly often when they first start working with VBE.

VESA Defined Mode Numbers

For older versions of the VBE specification (VBE 1.0, VBE 1.1 and VBE 1.2) VESA defined some standard mode numbers. For example, mode 0x0113 was 800 * 600 * 15-bpp. In VBE 2.0 these mode numbers were deprecated - VESA decided not to define any more of them, told video card manufacturers they don't need to use the old standard mode numbers, and told programmers to search for the mode they want without relying on any of the old standard mode numbers. Despite this, it's still possible to find obsolete information on the internet suggesting to use these old standard mode numbers.

Use "Bytes Between Lines"

People tend to assume that pixel data is contiguous (e.g. line 1, then line 2, then line 3, ...). This isn't always the case, and for a variety of reasons there may be padding between lines (e.g. line 1, padding, then line 2, padding, then line 3, ...). The "Get Mode Information" VBE function returns the number of bytes between lines, and this value should be used. For example, for a 640 * 480 * 16-bpp video mode, the offset of a line is found by "offset = line * bytes_per_line" and not by "offset = line * (640 * 2)". In the same way, you can't use one "rep stosb/w/d/q" or "memset()" to fill the entire display, and you should fill each line separately to avoid writing to any padding.

Note that (for VBE 3.0) there may be 2 "bytes between lines" values returned by the "Get Mode Information" VBE function. The first one is used when the video mode is setup for bank switching and the second one is used when linear frame buffer is being used. For older versions of VBE there is no "bytes between lines for linear frame buffer" value, and the "bytes between lines for bank switched" value is used for both bank switching and linear frame buffer.

Don't Assume Pixel Formats

Usually pixel data is in "RGB" format, but this isn't always the case. A pixel can be in "BGR" format or anything else, and may not even use Red, Green and Blue components - it could use "YUV" where there's one luma/brightness component (Y) and two chrominance (UV) components. Always check the "memory_model" field and (for 15-bpp and higher video modes) all of the "component mask" and "field position" fields (e.g. "red_mask_size", "red_field_position", etc) in the structure returned by the "Get Mode Information" VBE function.

Don't Assume Unused Bits Are Unused

For some colour depths there's "unused" bits in the pixel data (one "unused" bit per pixel in 15-bpp modes, and 8 "unused" bits per pixel for 32-bpp modes). These unused bits may actually be used by some video cards for a variety of extra features. For one example, I know of a video card where (in 15-bpp modes) if the highest/unused bit is set then it uses the lowest 8 bits of the pixel data as an index into the palette.

Don't Assume VGA Compatibility

For 8-bpp video modes people assume that they can use the VGA I/O ports to change the palette. It's much better to use the VBE functions instead, but if you don't then at least test if the video mode uses VGA I/O ports by testing the "VGA compatible mode" flag (bit 5) in "mode_attributes" field returned by the "Get Mode Information" VBE function.

In a similar way, for 4-bpp "planar" video modes some people assume that the VGA I/O ports can be used to switch between planes rather than testing if the "VGA compatible mode" flag is set first.

Don't Assume The Monitor Supports A Video Mode

We all like pretty graphics and high resolution video modes. Unfortunately, if VBE says that a video mode is supported by the video card it does not mean that the video mode is also supported by the monitor. VESA has defined 2 video mode timings that are meant to be supported by all monitors (640 * 480 standard VGA timing and 720 * 480 standard VGA timing). For all other video modes you should either use EDID to find out if the monitor supports the video mode's timing (or not), or provide a way for the user to test the video mode and change it if it doesn't work. This is the approach used by Windows, with a dialog box appearing with the option to accept or revert. No action within 15 seconds reverts. Just an idea ;).

Using EDID for this purpose is complicated. Unless you provide a "CRTC information block" structure when you set the video mode you can't be entirely sure what timing the video card will use; and only some video cards that support VBE 3.0 support the "CRTC information" correctly.

The other alternative is to only ever use video modes that (should) use 640 * 480 and 720 * 480 standard VGA timing. For video modes with lower horizontal resolutions the video card sends the each pixel twice, and for video modes with lower vertical resolutions the video card sends the each row of pixels twice (called "double scanning"); and in both cases the video timing is the same as it would be for the corresponding higher resolution video mode. This gives the following list of "safe" resolutions:

- 720 * 480
- 640 * 480
- 360 * 480 (actually uses "720 * 480" timing)
- 320 * 480 (actually uses "640 * 480" timing)
- 720 * 240 (actually uses "720 * 480" timing)
- 640 * 240 (actually uses "640 * 480" timing)
- 360 * 240 (actually uses "720 * 480" timing)
- 320 * 240 (actually uses "640 * 480" timing)

Colour depth doesn't/shouldn't effect the video timing signals. This means that the best possible safe video mode would be 720 * 480 * 24/32-bpp.

See Also

Threads

- VESA, higher modes - Initial Thread, reply by Dreamsmith (aka DaidalosGuy)

External Links

- VBE at Wikipedia (http://en.wikipedia.org/wiki/VESA_BIOS_Extensions)
- Guide: VESA graphics modes (http://www.delorie.com/djgpp/doc/ug/graphics/vesa.html)
- The VESA VBE 3.0 specification (http://www.petesqbsite.com/sections/tutorials/tuts/vbe3.pdf)