Open Firmware

Recommended Practice:

8-bit Graphics Extension

Version 1.2 (draft)

October 26, 1995 1:59 pm

Published by the Open Firmware Working Group

Revision History

Version 1.0	Original
Version 1.1	used template. minor clean-up. Added 16-color Text Extension defaults.
Version 1.2	fixed spelling of **`color!`**, **`color@`**, added **`dimensions`**, as per 7/18/95 meeting minor fixes.

1. Introduction

1.1. Purpose

A set of "bit-mapped" graphics methods (to be implementated by display device packages) is defined by this extension (`draw-rectangle`, `fill-rectangle`, `read-rectangle`) that are available to Client Programs (via the `call-method` client callback) to display and read rectangular areas of the display buffer. Also defined by this extension are display device package methods (`color!`, `color@`, `set-colors`, `get-colors`) that control the mapping of pixel values to display colors. A query method (`dimensions`) is defined that returns the current viewable size of the display.

> Note: a Client Program can detect whether the display package implements this extension by doing a `catch` of a `call-method` of `0 color@`. If the package does not implement this extension, an error will be indicated by a non-zero `throw` value.

1.2. Scope

This extension applies to Open Firmware display device packages and to Client programs that require access to these extensions.

2. References and Definitions

2.1. References

[1] *IEEE Std 1275-1994, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices*, published by IEEE.

[2] *Open Firmware Recommended Practice: 16-color Text Extension*, published by Open Firmware Working Group.

2.2. Definitions

color-number: a value that represents a color.

display device package: An Open Firmware package whose **`"device_type"`** value is **`"display"`** and that implements the methods and properties defined in section 3.7.1 of [1].

pixel-map: a set of contiguous bytes in memory that represents a rectangular region of pixels that can be drawn to (or, read from) the display buffer.

3. Graphics model

The model used by this extension is that pixels are represented by 8-bit values (color-numbers) that specify one of 256 colors. The mapping of a color-number to a particular display color is specified by means of 8-bit R-G-B components. The `color!` and `color@` methods pass the R-G-B components via the stack. The `set-colors` and `get-colors` methods reference a range of color values by means of a memory area where each color value consists of contiguous bytes where the first byte is the Red component, the second byte is the Green component and the third byte is the Blue component. A component value of 0 represents absence of that color, while a value of 255 represents full saturation of the component; i.e., 0,0,0 is black and 255,255,255 is white. When multiple color values are present, the Red component of the second color value immediately follows the Blue component of the first color value, and so on.

Rectangular regions of this display buffer are represented by a set of coordinates that specify the position of the top-left pixel (`x,y`) and its width and height (`w,h`). The top-left pixel of the display has the coordinate `0,0`. Data in memory that represent pixel-maps consist of `w*h` contiguous bytes, where the first `w` bytes represent the pixels of the first row of the rectangle (with the first byte representing the left-most pixel), the next `w` bytes represent the next row, etc. Each such byte contains the color-number of the corresponding display buffer pixel.

4. Requirements

A display device package that implements this extension *shall* provide all of the following methods:

4.1. Pixel-mapped graphics methods

`draw-rectangle` `( adr x y w h -- )` M

Draw a rectangle in the display buffer from a pixel-map.

Displays a rectangular image at pixel location *`x,y`* of size *`w,h`* from the pixel-map defined by *`adr`*. *`x,y`* are display coordinates (where `0,0` corresponds to the `top,left` displayed pixel); *`w,h`* are the width and height of the image.

`fill-rectangle` `( number x y w h -- )` M

Fill a rectangle in the display buffer with a constant color.

Fills a rectangular display area, defined by *`x,y,w,h`* with the color corresponding to *`number`*.

`read-rectangle` `( adr x y w h -- )` M

Read a pixel-map from a rectangle in the display buffer.

This method copies the 8-bit number values corresponding to the display pixels , defined by *`x,y,w,h`* into the buffer specified by *`adr`*.

Note: for displays that are not in 8-bit per pixel mode, the read-rectangle method is undefined. For this reason, it is recommended that displays provide an 8-bit mode and that this mode is used during Open Firmware execution.

4.2. Color-value specification methods

The following methods allow a Client program to control the color mapping:

`color!` `( r g b number -- )` M

Set the color corresponding to *`number`* to the value specified by *`r, g, b`*.

`color@` `( number -- r g b )` M

Read the color value correspoding to *`number`*, returning its *`r,g,b`* components.

`set-colors` `( adr number #numbers -- )` M

Set a range of *`#numbers`* consecutive colors, starting with *`number`*. *`adr`* is the address of the memory area from which the color components are specified.

`get-colors` `( adr number #numbers -- )` M

Read the color values of *`#numbers`* consecutive color values, starting with *`number`*. *`adr`* is the address of a memory area into which the R-G-B components will be copied.

4.3. Geometry query method

`dimensions` `( -- width height )` M

Returns visible *`width`* and *`height`* in pixels.

The returned values are the visible dimensions of the screen in the current mode.

Note: Client programs may test for the existence of this method, and if it does not exist, use the `"width"` and `"height"` properties. If the method does exist, its return values may differ from the static property values.

4.4. 16-Color Text Extension defaults

If a display device driver implements both the 8-bit Graphics Extension and the 16-color Text Extension, it *shall* initialize the first 16 color numbers to correspond to the colors indicated in Table 1 in the 16-color Text Extension. A client program can use those colors without explicitly initializing them. If a client program changes the color assignment for color numbers 0-15, the behavior of the 16-color Text Extension might be affected.