Graphics Abstraction Manual

WEBC GRAPHICS PORTING GUIDE

Revised September 2006

EBS Inc. 39 Court Street Groton MA 01450 USA
http://www.ebsembeddedsoftware.com

Table of Contents

Introduction

The WebC graphics abstraction layer allows version 2.5 and above to be ported with minimal effort to any graphics platform which provides a minimum of rendering services. It does this by providing a common interface to the underlying graphics system.

Porting Requirements
To port WebC to a graphics platform, the graphic platform must provide the following minimum rendering services:

1. Filled rectangle drawing
2. Bitmap drawing
3. Text drawing
4. Clipping of all drawing operations to an arbitrary rectangular region
5. Buffering of drawing operations to an off-screen frame buffer

In addition the platform must provide the following information services:

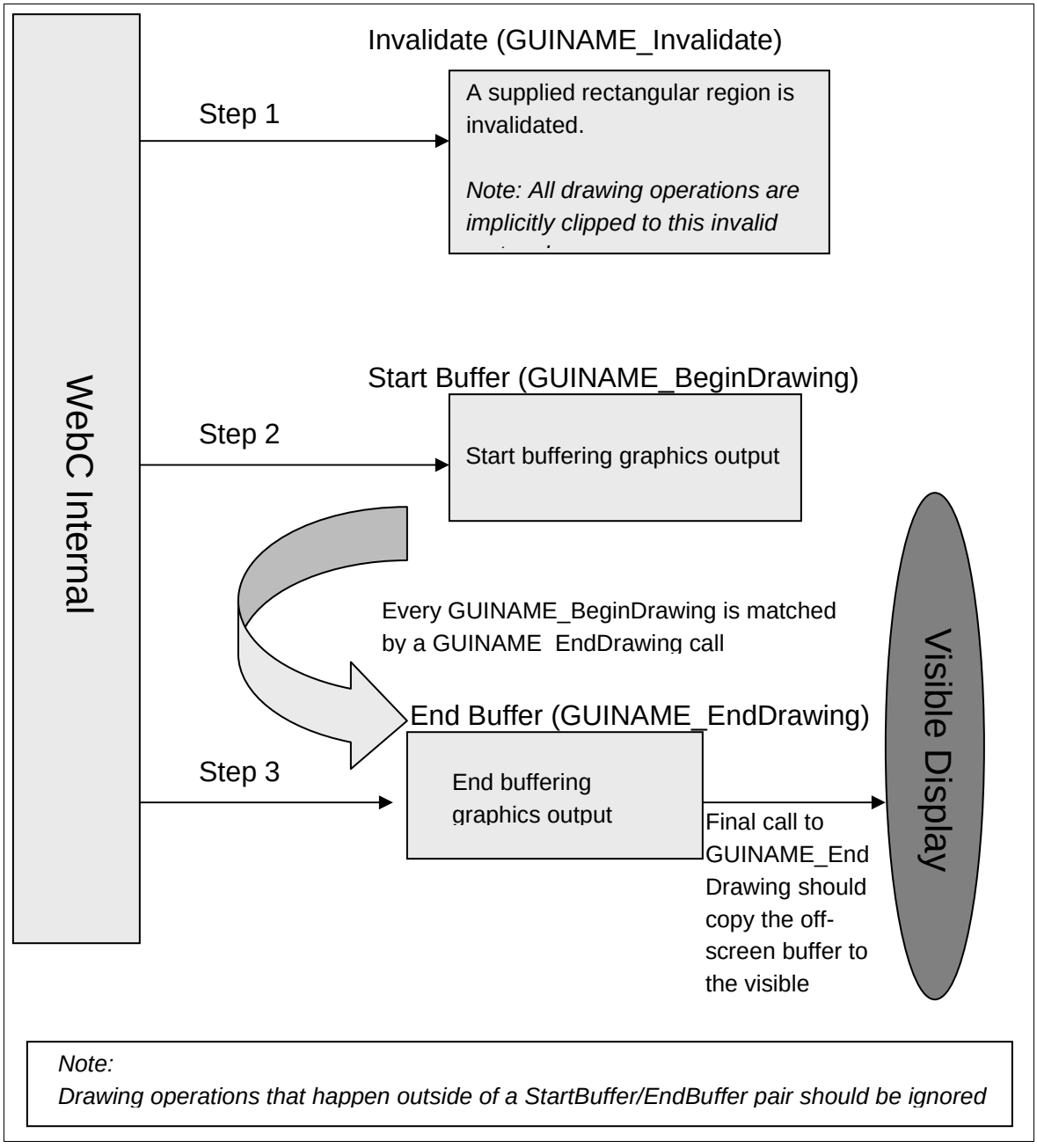
6. Color lookup (from RGB to platform-specific color value and vice versa)
7. Find the pixel width and height of a given text string rendered using a given font
8. Get the current clipping region
9. Get the bounding rectangle of the graphics display device.

In order to support the full feature set of WebC's HTML rendering engine, the graphics platform must also provide the following services:

10. Scaling of bitmaps (to arbitrary width and height) directly onto the frame buffer
11. Scaling of bitmaps (to arbitrary width and height) into a newly created bitmap

WebC Drawing Model

The WebC drawing model performs the following sequence of operations:



The clipping region is set at any time by GUINAME_SetClipRect ( ). All drawing operations must always be clipped to the rectangle specified at the most recent call to GUINAME_SetClipRect ( ). Clipping region is saved using GUINAME_GetClipRect ( ) and restored using GUINAME_SetClipRect ( ).

Operation Sequence
1. Invalidate drawing region (GUINAME_InvalidateRect) 2. Start buffering graphics output (GUINAME_BeginDrawing) 3. Perform actual drawing operations (which affect off-screen buffer) 4. End buffering graphics output (GUINAME_EndDrawing) - Copies the off-screen buffer onto the visible display

WebC Data Types

WebChar - This is the type specifier for characters. WebChar * is used for strings passed to the abstraction layer. If WebC is configured to support Unicode (WEBC_SUPPORT_UNICODE == 1) then this is an unsigned 16 bit value, otherwise it is the "char" type which may be signed or unsigned, depending on the compiler.

DISPLAY_INT - Signed 32 bit integer used internally by WebC to represent screen coordinates and dimensions. Screen coordinates dimensions such as height and width are passed to the graphics abstraction layer and returned from the abstraction layer using this type.

HTMLRect - WebC's internal representation of a rectangle. Rectangular coordinates are passed to and from the abstraction layer using this type.

HTMLRect Type

```
typedef struct s_HTMLRect
{
    DISPLAY_INT left;
    DISPLAY_INT top;
    DISPLAY_INT right;
    DISPLAY_INT bottom;
} HTMLRect;
```

HTMLColor - Unsigned 32 bit integer used by WebC to represent colors in RGBA. WebC provides arguments for RGB color values to the graphics abstraction layer in the HTMLColor RGBA format. If the underlying graphics driver uses a different format to represent RGBA the values must be remapped to the correct format.

HTMLColor	
Bits 0-7	BLUE
Bits 8-15	GREEN
Bits 16-23	RED
Bits 24-31	ALPHA

WebColor - Unsigned 32 bit integer used by WebC to a set of specific stock colors. WebC makes no assumptions about the format of WebColor. It asks GUINAME_GetColorByIndex() to return an appropriate 32 bit color "handle" for 16 stock colors and transparencies. (For example, RED, GREEN, BROWN, BLUE etc.). The returned 32 bit value is then later passed to the graphics abstraction layer as a color argument to draw functions. (For example to draw a BLUE rectangle WebC will call GUINAME_GetColorByIndex() to get a native 32 bit representation for

BLUE and then call GUINAME_DrawRectangle() using the 32 bit color identifier as one of it's argument. The graphics abstraction layer must also provide functions to convert native WebColor values to RGB and from RGB to native color representation. GUINAME_ColorToRGB() and GUINAME_RGBToColor() respectively.

WebcKeyMap guinameKeyMap;

CSSBorderStyle - WebC encodes border styles in this format and passes it to the abstraction layer when a frame rectangle draw is requested. The following values are possible:

Border Styles
CSS_BORDER_STYLE_DASHED CSS_BORDER_STYLE_DOTTED CSS_BORDER_STYLE_DOUBLE CSS_BORDER_STYLE_GROOVE CSS_BORDER_STYLE_INSET CSS_BORDER_STYLE_NONE CSS_BORDER_STYLE_OUTSET CSS_BORDER_STYLE_RIDGE CSS_BORDER_STYLE_SOLID

WebGraphBitmapInfo - This structure is how WebC represents bitmaps and pixmaps internally. WebC converts pixel maps like .png files, .jpg files and .gif files to this internal format. This structure is then provided to the abstraction layer's GUINAME_CreateWebBitmap() bitmap function to convert to an internal native representation of the bitmap. When WebC calls the abstraction layer's other bitmap manipulation functions it passes to them the address of the native bitmap or pixmap structure that was created by GUINAME_CreateWebBitmap().

The following abstraction layer functions operate on the converted native format:

GUINAME_GetBitmapWidth	Return the width of the bitmap
GUINAME_ GetBitmapHeight	Return the height of the bitmap
GUINAME_ DrawBitmap	Draw the bitmap
GUINAME_ CreateStretchedBitmap	Rescale a bitmap
GUINAME_ DrawStretchedBitmap	Draw a rescaled bit map
GUINAME_ TileBitmap	Tile a rectanular area with the bitmap
GUINAME_ DestroyStretchedBitmap	Free the rescaled bit map created by GUINAME_CreateStretchedBitmap
GUINAME_DestroyWebBitmap	Free the bit map created by GUINAME_CreateWebBitmap

This is the WebGraphBitmapInfo structure:

WebGraphBitmapInfo structure

```
typedef struct s_WebGraphBitmapInfo WebGraphBitmapInfo;
struct s_WebGraphBitmapInfo
{
    DISPLAY_INT             iWidth;
    DISPLAY_INT             iHeight;
    WEBC_SIZE               iBytesPerRow;
    WebBitmapPixelFormat pixelFormat;
    WEBC_BOOL               bHasTransparentColor;
    WEBC_INT16              iTransparentIndex;
    WEBC_UINT8              uTransparentRed;
    WEBC_UINT8              uTransparentGreen;
    WEBC_UINT8              uTransparentBlue;
    WEBC_UINT32             iPaletteSize;
    const WEBC_UINT8*    pPalette;
    const WEBC_UINT8*    pData;
};
```

Where,

iWidth	Width of the bitmap in pixels
iHeight	Height of the bitmap in pixels
iBytesPerRow	Number of bytes of data per row (may or may not be iWidth * (bytes per pixel)
pixelFormat	Format of the data. May be one of the following:

WB_PIXEL_FORMAT_RGB_24	Data is formatted in 24 bit RGB color each pixel is represented by 3 bytes stored red-green-blue
WB_PIXEL_FORMAT_INDEX_8	Data is formatted in 8 bit per pixel palette indexed mode. Each byte represents one pixel, the color of

	<table><tr><td></td><td>the pixel is determined by indexing into the supplied palette</td></tr><tr><td>WB_PIXEL_FORMAT_MONOCHROME_8</td><td>Data is formatted in 8 bit per pixel monochrome</td></tr></table>
bHasTransparentColor	If non-zero the bitmap contains transparent pixels and may require special processing
iTransparentIndex	Contains the 8 bit pixel value that is transparent. This field is valid only if 'bHasTransparentColor' field is non-zero and 'pixelFormat' == WB_PIXEL_FORMAT_INDEX_8
uTransparentRed uTransparentGreen uTransparentBlue	These three fields contain the RGB triplet pixel value that is transparent. These fields are valid only if 'bHasTransparentColor' filed is non-zero and pixelFormat == WB_PIXEL_FORMAT_RGB_24
iPaletteSize	Contains the number of 3 byte entries in the palette pointed to by pPallette. This field is valid only if pixelFormat == WB_PIXEL_FORMAT_INDEX_8
pPalette	Contains a color palette for converting color indexes to RGB. This is an array of bytes. Each color is represented by a 3 byte sequence in red-green-blue order so the RGB triplet for an index value can be calculated as (r,g,b) == (*(pPallet+(index*3)),*(pPallet+1+(index*3)),,*(pPallet+2+(index*3)) This field is valid only if pixelFormat == WB_PIXEL_FORMAT_INDEX_8
pData	Contains the image data. If pixelFormat == WB_PIXEL_FORMAT_INDEX_8 or pixelFormat == WB_PIXEL_FORMAT_MONOCHROME_8 each byte represents one pixel. If pixelFormat == WB_PIXEL_FORMAT_RGB_24 each 3 byte byte sequence represents one pixel in red-green-blue order. *Note: The pixel data may or may not be contiguous from row to row. If iBytesPerRow is equal to (bytes_per_pixel *pixels_per_row) the data is contiguous. Otherwise the offset to the first pixel in a row is row_number * (bytes_per_pixel * pixels_per_row).*

HTMLBrowserConfig - A pointer of this type is passed to the routine webc_Init GUINAMEBrowserConfig() when WebC is first initialized. This structure must be zeroed by the BrowserConfig() function and then appropriately initialized.

The structure is defined as follows:

HTMLBrowserConfig structure

```
struct HTMLBrowserConfig
{
    HTMLGraphicsContext** graphics;
    /* return 1 if message processed, 0 if not, -1 if destroyed */
    ProcessInputQueueFn* processInputQueue;
    HTMLEventStatus (*browserEventHandler) (
                                HBROWSER_HANDLE hbrowser,
                                HTMLBrowserEvent* event);
    void* privateData;
};
```

Where:

HTMLGraphicsContext** graphics	This field must be initialized with a pointer to the GUI's private graphics context structure. The private graphics context structure must contain as it's first field a structure of type HTMLGraphicsContext that has been initialized with the addresses of GUI specific graphics functions.
ProcessInputQueueFn* processInputQueue	This field must be initialized with a pointer to a GUI specific function that retrieves messages from the GUIs message queue and passes them along to its message dispatcher function. WebC calls this function to allow message dispatching to continue when it performing processing loops that don't return quickly.
HTMLEventStatus (*browserEventHandler)	This field is set by the gui specific startup code after it calls webc_InitGuinameBrowserConfig() it is the event handler for the web browser application framework. It is used to update web brower status bars and the like. It is typically set to point to chromeBrowserEventHandler, which is the default framework.

HTMLGraphicsContext - This structure must be initialized by webc_InitGuinameBrowserConfig(). It contains pointers to functions that perform basic graphics functions. Webgui.cpp contains all of these functions and webc_InitGuinameBrowserConfig() must populate this structure with pointers to these functions. A detailed description of each of these functions is included later on in this document.

HTMLGraphicsContext Struct

```
typedef struct s_HTMLGraphicsContext HTMLGraphicsContext;
struct s_HTMLGraphicsContext
{
    /*---------------------------------------------------------------*/
    // Canvas services
    void (*getCanvasRect) (
    /*---------------------------------------------------------------*/

    // Pointer services
    void (*claimPointer) (
                                    HTMLGraphicsContext** ctx);
    void (*releasePointer) (
                                    HTMLGraphicsContext** ctx);
    /*---------------------------------------------------------------*/

    // Drawing control
    void (*invalidateRect) (
    void (*beginDrawing) (
    void (*getClipRect) (
    void (*setClipRect) (
    void (*endDrawing) (
    /*---------------------------------------------------------------*/

    // Color services
    HTMLColor (*colorToRGBA) (
    WebColor (*RGBAToColor) (
    WebColor (*getColorByIndex) (
    /*---------------------------------------------------------------*/

    // Drawing services
    void (*drawText) (
    void (*drawTextLen) (
    void (*drawBitmap) (
    void (*drawBitmapStretchedToRect) (
    void (*drawRectangle) (
    /*---------------------------------------------------------------*/

    // Font services
    void* (*createFont) (
    void* (*getDefaultFont) (
    void (*destroyFont) (
    int (*getNumFontFamilies) (
    const WEBC_CHAR** (*getFontFamilyNames) (
```

```
    const WEBC_CHAR** (*getFontFamilyGenericNames) (
    DISPLAY_INT (*getTextHeight) (
    DISPLAY_INT (*getTextWidth) (
    DISPLAY_INT (*getTextHeightLen) (
    DISPLAY_INT (*getTextWidthLen) (
    DISPLAY_INT (*getFontHeight) (
    DISPLAY_INT (*getFontBaseline) (
    /*-------------------------------------------------------------*/

    // Bitmap services
    void* (*createWebBitmap) (
    void (*destroyWebBitmap) (
    void* (*createStretchedBitmap) (
    void (*destroyStretchedBitmap) (
    DISPLAY_INT (*getBitmapWidth) (
    DISPLAY_INT (*getBitmapHeight) (
    /*-------------------------------------------------------------*/

    // Optional services (defining these methods will speed things up)
    void (*optionalDrawHorizontalLine) (
    void (*optionalDrawVerticalLine) (
    void (*optionalDrawStyledFrame) (
    void (*optionalDrawBitmapTiled) (

    // used to re-size the browser window if this feature is supported
    void (*optionalSetCanvasRect) (
    // draw a vertical scroll bar
    WEBC_BOOL (*optionalDrawVScroll) (
    // same as optionalDrawVScroll above, but for horizontal scroll bars
    WEBC_BOOL (*optionalDrawHScroll) (
#ifdef WEBC_BUFFER_SCROLLING
    HTMLGraphicsContext** (*createBuffer) (
    HTMLGraphicsContext** (*resizeBuffer) (
    void (*destroyBuffer) (
    void (*copyBufferRegion) (
#endif // WEBC_BUFFER_SCROLLING
};
```

A Look at Graphics Abstraction Functions

GUINAME_GetCanvasRect

Prototype
static void GUINAME_GetCanvasRect (HTMLGraphicsContext** ctx, HTMLRect* rect)

This function must return the rectangular coordinates of WebC's drawing area.
For example in Windows the routine GetClientRect (GET_WINDOW_HANDLE(ctx), &winRect) is called.

GUINAME_ClaimPointer

Prototype
static void GUINAME_ClaimPointer (HTMLGraphicsContext** ctx)

This function is called by WebC when a mouse button down event is detected while the pointer is over a slider or button. This routine must insure that all mouse events are delivered to WebC until GUINAME_ReleasePointer() is called.
For example in Windows the routine SetCapture(g->windowHandle) is called.

GUINAME_ReleasePointer

Prototype
static void GUINAME_ReleasePointer (HTMLGraphicsContext** ctx)

This function is called by WebC when a mouse button up event is detected and it had previpusly called GUINAME_ClaimPointer(). This routine must inform the GUI framework that WebC no longer requires all mouse events and if the pointer focus has changed it may deliver mouse events to the appropriate window.
For example in Windows the routine ReleaseCapture() is called.

GUINAME_InvalidateRect

Prototype
static void GUINAME_InvalidateRect (HTMLGraphicsContext** ctx, HTMLRect* rect)

This function is called by WebC when it requires a rectangle to be added to drawing operations it must ensure that the GUI framework will allow draw operations may be performed in the region and if it double buffers draw operations it must ensure that all regions marked invalid after a call to GUINAME_BeginDrawing() are refreshed to the display when GUINAME_EndDrawing() is called and the regions are marked valid by GUINAME_EndDrawing(). All invalidated regions must be OR'ed together to create the smallest rectangle that will contain all of the rectangles passed to GUINAME_InvalidateRect.

WebC maintains it own internal view of invalidated regions and only calls this routine to synchronize with the underlying GUI framework. Depending on your framework's requirements you may not have to populate this function.

For example in Windows this function is implemented wholly within the porting abstraction layer and is use only if the abstraction layer is double buffering graphics, drawing to an off-screen bitmap. The PEG graphics library requires that it be informed of regions it must redisplay so it internal invalidate function is called.

GUINAME_BeginDrawing

Prototype
static void GUINAME_BeginDrawing (HTMLGraphicsContext** ctx)

This function is called by WebC when it is beginning a sequence of invalidate and draw operation that will end with a call to GUINAME_EndDrawing. If an off-screen buffer is being used for drawing the underlying draw engine should be informed that a sequence of draws is beginning.

Depending on your framework's requirements you may not have to populate this function.

GUINAME_EndDrawing

Prototype
static void GUINAME_EndDrawing (HTMLGraphicsContext** ctx)

This function is called by WebC when it is beginning a sequence of invalidate and draw operation has completed and the underlying framework need to update the screen. If necessary the framework should build all invalidated regions to the screen buffer and clear the invalidated region list.

Depending on your framework's requirements you may not have to populate this function.

GUINAME_SetClipRect

Prototype
static void GUINAME_SetClipRect (HTMLGraphicsContext** ctx, HTMLRect* rect)

This function is called by WebC when it wants drawing operations to be clipped to not draw beyond the provided rectangular coordinates.

For example in Windows the following sequence is invoked.

```
HTML_TO_RECT(gdiRect, *rect);
GET_GRAPHICS_OBJ(ctx)->SetClip(gdiRect, CombineModeReplace);
```

GUINAME_GetClipRect

Prototype
static void GUINAME_GetClipRect (HTMLGraphicsContext** ctx,HTMLRect* rect)

This function is called by WebC when it needs to know what the current clipping region is set to.

For example in Windows the following sequence is invoked.

```
{
    Rect clipRect;
    GET_GRAPHICS_OBJ(ctx)->GetClipBounds(&clipRect);
    RECT_TO_HTML(*rect, clipRect);
}
```

GUINAME_GetColorByIndex

Prototype

This function is called by WebC when it needs to convert one of its named colors to a color that can be passed to native draw functions. The passed argument is a named color and the return value is a 32 bit color value that can be passed directly to the draw functions. The returned color may be a color constant used by the GUI package or an RGB triplet if the GUI package used RGB triplets for drawing colors.

The following arguments are possible:

WGC_BLACK
WGC_BLUE
WGC_DARKGRAY

WGC_LIGHTBLUE WGC_TRANSPARENT WGC_RED WGC_MAGENTA WGC_LIGHTRED WGC_LIGHTMAGENTA WGC_GREEN WGC_CYAN WGC_LIGHTGREEN WGC_LIGHTCYAN WGC_BROWN WGC_LIGHTGRAY WGC_YELLOW WGC_WHITE

The windows CE graphics abstraction file contains an example of an implementation that returns RGB triplets.

GUINAME_RGBToColor

Prototype
static WebColor GUINAME_RGBToColor (HTMLGraphicsContext** ctx, HTMLColor color, HTMLColor* error)

This function is called by WebC when it needs to convert and RGB triplet to a natively displayable color that may be passed to native draw functions.
The error argument is not used and should be processed as follows.

```
if (*error)
{
    *error = 0;
}
```

The following macros are available for extracting R,G and B components from the HTMLColor argument.

HTML_ColorGetRed(color),
HTML_ColorGetGreen(color),
HTML_ColorGetBlue(color)

GUINAME_ColorToRGB

Prototype
static HTMLColor GUINAME_ColorToRGB (HTMLGraphicsContext** ctx,WebColor colorval)

This function is called by WebC when it needs to convert a natively displayable color to an RGB triplet in WebC's internal form HTMLColor.

The following macros is available for stuffing R,G, B and A (alpha) components into the HTMLColor 32 bit internal color representation.

return HTML_RGBAToColor (R, G, B, A);

A should always be zero when used in this context.

GUINAME_DrawText

Prototype
static void GUINAME_DrawText (HTMLGraphicsContext** ctx, DISPLAY_INT x, DISPLAY_INT y, const WebChar* text, WebColor textColor, WebColor backgroundColor, WEBC_BOOL fill, void* font)

This function is called by WebC when it needs to draw a NULL terminated text string at X, Y. The text itself should be drawn in the native color passed as TextColor. If Fill is non-zero the text background should be drawn in the native color passed as backgroundColor. The font to use is passed in the void pointer "font". This is the native font family pointer or handle that was returned from the functions createFont or getDefaultFont.

GUINAME_DrawTextLen

Prototype
static void GUINAME_DrawTextLen (HTMLGraphicsContext** ctx, DISPLAY_INT x, DISPLAY_INT y, const WebChar* text,

```
        WebColor textColor,
        WebColor backgroundColor,
        WEBC_BOOL fill,
        void* font,
        long textLen)
```

This function is identical to GUINAME_DrawText except the text to display is not terminated and the function must display "textlen" characters only.

GUINAME_DrawBitmap

Prototype
static void GUINAME_DrawBitmap (HTMLGraphicsContext** ctx, DISPLAY_INT x, DISPLAY_INT y, void* bmp)

This function is called by WebC when it needs to render a bitmap. The bitmap is passed through the void pointer at *bmp. This bitmap is the void pointer that was returned from GUINAME_CreateWebBitmap().

GUINAME_DrawStretchedBitmap

Prototype
static void GUINAME_DrawStretchedBitmap (HTMLGraphicsContext** ctx, HTMLRect* rect, void* b)

This function is called by WebC when it needs to render a scaled bitmap within a rectangular area. The bitmap is passed through the void pointer at *b. This bitmap is the void pointer that was returned from GUINAME_CreateStretchedBitmap().

GUINAME_DrawRectangle

Prototype
static void GUINAME_DrawRectangle (HTMLGraphicsContext** ctx, HTMLRect* rect, WebColor outlineColor, WebColor fillColor, WEBC_BOOL fill)

This function is called by WebC when it needs to draw a rectangle. The rectangle must be drawn in the native color passed in outlineColor and if "fill" is non-zero the rectangular area should be filled with the native color passed in fillcolor.

GUINAME_GetDefaultFont

Prototype
void* GUINAME_GetDefaultFont (HTMLGraphicsContext** ctx)

This function is called by WebC to retrieve a handle or pointer to a default font that it will use to draw text in list boxes and drop boxes. The returned value will later be passed to other graphics abstraction layer functions that take a font handle/pointer as one of their arguments these functions include

GUINAME_DrawText, GUINAME_DrawTextLen, GUINAME_GetTextHeight, GUINAME_GetTextHeightLen, GUINAME_GetTextWidth, GUINAME_GetTextWidthLen, GUINAME_GetFontHeight, GDIPlus_GetFontBaseline

GUINAME_CreateFont

Prototype
void* GUINAME_CreateFont (HTMLGraphicsContext** ctx, const WEBC_CHAR* familyName, const WEBC_CHAR* genericName, WEBC_UINT16 pointSize, WEBC_UINT16 attrib, int familyIndex)

Where:

familyName	The name of the font to be created Such as: WEBC_STR_ARIAL, WEBC_STR_COURIER, WEBC_STR_TIMES or WEBC_STR_VERDANA
genericName	Corresponding generic font (category) or font typeface associated with the 'familyName' such as: WEBC_STR_SANS_SERIF, WEBC_STR_MONOSPACE, WEBC_STR_SERIF
pointSize	The point size of the font. This value will be between WEBC_SMALLEST_FONT_SIZE_PT(6) and WEBC_LARGEST_FONT_SIZE_PT(36). If the exact point size is not available the font of the nearest available size should be used.
attrib (0-3)	Regular, Bold, Italic, BoldItalic
familyIndex	Index into the user defined font name tables

This function is called by WebC to retrieve a handle or pointer to a font that it will use to render html content. The returned value will later be passed to other graphics abstraction layer functions that take a font handle/pointer as one of their arguments these functions include

GUINAME_DrawText, GUINAME_DrawTextLen, GUINAME_GetTextHeight, GUINAME_GetTextHeightLen,

GUINAME_GetTextWidth,
GUINAME_GetTextWidthLen,
GUINAME_GetFontHeight,
GDIPlus_GetFontBaseline

The routine should return a handle to a font that best approximates the requested font family or generic font type, the pointsize and the style.

The user must create two tables, where one contains the names of fonts supported by the underlying graphics library and the second contains their corresponding generic names (categories). The format of these tables is similar to that of those shown below.

Shown below is an example of such tables in PEG porting file (*wfpeg.cpp)*

gpFontFamilyName	gpFontFamilyGenericName

static WEBC_READONLY WEBC_CHAR* gpFontFamilyName [WEBC_CFG_MAX_FONT_FAMILIES] = {
 	
WEBC_STR_ARIAL,
WEBC_STR_COURIER,
WEBC_STR_TIMES,
WEBC_STR_VERDANA
};
	static WEBC_READONLY WEBC_CHAR* gpFontFamilyGenericName [WEBC_CFG_MAX_FONT_FAMILIES] = {
	
	WEBC_STR_SANS_SERIF,
	WEBC_STR_MONOSPACE,
	WEBC_STR_SERIF,
	WEBC_STR_SANS_SERIF
	};

Note that for each font name in the gpFontFamilyName table there is a matching generic name entry in the gpFontFamilyGenericName table.

WebC Font Selection:
WebC uses the following logic to look up the name tables for a particular font

WebC uses the underlying graphics layer to locate a particular font in the following way.

First it calls the following:
GUINAME_NumFontFamilies ( ). – To get the number of fonts supported
Then it calls:
GUINAME_GetFontFamilyName() – To get the FontFamilyName table (which must have GUINAME_NumFontFamilies ( ).entries.
Then it calls:
GUINAME_GetFontFamilyGenericNames() – To get the FontFamilyGenericNames table (which must have GUINAME_NumFontFamilies ( ).entries.

Then it performs the following search:

For each index in (0, GUINAME_NumFontFamilies)
- Check to see if target font name equals FontFamilyName[index_number]
  If not check if target font name equals FontFamilyGenericName[index_number]

If no match is found it may do the search using a generic font name if the HTML content provided one.

If no match is found WebC defaults to using the font at index zero

For example an HTML document contains a font declared in a style sheet as:

```
{ font-family: "Times New Roman", serif }
```

WebC first tries to find the “Times New Roman” font, if it cannot the time font, it tries to find the Serif font, since this was provided as an alternative.

Finally if neither “Times New Roman” nor Serif can be found, WebC defaults to the font at index zero, which in the above table is Arial.

WebC then calls GUINAME_CreateFont ( ) with the following arguments:

familyindex - is equal to the index described above.
‘familyName’ - is equal to FontFamilyName[index_number]
genericName’ - is equal to FontFamilyGenericName[index_number].

Note the index and name arguments are actually redundant but are provided for convenience.

It also hands in pointsize and attributes. Some font libraries may be able to create a font from the family information with this pointsize and attributes. Others may require a unique font for each familyindex, pointsize, attributes triplet and some combination of these three integers may be used to index into a table of these fonts.

As seen from the above segment familyIndex supplied to GUINAME_CreateFont ( ) is always between 0 and return value of GUINAME_NumFontFamilies ( ). You may use either the familyIndex or familyName to identify the font type to create. Or you may use both arguments.. for example you may use the familyName argument to initially create a font of that family and store it internally in a table indexed by familyIndex. Subsequent calls can use the font in your internal table indexed by familyIndex.

Note: There are several examples of these functions in the provided GUI abstraction layers. wfpeg.cpp implements this function using tables of fixed sized, fixed style fonts.

wfgdip.cpp implements this function using truetype base font types and underlying GUI functions that return variants of pointSize and style attributes.

wfgrafx implements this function using tables of fixed sized, fixed style fonts and demonstrates font handling with a reduced set of available fonts.

GUINAME_NumFontFamilies

Prototype
int GUINAME_NumFontFamilies (HTMLGraphicsContext** ctx)

This function is called by WebC to retrieve the number of font families that are available.

It must return WEBC_CFG_MAX_FONT_FAMILIES, and the GUI abstraction layer must support WEBC_CFG_MAX_FONT_FAMILIES separate font families.
WEBC_CFG_MAX_FONT_FAMILIES should be changed to reflect this value.

GUINAME_GetFontFamilyNames

Prototype
const WEBC_CHAR** GUINAME_GetFontFamilyNames (HTMLGraphicsContext** ctx)

This function is called by WebC to retrieve a table of the font families that are available by name. The table must contain WEBC_CFG_MAX_FONT_FAMILIES entries.

For example: The default windows implementation supports four fonts, Arial, Courier, Times and Verdana. And its version of GUINAME_GetFontFamilyNames() returns the address of the following table.

```
static WEBC_READONLY WEBC_CHAR*
gpFontFamilyName[WEBC_CFG_MAX_FONT_FAMILIES] = {

        WEBC_STR_ARIAL,
        WEBC_STR_COURIER,
        WEBC_STR_TIMES,
        WEBC_STR_VERDANA
    };
```

To add additional fonts or to change the mix of available font increase or decrease WEBC_CFG_MAX_FONT_FAMILIES, update the font family name and generic font name tables, and edit your GUINAME_CreateFont() function to support the new fonts.

GUINAME_GetFontFamilyGenericNames

Prototype
const WEBC_CHAR** GUINAME_GetFontFamilyGenericNames (HTMLGraphicsContext** ctx)

This function is called by WebC to retrieve a table of the generic font families that are available by name. The table must contain WEBC_CFG_MAX_FONT_FAMILIES entries.

For example: The default windows implementation supports four generic fonts, SNS_SERIF, MONOSPACE and SERIF. And its version of GUINAME_GetFontFamilyGenericNames() returns the address of the following table.

```
static WEBC_READONLY WEBC_CHAR*
gpFontFamilyGenericName[WEBC_CFG_MAX_FONT_FAMILIES] = {
        WEBC_STR_SANS_SERIF,
        WEBC_STR_MONOSPACE,
        WEBC_STR_SERIF,
        WEBC_STR_SANS_SERIF
    };
```

GUINAME_GetTextHeight

Prototype
static DISPLAY_INT GUINAME_GetTextHeight (HTMLGraphicsContext** ctx,const WebChar* text, void* font)

This function is called by WebC to retrieve the height of the rectangle occupied by the null terminated string when it is rendered in the provided font. If the font height is fixed this will be the character height of the font, otherwise it will be the distance from the lowest decender to the highest ascender in the string.

GUINAME_GetTextHeightLen

Prototype
static DISPLAY_INT GUINAME_GetTextHeightLen (HTMLGraphicsContext** ctx, const WebChar* text,void* font, long textLen)

This function is called by WebC to retrieve the height of the rectangle occupied by the provided non terminated sequence of characters of the specified length. It is identical to GUINAME_GetTextHeight() except the string is length specified rather than NULL terminated.

GUINAME_GetTextWidth

Prototype
static DISPLAY_INT GUINAME_GetTextWidth (HTMLGraphicsContext** ctx, const WebChar* text, void* font)

This function is called by WebC to retrieve the width of the rectangle occupied by the null terminated string when it is rendered in the provided font.

GUINAME_GetTextWidthLen

Prototype
static DISPLAY_INT GUINAME_GetTextWidthLen (HTMLGraphicsContext** ctx, const WebChar* text, void* font, long size)

This function is called by WebC to retrieve the width of the rectangle occupied by the length specified string when it is rendered in the provided font. It is identical to GUINAME_GetTextWidth() except the string is length specified rather than NULL terminated.

GUINAME_GetFontHeight

Prototype
static DISPLAY_INT GUINAME_GetFontHeight (HTMLGraphicsContext** ctx,void* font)

This function is called by WebC to retrieve the height of a rectangle occupied by the provided font. If the font is fixed height the returned height should be the fixed size. If characters have different heights it should return the distance from the lowest decender in the font to the highest ascender.

GUINAME_GetFontBaseline

Prototype
static DISPLAY_INT GUINAME_GetFontBaseline (HTMLGraphicsContext** ctx,void* font)

This function is called by WebC to retrieve the distance in pixels from the top of the cell ascent part of a character to the base of the ascent part. Take for example the letters M and Q. There is no descender for M so the font baseline is the height of the letter. For Q the font baseline is the base base of the large circle. If the pointer to this function is not populated WebC defaults to using four fifths of the font height + 1 as the font baseline.

GUINAME_CreateWebBitmap

Prototype
static void* GUINAME_CreateWebBitmap (HTMLGraphicsContext** ctx,const WebGraphBitmapInfo* info)

This function is called by WebC when it needs a bitmap (pixmap) that can be displayed, stretched and measured by native graphics implementation. The routine must allocate data to hold the native formatted bitmap and populate it using the content provided in the WebGraphBitmapInfo structure. The format of WebGraphBitmapInfo is provided above in the WebC type descriptions and should provide a framework for how to create a native bitmap from the input.

This routine must return a void pointer to the native bitmap or 0 if it fails.

GUINAME_DestroyWebBitmap

Prototype
static void GUINAME_DestroyWebBitmap (HTMLGraphicsContext** ctx,void* b)

This function is called by WebC to release (free) a native bitmap structure that was created by GUINAME_CreateWebBitmap. The input argument is the same address that was returned from GUINAME_CreateWebBitmap.

GUINAME_CreateStretchedBitmap

Prototype
static void* GUINAME_CreateStretchedBitmap (HTMLGraphicsContext** ctx, void* b, DISPLAY_INT w, DISPLAY_INT h);

This function is called by WebC when it needs to scale a bitmap (pixmap) that was returned from GUINAME_CreateWebBitmap. The input argument is the same address that was returned from

GUINAME_CreateWebBitmap. The return value must be a void pointer to a new native bitmap that is scaled to the provided width and height. If the routine fails it must return 0.

GUINAME_DestroyStretchedBitmap

Prototype
static void GUINAME_DestroyStretchedBitmap(HTMLGraphicsContext** ctx, void* bmp)

This function is called by WebC to release (free) a native bitmap structure that was created by GUINAME_CreateStretchedBitmap. The input argument is the same address that was returned from GUINAME_CreateStretchedBitmap.

GUINAME_GetBitmapWidth

Prototype
static DISPLAY_INT GUINAME_GetBitmapWidth (HTMLGraphicsContext** ctx, void* bmp)

This function is called by WebC to retrieve the width of a native bitmap structure that was created by GUINAME_CreateWebBitmap or GUINAME_CreateStretchedBitmap. The input argument is the same address that was returned from GUINAME_CreateWebBitmap or GUINAME_CreateStretchedBitmap.

GUINAME_GetBitmapHeight

Prototype
static DISPLAY_INT GUINAME_GetBitmapHeight (HTMLGraphicsContext** ctx, void* bmp)

This function is called by WebC to retrieve the Height of a native bitmap structure that was created by GUINAME_CreateWebBitmap or GUINAME_CreateStretchedBitmap. The input argument is the same address that was returned from GUINAME_CreateWebBitmap or GUINAME_CreateStretchedBitmap.

The following optional function fields in the graphics context are not used and should be set to zero..

```
void (*optionalDrawHorizontalLine) (
void (*optionalDrawVerticalLine) (
void (*optionalSetCanvasRect) (
```

GUINAME_optionalDrawStyledFrame

Prototype
void GUINAME_optionalDrawStyledFrame (HTMLGraphicsContext** ctx, HTMLRect* rect, WebColor color, DISPLAY_INT thickness, CSSBorderStyle style)

This function is used to draw a decorated rectangular frame. If the field in the graphics context named void (*optionalDrawStyledFrame) is populated, this function is called to draw decorated frames. Otherwise WebC uses other primitives to draw the frame itself.
Where border style may be one of the following values:

CSS_BORDER_STYLE_DASHED, CSS_BORDER_STYLE_DOTTED, CSS_BORDER_STYLE_DOUBLE, CSS_BORDER_STYLE_GROOVE, CSS_BORDER_STYLE_INSET, CSS_BORDER_STYLE_NONE, CSS_BORDER_STYLE_OUTSET, CSS_BORDER_STYLE_RIDGE, CSS_BORDER_STYLE_SOLID

GUINAME_optionalDrawBitmapTiled

Prototype
void GUINAME_optionalDrawBitmapTiled(HTMLGraphicsContext** ctx, HTMLRect* rect, DISPLAY_INT patternXOffset, DISPLAY_INT patternYOffset, void* bmp)

This function is used to draw a rectangle with a tiled bitmap background. If the field in the graphics context named void (*optionalDrawBitmapTiled) is populated, this function is called to draw tiled rectangles. Otherwise WebC uses other primitives to draw tiled rectangle itself.

The function must fill the rectangle starting at patternXOffset, patternYOffset, with the bitmap.

GUINAME_optionalDrawVScroll

Prototype
WEBC_BOOL GUINAME_optionalDrawVScroll (HTMLGraphicsContext** ctx, HTMLRect* rect, WEBC_UINT32 buttonLength, WEBC_UINT32 sliderBegin, WEBC_UINT32 sliderLength, WEBC_UINT32 scrollFlags)

Where scrollFlags is a bit-wise combination of the following:

HTML_SCROLL_FLAG_DECREASE_ACTIVE	The decrease scroll position button is active
HTML_SCROLL_FLAG_INCREASE_ACTIVE	The increase scroll position button is active
HTML_SCROLL_FLAG_SLIDER_ACTIVE	The slider is active
HTML_SCROLL_FLAG_BEFORE_SLIDER_ACTIVE	The region representing the range below the visible region is active
HTML_SCROLL_FLAG_AFTER_SLIDER_ACTIVE	The region representing the range above the visible region is active
HTML_SCROLL_FLAG_HAS_FOCUS	The scroll bar has input focus
HTML_SCROLL_FLAG_LEFT_FLUSH	The scroll bar is flush against the left side of the parent rect
HTML_SCROLL_FLAG_TOP_FLUSH	The scroll bar is flush against the top side of the parent rect
HTML_SCROLL_FLAG_RIGHT_FLUSH	The scroll bar is flush against the right side of the parent rect
HTML_SCROLL_FLAG_BOTTOM_FLUSH	The scroll bar is flush against the bottom side of the parent rect
HTML_SCROLL_FLAG_DISABLED	The scroll bar is disabled
HTML_SCROLL_FLAG_READONLY	The scroll bar is in a read-only mode

This function is used to draw a draw a vertical scroll bar. If the field in the graphics context named WEBC_BOOL (*optionalDrawVScroll) is populated, this function is called to draw vertical scroll bars. Otherwise WebC uses other primitives to draw vertical scroll bars itself.
For most ports this function is not used. See the PEG graphics abstraction layer for an example of how it can be implemented.

GUINAME_optionalDrawHScroll

Prototype
WEBC_BOOL GUINAME_optionalDrawHScroll (HTMLGraphicsContext** ctx, HTMLRect* rect, WEBC_UINT32 buttonLength, WEBC_UINT32 sliderBegin, WEBC_UINT32 sliderLength, WEBC_UINT32 scrollFlags);

This function is used to draw a horizontal scroll bar. If the field in the graphics context named WEBC_BOOL (*optionalDrawHScroll) is populated, this function is called to draw horizontal scroll bars. Otherwise WebC uses other primitives to draw horizontal scroll bars itself.

This function is the same as optionalDrawVScroll above, but for horizontal scroll bars

Optional buffer scrolling routines

These routines are provided to allow WebC to render into a memory based screen buffer that is wider than the actual display screen. These functions are only required if the constant WEBC_BUFFER_SCROLLING is defined.

This mode of operation is experimental but has been used succesfully with WindowsCE and should work with any graphics library that supports drawing to memory devices and bit blitting sections of that device to the screen.

The fields in the graphics context that must be initialized to support BUFFER_SCROLLING are:

```
HTMLGraphicsContext** (*createBuffer) (
    HTMLGraphicsContext** (*resizeBuffer) (
    void (* destroyBuffer) (
    void (*copyBufferRegion) (
```

Please study the windowsCE porting file webcdi_wce.cpp for a sample implementation of these functions.

```
HTMLGraphicsContext** GUINAME_createBuffer (
        HTMLGraphicsContext** ctx,
        DISPLAY_INT w,
        DISPLAY_INT h);

HTMLGraphicsContext** GUINAME_resizeBuffer (
        HTMLGraphicsContext** ctx,
```

```
        HTMLGraphicsContext** buffer,
        DISPLAY_INT w,
        DISPLAY_INT h);

void GUINAME_destroyBuffer (
        HTMLGraphicsContext** ctx,
        HTMLGraphicsContext** buffer);


void GUINAME_copyBufferRegion) (
        HTMLGraphicsContext** dstCtx,
        HTMLRect* dstRect,
        HTMLGraphicsContext** srcBuffer,
        DISPLAY_INT srcX,
        DISPLAY_INT srcY);
```