The Frame Buffer Device API

Last revised: June 21, 2011

0. Introduction

This document describes the frame buffer API used by applications to interact with frame buffer devices. In-kernel APIs between device drivers and the frame buffer core are not described.

Due to a lack of documentation in the original frame buffer API, drivers behaviours differ in subtle (and not so subtle) ways. This document describes the recommended API implementation, but applications should be prepared to deal with different behaviours.

1. Capabilities

Device and driver capabilities are reported in the fixed screen information capabilities field:

```
struct fb_fix_screeninfo {
        ...
        __u16 capabilities;             /* see FB_CAP_*                 */
        ...
};
```

Application should use those capabilities to find out what features they can expect from the device and driver.

- FB_CAP_FOURCC

The driver supports the four character code (FOURCC) based format setting API. When supported, formats are configured using a FOURCC instead of manually specifying color components layout.

2. Types and visuals

Pixels are stored in memory in hardware-dependent formats. Applications need to be aware of the pixel storage format in order to write image data to the frame buffer memory in the format expected by the hardware.

Formats are described by frame buffer types and visuals. Some visuals require additional information, which are stored in the variable screen information bits_per_pixel, grayscale, red, green, blue and transp fields.

Visuals describe how color information is encoded and assembled to create macropixels. Types describe how macropixels are stored in memory. The following types and visuals are supported.

- FB_TYPE_PACKED_PIXELS

Macropixels are stored contiguously in a single plane. If the number of bits per macropixel is not a multiple of 8, whether macropixels are padded to the next multiple of 8 bits or packed together into bytes depends on the visual.

Padding at end of lines may be present and is then reported through the fixed screen information line_length field.

- FB_TYPE_PLANES

Macropixels are split across multiple planes. The number of planes is equal to the number of bits per macropixel, with plane i'th storing i'th bit from all macropixels.

Planes are located contiguously in memory.

- FB_TYPE_INTERLEAVED_PLANES

Macropixels are split across multiple planes. The number of planes is equal to the number of bits per macropixel, with plane i'th storing i'th bit from all macropixels.

Planes are interleaved in memory. The interleave factor, defined as the distance in bytes between the beginning of two consecutive interleaved blocks belonging to different planes, is stored in the fixed screen information type_aux field.

- FB_TYPE_FOURCC

Macropixels are stored in memory as described by the format FOURCC identifier stored in the variable screen information grayscale field.

- FB_VISUAL_MONO01

Pixels are black or white and stored on a number of bits (typically one) specified by the variable screen information bpp field.

Black pixels are represented by all bits set to 1 and white pixels by all bits set to 0. When the number of bits per pixel is smaller than 8, several pixels are packed together in a byte.

FB_VISUAL_MONO01 is currently used with FB_TYPE_PACKED_PIXELS only.

- FB_VISUAL_MONO10

Pixels are black or white and stored on a number of bits (typically one) specified by the variable screen information bpp field.

Black pixels are represented by all bits set to 0 and white pixels by all bits set to 1. When the number of bits per pixel is smaller than 8, several pixels are packed together in a byte.

FB_VISUAL_MONO01 is currently used with FB_TYPE_PACKED_PIXELS only.

- FB_VISUAL_TRUECOLOR

Pixels are broken into red, green and blue components, and each component indexes a read-only lookup table for the corresponding value. Lookup tables are device-dependent, and provide linear or non-linear ramps.

Each component is stored in a macropixel according to the variable screen information red, green, blue and transp fields.

- FB_VISUAL_PSEUDOCOLOR and FB_VISUAL_STATIC_PSEUDOCOLOR

Pixel values are encoded as indices into a colormap that stores red, green and blue components. The colormap is read-only for FB_VISUAL_STATIC_PSEUDOCOLOR and read-write for FB_VISUAL_PSEUDOCOLOR.

Each pixel value is stored in the number of bits reported by the variable screen information bits_per_pixel field.

- FB_VISUAL_DIRECTCOLOR

Pixels are broken into red, green and blue components, and each component indexes a programmable lookup table for the corresponding value.

Each component is stored in a macropixel according to the variable screen information red, green, blue and transp fields.

- FB_VISUAL_FOURCC

Pixels are encoded and interpreted as described by the format FOURCC identifier stored in the variable screen information grayscale field.

3. Screen information

Screen information are queried by applications using the FBIOGET_FSCREENINFO and FBIOGET_VSCREENINFO ioctls. Those ioctls take a pointer to a fb_fix_screeninfo and fb_var_screeninfo structure respectively.

struct fb_fix_screeninfo stores device independent unchangeable information about the frame buffer device and the current format. Those information can't be directly modified by applications, but can be changed by the driver when an application modifies the format:

```
struct fb_fix_screeninfo {
      char id[16];                    /* identification string eg "TT Builtin" */
      unsigned long smem_start;       /* Start of frame buffer mem */
                                      /* (physical address) */
      __u32 smem_len;                 /* Length of frame buffer mem */
      __u32 type;                     /* see FB_TYPE_*                */
      __u32 type_aux;                 /* Interleave for interleaved Planes */
      __u32 visual;                   /* see FB_VISUAL_*              */
      __u16 xpanstep;                 /* zero if no hardware panning  */
      __u16 ypanstep;                 /* zero if no hardware panning  */
      __u16 ywrapstep;                /* zero if no hardware ywrap    */
      __u32 line_length;              /* length of a line in bytes    */
      unsigned long mmio_start;       /* Start of Memory Mapped I/O   */
                                      /* (physical address) */
      __u32 mmio_len;                 /* Length of Memory Mapped I/O  */
      __u32 accel;                    /* Indicate to driver which     */
                                      /*  specific chip/card we have  */
      __u16 capabilities;             /* see FB_CAP_*                 */
      __u16 reserved[2];              /* Reserved for future compatibility */
};
```

struct fb_var_screeninfo stores device independent changeable information about a frame buffer device, its current format and video mode, as well as other miscellaneous parameters:

```
struct fb_var_screeninfo {
      __u32 xres;                     /* visible resolution           */
      __u32 yres;
      __u32 xres_virtual;             /* virtual resolution           */
      __u32 yres_virtual;
      __u32 xoffset;                  /* offset from virtual to visible */
      __u32 yoffset;                  /* resolution                   */

      __u32 bits_per_pixel;           /* guess what                   */
      __u32 grayscale;                /* 0 = color, 1 = grayscale,    */
                                      /* >1 = FOURCC                  */
      struct fb_bitfield red;         /* bitfield in fb mem if true color, */
      struct fb_bitfield green;       /* else only length is significant */
      struct fb_bitfield blue;
      struct fb_bitfield transp;      /* transparency                 */
```

```
        __u32 nonstd;                   /* != 0 Non standard pixel format */

        __u32 activate;                 /* see FB_ACTIVATE_*            */

        __u32 height;                   /* height of picture in mm    */
        __u32 width;                    /* width of picture in mm     */

        __u32 accel_flags;              /* (OBSOLETE) see fb_info.flags */

        /* Timing: All values in pixclocks, except pixclock (of course) */
        __u32 pixclock;                 /* pixel clock in ps (pico seconds) */
        __u32 left_margin;              /* time from sync to picture    */
        __u32 right_margin;             /* time from picture to sync    */
        __u32 upper_margin;             /* time from sync to picture    */
        __u32 lower_margin;
        __u32 hsync_len;                /* length of horizontal sync    */
        __u32 vsync_len;                /* length of vertical sync      */
        __u32 sync;                     /* see FB_SYNC_*                */
        __u32 vmode;                    /* see FB_VMODE_*               */
        __u32 rotate;                   /* angle we rotate counter clockwise */
        __u32 colorspace;               /* colorspace for FOURCC-based modes */
        __u32 reserved[4];              /* Reserved for future compatibility */
};
```

To modify variable information, applications call the FBIOPUT_VSCREENINFO ioctl with a pointer to a fb_var_screeninfo structure. If the call is successful, the driver will update the fixed screen information accordingly.

Instead of filling the complete fb_var_screeninfo structure manually, applications should call the FBIOGET_VSCREENINFO ioctl and modify only the fields they care about.

4. Format configuration

Frame buffer devices offer two ways to configure the frame buffer format: the legacy API and the FOURCC-based API.

The legacy API has been the only frame buffer format configuration API for a long time and is thus widely used by application. It is the recommended API for applications when using RGB and grayscale formats, as well as legacy non-standard formats.

To select a format, applications set the fb_var_screeninfo bits_per_pixel field to the desired frame buffer depth. Values up to 8 will usually map to monochrome, grayscale or pseudocolor visuals, although this is not required.

- For grayscale formats, applications set the grayscale field to one. The red, blue, green and transp fields must be set to 0 by applications and ignored by drivers. Drivers must fill the red, blue and green offsets to 0 and lengths to the bits_per_pixel value.
- For pseudocolor formats, applications set the grayscale field to zero. The red, blue, green and transp fields must be set to 0 by applications and ignored by drivers. Drivers must fill the red, blue and green offsets to 0 and lengths to the bits_per_pixel value.
- For truecolor and directcolor formats, applications set the grayscale field to zero, and the red, blue, green and transp fields to describe the layout of color components in memory:

  ```
  struct fb_bitfield {
      __u32 offset;                   /* beginning of bitfield        */
      __u32 length;                   /* length of bitfield           */
      __u32 msb_right;                /* != 0 : Most significant bit is */
                                      /* right */
  };
  ```

  Pixel values are bits_per_pixel wide and are split in non-overlapping red, green, blue and alpha (transparency) components. Location and size of each component in the pixel value are described by the fb_bitfield offset and length fields. Offset are computed from the right.

  Pixels are always stored in an integer number of bytes. If the number of bits per pixel is not a multiple of 8, pixel values are padded to the next multiple of 8 bits.

Upon successful format configuration, drivers update the fb_fix_screeninfo type, visual and line_length fields depending on the selected format.

The FOURCC-based API replaces format descriptions by four character codes (FOURCC). FOURCCs are abstract identifiers that uniquely define a format without explicitly describing it. This is the only API that supports YUV formats. Drivers are also encouraged to implement the FOURCC-based API for RGB and grayscale formats.

Drivers that support the FOURCC-based API report this capability by setting the FB_CAP_FOURCC bit in the fb_fix_screeninfo capabilities field.

FOURCC definitions are located in the linux/videodev2.h header. However, and despite starting with the V4L2_PIX_FMT_prefix, they are not restricted to V4L2 and don't require usage of the V4L2 subsystem. FOURCC documentation is available in Documentation/userspace-api/media/v4l/pixfmt.rst.

To select a format, applications set the grayscale field to the desired FOURCC. For YUV formats, they should also select the appropriate colorspace by setting the colorspace field to one of the colorspaces listed in linux/videodev2.h and documented in Documentation/userspace-api/media/v4l/colorspaces.rst.

The red, green, blue and transp fields are not used with the FOURCC-based API. For forward compatibility reasons applications must zero those fields, and drivers must ignore them. Values other than 0 may get a meaning in future extensions.

Upon successful format configuration, drivers update the fb_fix_screeninfo type, visual and line_length fields depending on the selected format. The type and visual fields are set to FB_TYPE_FOURCC and FB_VISUAL_FOURCC respectively.