Font Utilities—Some Handy-Dandy Font-Handling Functions

Charlie Kindel, Microsoft Premier Support Services
Kyle Marsh, Microsoft Developer Network Technology Group
Created: January 10, 1993

Click to open or copy the files in the FONTUTIL sample application for this technical article.
tech\gdi\fontutil

Abstract

Handling fonts in Microsoft® Windows™-based applications can involve a lot of grunt work. For example, essential tasks such as creating fonts, retrieving point sizes, and calculating text metrics can take up a lot of the developer's time and effort. This article describes some useful font-handling functions that take care of most of this tedious work, thus making it easier to create and manipulate fonts in Windows.
The FONTUTIL sample application demonstrates most of the font-handling functions discussed in this technical article.

Introduction

The font utilities described in this article are primarily for applications that need the relatively simplistic font model presented in the **ChooseFont** common dialog box. **ChooseFont** deals with fonts using four primary variables: *typeface name*, *typeface style*, *point size*, and *effects*. **ChooseFont** also allows for color selection, but the font-handling functions do not currently support this feature.

The font-handling functions allow an application to create and interrogate fonts in a manner that is consistent with the **ChooseFont** model. For example, you can create a font by using the **ReallyCreateFont** function and specifying the typeface name, typeface style, and point size; you can retrieve the typeface style of a font by using the **GetTextStyle** function; and so on.

The font utility functions support Microsoft® Windows™ versions 3.0 and 3.1, and Windows NT™ version 3.1. You can include the functions either in an application or in a dynamic-link library (DLL).

Using the Font Utilities

There are two important points to remember when using the font utilities:

- You must export the **fnEnumReallyEx** callback function in the application's or DLL's module definition file, or use an appropriate compiler option.
- The font utilities are English-specific; to support other languages, you must use the **LoadString** function with string resources.

Exporting fnEnumReallyEx

The application or DLL containing the font utilities must export the font enumeration callback function **fnEnumReallyEx** in the module definition file. For example, the FONTUTIL sample application's FONT.DEF file exports **fnEnumReallyEx** with the following lines:

```
EXPORTS
    fnEnumReallyEx      @103
```

Alternatively, Microsoft C/C++ version 7.0 and Visual C++™ version 1.0 users can compile the utilities with the **–GA** option (for applications) or the **–GD** option (for DLLs). These options eliminate the need to export **fnEnumReallyEx** in the module definition file.

Supporting Different Languages

The font utilities use hard-coded English strings for font attributes. To support other languages, you must use string resources and load them into the utilities with the **LoadString** function. For example, to support German, you must translate the term "bold" into something like "Fettdruck." The variables holding the strings are currently declared as follows:

```
char szBold[]       = "Bold" ;
char szItalic[]     = "Italic" ;
char szBoldItalic[] = "Bold Italic" ;
char szRegular[]    = "Regular" ;
```

To support other languages, the variables would be declared as:

```
extern LPSTR rglpsz[] ;

#define szBold       rglpsz[IDS_BOLD]
#define szItalic     rglpsz[IDS_ITALIC]
#define szBoldItalic rglpsz[IDS_BOLDITALIC]
#define szRegular    rglpsz[IDS_REGULAR]
```

Implementing this feature is left as an exercise for the reader.

Function Reference

The font utilities consist of seven functions:

- Two of these functions create fonts: **ReallyCreateFont** creates a font for a specific device context (DC), and **CreateMatchingFont** creates a font that can be used by two device contexts.
- The remaining five functions retrieve information about a selected font: **GetAverageWidth** computes the character width, **GetTextStyle** retrieves the style name, **GetTextFullName** retrieves the full typeface name, **GetTextPointSize** retrieves the point size, and **GetTextExtentABCPoint** computes the dimensions of the font based on ABC widths.

The sections below describe these functions in alphabetical order.

CreateMatchingFont

This function gets the closest matching font between two device contexts. This information is useful when producing WYSIWYG output.

Syntax

```
HFONT WINAPI CreateMatchingFont

   (HDC hdcScr,
    HDC hdcPrn,
    LPSTR lpszFace,
    LPSTR lpszStyle,
    LPSTR nPointSize,
    UINT uiFlags);
```

Parameters

hdcScr
Specifies the device context to which the specified font will be mapped. If this parameter is NULL, the screen DC is used.

hdcPrn
Identifies a printer DC. The typeface, style, and point-size parameters specify a font that is already selected into the printer DC.

lpszFace
Points to a null-terminated string that identifies the typeface name of the font to be created (for example, "MS Sans Serif").

lpszStyle
Points to a null-terminated string that identifies the font style to be used (for

example, "Bold Italic"). If *lpszStyle* is NULL, the style indicated by *uiFlags* is used.

nPointSize

Specifies the point size of the font to be created (1 point = 1/72 inch).

uiFlags

Specifies flags that modify the behavior of the function. You may use any combination of the following:

Value	**Meaning**
RCF_NORMAL	Creates a font with the regular font style.
RCF_ITALIC	If *lpszStyle* is NULL, creates an italic font.
RCF_UNDERLINE	Creates an underlined font.
RCF_STRIKEOUT	Creates a font with strikeout.
RCF_BOLD	If *lpszStyle* is NULL, creates a bold font.
RCF_NODEFAULT	Specifies that a system font should not be returned. Normally, **CreateMatchingFont** always attempts to create a font. If the specified parameters do not match an installed font, it creates a system font unless this flag is

set.

Return value

The **CreateMatchingFont** function returns the handle to the created font. If a font could not be created, the function returns NULL.

GetAverageWidth

This function retrieves the text metrics of the font currently selected into the device context and returns the average character width. It computes the average character width by using **GetTextExtent** on the "abc...xyzABC...XYZ" string. This computation method works much better than the **tmAveCharWidth** value in the **TEXTMETRIC** structure returned by the **GetTextMetrics** function for alphanumeric, Latin-based proportional fonts, especially when used for dialog-box unit calculations.

Syntax

```
int WINAPI GetAverageWidth

   (HDC hdc,
    LPTEXTMETRIC lptm);
```

Parameters

hdc
: Identifies the device context for which the text metrics information should be returned.

lptm
: Points to the **TEXTMETRIC** structure that receives the metrics.

Return value

The **GetAverageWidth** function returns the average character width of the currently selected font, computed as described below (the *rgchAlphabet* parameter contains the string "abc...xyzABC...XYZ").

- If the code detects it is running on Windows 3.0, it uses:

```
nAveWidth = LOWORD( GetTextExtent( hDC,
                    (LPSTR)rgchAlphabet, 52 ) ) / 52 ;
```

- If the code detects it is running on Windows 3.1, it uses:

```
nAveWidth = LOWORD( GetTextExtent( hDC,
                    (LPSTR)rgchAlphabet, 52 ) ) / 26 ;
// Round up
nAveWidth = (nAveWidth + 1) / 2 ;
```

- If the code detects neither Windows 3.0 or 3.1, it assumes a later version of Windows (for example, Windows NT) and uses:

```
GetTextExtentPoint( hDC, (LPSTR)rgchAlphabet, 52, &size ) ;
nAveWidth = size.cx / 26 ;
nAveWidth = (nAveWidth + 1) / 2 ;
```

Comments

The calculations above are identical to those that USER uses to calculate character widths for dialog-box units.

GetTextExtentABCPoint

The **GetTextExtentABCPoint** function computes the dimensions (advance width) of the specified text string, taking ABC widths into consideration. **GetTextExtentABCPoint** uses the currently selected font to compute the width and height of the string in logical units, without considering any clipping.

The "A" width of a character is the distance added to the current position before drawing the character glyph. A negative "A" width indicates an underhang. The "B" width of a character is the width of the drawn portion of the character glyph. The "C" width of a character is the distance added to the current position to provide white space to the right of the character glyph. A negative "C" width indicates an overhang. The *advance width* of a character is the sum of the A, B, and C widths.

Syntax

```
UINT WINAPI GetTextExtentABCPoint
```

```
(HDC hdc,
 LPSTR lpszString,
 int cbString,
 LPSIZE lpSize);
```

Parameters

hdc
: Identifies the device context.

lpszString
: Points to a text string.

cbString
: Specifies the number of bytes in the text string.

lpSize
: Points to a **SIZE** structure that will receive the dimensions of the string.

Returns

The **GetTextExtentABCPoint** function returns the "A" width of the first character in the string. If the "A" width is negative, **GetTextExtentABCPoint** returns its absolute value. In the case of an error, the function returns zero.

Comments

Unlike **GetTextExtentPoint**, this function uses the **GetCharABCWidths** function to calculate the extent of the specified string. This function is very useful if you want to calculate the full extent of a string.

For example, if you use the standard Windows **GetTextExtentPoint** function to calculate the string extent, the string will appear clipped on both the left and the right (Figure 1).

Figure 1. Using GetTextExtentPoint

If you calculate the string extent with the **GetTextExtentABCPoint** function, but do not use the return value to offset the first character, the string will appear clipped on the left (Figure 2).

Figure 2. Using GetTextExtentABCPoint (return value ignored)

Figure 3 illustrates the string when you use the **GetTextExtentABCPoint** function, and offset the first character by the return value. The string appears in its entirety, with no clipping.

Figure 3. Using GetTextExtentABCPoint (return value considered)

Under Windows 3.0 or when a non-TrueType® font is selected, the **GetTextExtentABCPoint** function is simply a wrapper around **GetTextExtent** or **GetTextExtentPoint**.

GetTextExtentABCPoint also works around a bug in Windows 3.1: The **GetCharABCWidths** function calculates the ABC spacing incorrectly for fonts that simulate boldface. For example, Windows 3.1 does not include the TrueType Wingdings™ Bold font. If the user selects a font that was created with the Wingdings typeface name and a weight greater than FW_NORMAL into a DC, the graphical device interface (GDI) simulates bold by overstriking the characters. When an application uses **GetCharABCWidths** to determine the advance width of this font, the ABC widths returned are off by one for each character. To work around this bug, the **GetTextExtentABCPoint** function adds one logical unit to the "B" width of each character.

GetTextFullName

This function retrieves the full name of the selected font, and copies it into the buffer as a null-terminated string.

The full typeface name is found only in TrueType fonts. It usually contains a version number and other identifying information. For example, the full name of the Windows Times New Roman Bold Italic typeface is *Monotype:Times New Roman Bold Italic:Version 1 (Microsoft)*.

Syntax

```
 UINT WINAPI GetTextFullName

   (HDC hdc,
    UINT cbBuffer,
    LPSTR lpszStyle);
```

Parameters

hdc
Identifies the device context.

cbBuffer
Specifies the buffer size in bytes. If the full name of the typeface is longer than the number of bytes specified by this parameter, the name is truncated.

lpszFace
Points to the buffer that contains the full typeface name.

Return value

If the **GetTextFullName** function is successful, its return value specifies the number of bytes copied to the buffer, not including the terminating null character. Otherwise, the return value is zero.

Comments

This function uses the **GetOutlineTextMetrics** function in Windows 3.1 and later, and returns a null string in Windows 3.0.

GetTextPointSize

This function calculates the point size of the selected font using the following code:

```
nPtSize = MulDiv( tm.tmHeight - tm.tmInternalLeading,
              72,
              GetDeviceCaps( hDC, LOGPIXELSY ) );
```

Syntax

```
UINT WINAPI GetTextPointSize

  (HDC hdc);
```

Parameter

hdc
Handle to the device context.

Return value

The **GetTextPointSize** function returns the point size of the currently selected font.

GetTextStyle

This function retrieves the style name of the selected font, and copies it into the buffer as a null-terminated string.

Syntax

```
UINT WINAPI GetTextStyle

  (HDC hdc,
        UINT cbBuffer,
        LPSTR lpszStyle);
```

Parameters

hdc
: Identifies the device context.

cbBuffer
: Specifies the buffer size in bytes. If the style name is longer than the number of bytes specified by this parameter, the name is truncated.

lpszStyle
: Points to the buffer that contains the typeface style name.

Returns

If the **GetTextStyle** function is successful, its return value specifies the number of bytes copied to the buffer, not including the terminating null character. Otherwise, the return value is zero.

Comments

This function uses the **GetOutlineTextMetrics** function in Windows 3.1 and later, and the **GetTextMetrics** in Windows 3.0.

ReallyCreateFont

This function creates a font based on typeface name, typeface style, and point size.

Syntax

```
 HFONT WINAPI ReallyCreateFont

    (HDC hdc,
     LPSTR lpszFace,
     LPSTR lpszStyle,
     int nPointSize,
     UINT uiFlags);
```

Parameters

hdc
: Identifies the device context to use (NULL for the screen DC).

lpszFace
: Points to a null-terminated string that identifies the typeface name of the font to be created (for example, "MS Sans Serif").

lpszStyle

Points to a null-terminated string that identifies the typeface style of the font to be created (for example, "Bold Italic"). If *lpszStyle* is NULL, the style indicated by *uiFlags* is used.

nPointSize

Specifies the point size of the font to be created (1 point = 1/72 inch).

uiFlags

Specifies flags that modify the behavior of the function. You may use any combination of the following:

Value	**Meaning**
RCF_NORMAL	Creates a font with the regular font style.
RCF_ITALIC	If *lpszStyle* is NULL, creates an italic font.
RCF_UNDERLINE	Creates an underlined font.
RCF_STRIKEOUT	Creates a font with strikeout.
RCF_BOLD	If *lpszStyle* is NULL, creates a bold font.
RCF_NODEFAULT	Specifies that a system font should not be returned. Normally, **ReallyCreateFont** always attempts to create a

font. If the specified parameters do not match an installed font, it creates a system font unless this flag is set.

Return value

The **ReallyCreateFont** function returns the handle to the created font. If a font could not be created, the function returns NULL.

Supporting Different Versions of Windows

You can include the font utilities in an application or a DLL. In an application, the font utilities call **MakeProcInstance** in the font enumeration procedure; in a DLL, they do not call this function.

You can compile the font utilities to run under Windows 3.0, Windows 3.1, or Windows NT version 3.1. The font utilities are based on the Windows 3.1 application programming interface (API); some extra steps were taken to support the Windows 3.0 and Win32™ APIs.

The font utilities use two techniques to get information about the available fonts:

- To search for a font, the font utilities call **EnumFonts** when running under Windows 3.0, and **EnumFontFamilies** when running under Windows 3.1.
- To get information about the current TrueType font, the font utilities call **GetOutlineTextMetrics**. The functions do not call **GetOutlineTextMetrics** for non-TrueType fonts or when running under Windows 3.0.

Supporting Windows 3.0

The font utilities use three functions that do not exist in Windows 3.0: **GetOutlineTextMetrics**, **GetCharABCWidths**, and **GetTextExtentPoint**. These functions operate on TrueType fonts, which were introduced in Windows 3.1. The font utility functions operate correctly if called under Windows 3.0, or for non-TrueType fonts. However, if an application links calls to **GetOutlineTextMetrics**, **GetCharABCWidths**, and **GetTextExtentPoint**, the Windows 3.0 kernel will not load the application. To avoid this problem, you can compile the font utilities so that they use the internal wrapper functions **MyGetOutlineTextMetrics**, **MyGetCharABCWidths**, and **MyGetTextExtentPoint** instead of calling the Windows 3.1 functions directly. These wrapper functions call **GetProcAddress** to dynamically link in the functions at run time so the Windows 3.0 kernel can load the application.

In addition, the font utilities call the **MyGetOutlineTextMetrics** and **MyGetCharABCWidths** wrapper functions only when running under Windows 3.1. The font utilities do call **MyGetTextExtentPoint** when running under Windows 3.0, so this wrapper function includes logic to call the Windows 3.0 **GetTextExtent** function instead of the Windows 3.1 **GetTextExtentPoint** function. You can also compile the font utilities so that they do not run under Windows 3.0. In this case, the Windows 3.1 functions are called directly.

Here is how the wrapper functions are set up:

```
#ifdef WORK_IN_WIN30
/* Wrapper functions we use so we can run in 3.0 and 3.1 since
 * GetOutlineTextMetrics, GetCharABCWidths, and GetTextExtentPoint
 * do not exist in 3.0, and the application will not load if
 * kernel 3.0 sees that we're importing externals that don't
 * exist in GDI.
 */

UINT NEAR PASCAL MyGetOutlineTextMetrics(HDC hdc, UINT ui,
                    OUTLINETEXTMETRIC FAR* lpOTM);
BOOL NEAR PASCAL MyGetCharABCWidths(HDC hdc, UINT ui1, UINT ui2,
                    ABC FAR* lpABC) ;
BOOL NEAR PASCAL MyGetTextExtentPoint(HDC, LPCSTR, int, SIZE FAR*);

/*
 * You probably want to make this a global in your app. There is,
 * after all, no reason to call GetVersion all the time.
 */

#define fWin30 ((BOOL)(LOWORD( GetVersion() ) == 0x0003))

#else

#define fWin30 FALSE
/*
 * If we are a 3.1-only app, just call the 3.1 functions directly!
 */
#define MyGetOutlineTextMetrics GetOutlineTextMetrics
#define MyGetCharABCWidths      GetCharABCWidths
#define MyGetTextExtentPoint    GetTextExtentPoint

#endif
```

Another step the font utilities take for Windows 3.0 compatibility is to call **EnumFonts** when running under Windows 3.0 and **EnumFontFamilies** when running under Windows 3.1. The font utilities do this through run-time dynamic linking—by calling **GetProcAddress** and using the Windows function that corresponds to the correct version of Windows. Here is how the font utilities call either **EnumFonts** or **EnumFontFamilies**:

```
int  (WINAPI *lpfnEnumFont)(HDC,LPSTR,FONTENUMPROC,LPARAM) ;
/*
 * On 3.0 call EnumFonts, on 3.1 call EnumFontFamilies.
 *
 * EnumFonts is exported from GDI at ordinal #70.
 * EnumFontFamilies is exported from GDI at ordinal #330
 * (3.1 and later).
 */

(FARPROC)lpfnEnumFont = GetProcAddress( GetModuleHandle( "GDI" ),
   (LPCSTR)(fWin30 ? MAKEINTRESOURCE( 70 )
                   : MAKEINTRESOURCE( 330 )) ) ;

(*lpfnEnumFont)( hdcCur, lpszFace, lpfn, (LPARAM)(LPVOID)&elf ) ;
```

Supporting the Win32 API

Supporting the Win32 API is much simpler than supporting Windows 3.0. The font utilities do not need Windows 3.0 support for Win32, so they can call the **GetOutlineTextMetrics**, **GetCharABCWidths**, **GetTextExtentPoint**, and **EnumFontFamilies** functions in Windows 3.1 directly without using the wrapper functions. For example:

```
#ifdef WIN32
EnumFontFamilies( hdcCur, lpszFace, lpfn,(LPARAM)(LPVOID)&elf ) ;
#else
(*lpfnEnumFont)( hdcCur, lpszFace, lpfn, (LPARAM)(LPVOID)&elf ) ;
#endif
```