QBWinFont

Routines to Display Window's Raster Fonts in QuickBasic and DOS QBasic

Unregistered Version 1.0

Nybble Software
P.O. Box 7213
Tuscaloosa, AL 35486

QBWinFont
Unregistered Version 1.0
Release Date: March 1, 1994

Nybble Software
P.O. Box 7213
Tuscaloosa, AL 35486

Liabilities: This software is provided AS IS. No expressed or implied warranty of any kind exists, including but not limited to the implied warranty of fitness for any purpose. Users of the software assume all responsibility for its application and performance.

Nybble Software makes no statements or claims concerning the legal issues involved in using Windows' fonts for your programs. One should assume all font resources are copyrighted, unless specifically stated otherwise in the font documentation.

Questions, comments, or suggestions are welcomed
at the above address or at the e-mail address:
joe@galcit.caltech.edu

1 Introduction

QBWinFont is a set of QB routines which allow Windows' raster (bitmap) fonts to be displayed in graphics modes in QuickBasic or DOS QBasic. The **QBWinFont** routines can load and display fonts from individual font files (FNT), font resource files (FON), and other Windows' resource files (EXE, DLL, *etc*.). Fonts are loaded into easy to access integer arrays. All routines are coded entirely in QB for convenient use and modification in either QuickBasic and DOS QBasic. Extensive documentation is included along with an example program for each routine.

QBWinFont is shareware. If you find this product useful, please register it. Instructions and an order form are included at the end of this manual.

2 Windows Font Files

Windows uses three types of fonts: raster (or bitmap) fonts, vector fonts, and TrueType fonts. A bitmap font stores characters as a series of bitmaps for each row of a character. A vector font stores characters as a series of line segment to be drawn. TrueType fonts, introduced in Windows 3.1, are complex outline fonts which can be scaled to any point size and look the same when printed as when viewed on the screen. The first two types of fonts are called screen fonts and plotter fonts respectively in the Windows documentation.

Bitmap fonts can be further divided into fixed (or monospace) and proportional fonts. For fixed fonts all characters have the same width, such as with the DOS system fonts. For proportional fonts, the widths of the characters vary. For example in a proportional font, the letter *i* would be much narrower than the letter *W*. Proportional fonts look much more professional, although arguably at a cost of readibility for small font sizes.

FNT files are files which contain store individual bitmap or vector fonts. These individual fonts are generally found compiled into FON resource files which contain one or more related fonts. However, FON files are just one type of Windows' resource files. Executable files (EXE) and dynamic link libraries (DLL) are also examples of resource files.

3 Getting Started

QBWinFont can only be used with raster (bitmap) fonts. The **QBWinFont** subroutines can load fonts from FNT files, FON files, or other types of resource files, although resource files other than FON files rarely contain any font resources. In addition, the included QB program **FileInfo** can be used to browse through resource files.

3.1 QuickBasic

In QuickBasic 4.x, the **QBWinFont** subroutines can be easily accessed by loading the file **qbwinfnt.bas** as a separate program module. When running the example programs, remember to be sure the **qbwinfnt.bas** file is loaded. This will be done automatically if the

Open command is used to load the programs.

3.2 DOS QBasic Users

DOS QBasic does not allow loading more than one program module. This means programs which use the **QBWinFont** subroutines will have to include the source code of the subroutines directly in the program. The easiest way to do this is to copy the file **qbwinfnt.bas** to a work file and then program directly in the work file.

To run the **QBWinFont** example programs, they must be combined with the subroutines in the file **qbwinfnt.bas**. First copy the file **qbwinfnt.bas** to a work file. Next Open the example program you would like to run in QBasic, select the entire program text with the cursor or mouse, and choose Copy in the Edit menu. Then, Open the work file (which contains all the subroutines), and Paste the entire example program into the work file at the end of the main program. The work file can then be executed.

3.3 Loading Fonts

The **QBWinFont** routines load bitmap fonts stored in Windows' font files (FNT) or resource files (FON, EXE, DLL, *etc*.), although few resource files other than FON files contain any fonts. To load a font from a FNT file, call the **LoadFontFile** with a file name and a font array in which to store the font. For example, to load the font in the included file **sample.fnt**:

```
REDIM FontArray%(1)
CALL LoadFontFile("SAMPLE.FNT", FontArray%(), RetCode%, RetMsg$)
```

Notice that the font array is created with `REDIM` to be `DYNAMIC`, so that the load routine can resize the array.

To load a font from a FON resource file, call the **LoadRsrcFileFont** with a file name and a font array in which to store the font. Only one additional step is needed that was not needed before: since the FON file may contain more than one font, a font number must be specified. For example, to load the first (and only) font in the included font resource file **sample.fon**:

```
REDIM FontArray%(1)
FontNum% = 1
CALL LoadRsrcFileFont("SAMPLE.FON", FontNum%, FontArray%(), _
        RetCode%, RetMsg$)
```

There are several ways to see how many fonts of what size are in a FON resource file. First, the included QB program **FileInfo** will scan a resource file and print information on all the fonts and font directories found. Alternately, one could go into Windows and load the font using the Fonts applet in the Control Panel. For bitmap fonts, the sizes for all the installed fonts are listed. Finally of course, one could just keep trying successive font numbers with the **LoadRsrcFileFont** subroutine until an error occurs.

3.4 Displaying Text from Loaded Fonts

Once the fonts are loaded, the subroutines **DispChar** and **FastChar** can be used to display

single characters from the font, and **DispString** and **FastString** to display text strings. For example, we'll display "Hello World" from one of the fonts loaded above. We'll use 640x480 graphics mode with the text in foreground color 15 (white) and background color 1(blue) at location (x,y) = (100,100):

```
SCREEN 12
CALL DispString("Hello World", 15, 1, 100, 100, FontArray%())
```

Note that since font display is done using the QB `LINE` command, we have to be in a graphics mode.

The subroutines **FastChar** and **FastString** display faster by only using a foreground color. This allows the main loop to be optimized for a major speed increase. Unless background colors are needed, you will probably want to use the faster routines.

All the display routines return the x-coordinate set to the start of the next character. This means that if you call the routines consecutively without redefining the x-coordinate, characters will be spaced correctly. For example, the following commands are identical to displaying the digits as a single string, since `X%` is updated by the call to **FastString**:

```
X% = 100
CALL FastString("01234", 15, X%, 100, FontArray%())
CALL FastString("56789", 15, X%, 100, FontArray%())
```

3.5 Saving Fonts

Font arrays can be saved with the routine **BSAVEFont** which uses the QB command `BSAVE` to store the entire font array. For example, to save the font array loaded above in the file **test.bin**:

```
CALL BSAVEFont("TEST.BIN", FontArray%())
```

A font in a file created with **BSAVEFont** can be loaded quickly with **BLoadFont**, much quicker than the font can be loaded from a FNT or resource file. If we wanted to load the file **dtch_bld.bin** which is included with the package:

```
REDIM BoldFont%(1)
CALL BLOADFont("DTCH_BLD.BIN", BoldFont%(), RetCode%)
```

Note that once again we've used `REDIM` so the font array can be resized by the subroutine.

4 Summary of QBWinFont Routines

DECLARE SUB BLOADFont(FlName$, FontArray%(), RetCode%)

Subroutine to BLOAD a font array from the specified file, which must have been created with BSAVE. The font array must by DYNAMIC to allow for redimensioning.

DECLARE SUB BSAVEFont(FlName$, FontArray%())

Subroutine to store a font array to the specified file using BSAVE.

DECLARE SUB DispChar(Char%, FClr%, BClr%, X%, Y%, FontArray%())

Subroutine to display the specified character with upper-left corner at (X%,Y%) in fore- and background colors. The x-coordinate of the start of the next character is returned in X%.

DECLARE SUB DispString(Text$, FClr%, BClr%, X%, Y%, _
FontArray%())

Subroutine to display the given text string with the upper-left corner of the first character at (X%,Y%) in fore- and background colors. The x-coordinates of the start of the next character is returned in X%.

DECLARE SUB FastChar(Char%, FClr%, X%, Y%, FontArray%())

Subroutine to display the specified character with upper-left corner at (X%,Y%) in foreground color only. The coordinates of the start of the next character are returned in X%.

DECLARE SUB FastString(Text$, FClr%, X%, Y%, FontArray%())

Subroutine to display the given text with upper-left corner of the first character at (X%,Y%) in foreground color only. The x-coordinates of the start of the next character is returned in X%.

DECLARE SUB LoadFontFile(FlName$, FontArray%(), RetCode%, _
RetMsg$)

Subroutine to create a font array from the given Windows' font (FNT) file. The font array must by DYNAMIC to allow for redimensioning.

DECLARE SUB LoadRsrcFileFont(FlName$, FontNum%, FontArray%(), _
RetCode%, RetMsg$)

Subroutine to create a font array from the specified font number in a Windows' resource file, typically a FON file. The font array must by DYNAMIC to allow for redimensioning.

DECLARE FUNCTION WidthString%(Text$, FontArray%())

Function to return the width in pixels of a string in the given font.

5 Finding Fonts

Nybble Software makes no statements or claims concerning the legal issues involved in using Windows' fonts for your programs. One should assume all font files are copyrighted, unless specifically stated otherwise in documentation provided with the fonts.

(As a dissenting view, we note Steve Rimmer in *Super VGA Graphics: Programming Secrets* states that although font names can be (and are) copyrighted, the actual bitmap patterns cannot be copyrighted. He states: "In theory, this means you can use any fonts you acquire from any source in your applications without having to be concerned with who created them." Please note however that he does go on to constrain this statement with several cautions.)

5.1 Locating FON files

With the advent of TrueType fonts in Windows 3.1, bitmap fonts are becoming increasingly hard to find. Some examples are available at **ftp.cica.indiana.edu** (or one of its many mirror sites) in the directory **\pub\pc\win3\fonts**. Although the copyright status of several of these fonts is not clear, most are labeled as freeware or shareware.

When handling Windows' resource files, the freeware program **RX** might be helpful. The program along with source code can be found in **rx.zip** at **ftp.cica.indiana.edu** in the directory **\pub\pc\win3\util**. The program can be used to extract individual resources from Windows' compiled resource files. For example, individual fonts can be extracted from FON files into FNT files.

5.2 Converting TrueType Fonts

QBWinFont only supports bitmap fonts, however separate programs are available which will convert a TrueType font at a specified point size to a bitmap font. One example is the freeware program **SysFon** which converts TrueType fonts to bitmap FON resource files. This program with source code is available by anonymous ftp from **ftp.cica.indiana.edu** (or one of its many mirror sites) in the directory **\pub\pc\win3\fonts**. Using **SysFon**, TrueType font characters can be accurately reproduced at a given point size, except the bitmap fonts do not allow for overhanging characters. This can cause problems for some fonts, particularly italic fonts.

TrueType fonts are the easist to find font files for Windows. A nice selection of TrueType fonts already comes with Windows, and collections of fonts are available commercially in the range of $1 per font family. Anonymous ftp sites and BBS's often contain TrueType fonts, although many of these are foreign language or novelty fonts.

5.3 Other

Of course a wide range of commercial programs exist which allow creating, converting, and editing fonts for Windows. For example, Adobe Type Manager (ATM) for Windows is a

powerful package, and a wide variety of ATM fonts are available by anonymous ftp or commercially.

6 Internal Workings

6.1 Font Arrays

The font data is read in from the resource file and stored in an integer array. Using an integer array allows the font header information to be read quickly as elements of the array. However to save memory the bitmap data is stored directly into memory, and not aligned into array elements. The bitmap data must therefore be accessed using `PEEK`.

The font array header contains information about the font which needs to be quickly accessed for displaying characters. These values are placed in array elements for faster access. The header consists of the elements 0 through 10 of the font array defined as follows:

ELEMENT	VALUE
FontArray%(0)	Font height in pixels
1	Number of characters in font char set
2	First character in font char set
3	Last character in font char set
4	Default character
5	Break character
6	Maximum character width in pixels
7	Vertical spacing
8	Ascent (distance from baseline to char top)
9	Horizontal padding (to add between chars)
10	Vertical padding (to add between lines)

Upon creating a font array from a font or resource file, the font height, first/last/default/break characters, maximum character width, and ascent are taken directly from the file values. The number of characters is calculated from the first and last character values. The vertical spacing is set to the character height. Finally, the vertical and horizontal padding values are set to zero.

Following the font array header, the width in pixels of each character is stored in the font array elements 11 to 11+N-1, where N is the total number of characters in the font character set. Therefore the width of a character with specified `ASCII` code (`char%`) can be found using the formula:

```
RelChar% = char% - FontArray%(2)          '...relative char #
CharWidth% = FontArray%(11 + RelChar%)    '...char width in pixels
```

Note that the character number relative to the first character is used to access the correct array element.

Following the character width information, the offset to the start of each character's bitmap data is stored in font array elements 11+N to 11+2*N-1, where N is the total number of

characters in the font character set. The offset of the bitmap data for a given character with a given ASCII code (char%) can be found using the formula:

```
RelChar% = char% - FontArray%(2)          '...relative char #
Ptr& = FontArray%(11 + FontArray%(1) + RelChar%)
```

Note that both the character number relative to the first character and the total number of characters (FontArray%(1)) are needed to access the correct array element. The situation is further complicated by the fact that the unsigned offset integer is stored as a signed value. For large fonts, this can lead to the offset (Ptr&) being negative. Therefore the long-integer pointer must be converted to an unsigned integer value before using:

```
IF (Ptr& < 0) THEN Ptr& = 65536 + Ptr&    '...adjust offset
```

Once the offset is found, the bitmap data can be accessed using PEEK. For example, the first row of bitmap data is found using the above pointer:

```
Pattern% = PEEK(Ptr& + VARPTR(FontArray%(0)))
```

This gives the first byte of the first row of the character. The bitmap data then continues down the character until all rows in the height of the character are completed. If the character is over a byte (8 pixels) wide, the bitmap data continues with the second byte of the first row. The data then continues down the second byte of the character, and so on for the number of bytes in the width of the character.

6.2 Font Resource Data

The font header for Windows' font resources is shown below defined as a QB data type. Note that integers (2-bytes) can be read in directly. However for large values care has to be taken since QB reads in the values as signed integers. QB does not have a byte type, so bytes have to be read in as strings of length one. To use the byte values, the one character strings are converted to integers with the ASC command.

```
TYPE FontType
 Version        AS INTEGER
 Size           AS LONG
 Copyright      AS STRING * 60
 FType          AS INTEGER
 Pnt            AS INTEGER
 VertRes        AS INTEGER
 HorizRes       AS INTEGER
 Ascent         AS INTEGER
 IntLeading     AS INTEGER
 ExtLeading     AS INTEGER
 Italic         AS STRING * 1
 Underline      AS STRING * 1
 StrikeOut      AS STRING * 1
 Weight         AS INTEGER
 CharSet        AS STRING * 1
 PixWidth       AS INTEGER
 PixHeight      AS INTEGER
 PitchFamily    AS STRING * 1
```

```
    AvgWidth        AS INTEGER
    MaxWidth        AS INTEGER
    MaxWidth        AS INTEGER
    FirstChar       AS STRING * 1
    LastChar        AS STRING * 1
    DefaultChar     AS STRING * 1
    BreakChar       AS STRING * 1
    WidthBytes      AS INTEGER
    Device          AS LONG
    Face            AS LONG
    BitsPointer     AS LONG
    BitsOffset      AS LONG
    Reserved1       AS STRING * 1
  END TYPE
```

The entries are briefly described below. Only the definitions relevant to bitmap fonts are mentioned here, some of the values have different meanings for vector fonts.

Version Equals 0200h for version 2.x or 0300h for version 3.x. Version 3.x fonts are stored in a slightly different format and are not supported by **QBWinFont**.

Size Size of the font resource in bytes.

Copyright Copyright.

FType Font type. If bit 0 is set the font is a vector font, otherwise the font is a raster (bitmap) font. **QBWinFont** does not support vector fonts.

Pnt Point size of the font.

VertRes Recommended vertical resolution in pixels per inch.

HorizRes Recommended horizontal resolution in pixels per inch.

Ascent Distance from character baseline to the top of the character cell. This value is important for aligning different size fonts on the same baseline.

IntLeading Internal leading. Distance above top of character to top of character cell. May be zero.

ExtLeading External leading. Recommended distance above the character cell between font rows. May be zero.

Italic Set to 1 for italic font.

Underline Set to 1 for underline font.

StrikeOut Set to 1 for a strike-out font.

Weight Weight of the font ranging from 0 to 1000, the higher the value the denser the font. (Common values are 0 = don't care, 400 = regular, 700 = bold.)

CharSet Type of font character set: 0 = ANSI character set, 2 = symbolic character set, 255 = OEM for DOS compatible fonts.)

PixWidth Width of font in pixels. For proportional fonts this value is zero, and each character's width is specified individually.

PixHeight Height of font in pixels.

PitchFamily Pitch and family. The font family is stored in the upper nibble: 0 = don't care, 10h = roman, 20h = swiss, 30h = modern, 40h = script, 50h = decorative. The pitch is stored in the lowest bit. If bit 0 is set, the font pitch is variable.

AvgWidth Average width, typically the width of a capital X.

MaxWidth Maximum character width.
FirstChar ASCII value of first character (0 - 255) in font. A character range is required since many fonts do not define all possible characters.
LastChar ASCII value of last character (0 - 255) in font.
DefaultChar Default character, relative to the first char, to display for characters outside the range of characters defined in the font.
BreakChar Default character, relative to the first char, to use as a word seperator, typically the space character.
WidthBytes Width in bytes in each row of a raster font's bitmap.
Device Offset in bytes from beginning of font resource to font's device name stored as a string. May be null.
Face Offset in bytes from beginning of font resource to font's face name stored as a string.
BitsPointer Unused and set to zero.
BitsOffset Offset in bytes from beginning of font resource to start of font bitmap data.
Reserved Reserved.

Note that not all of the font information is stored in the QB font array. Much of the font information is provided to help programs identify a particular font, and to locate the available font closest to a particular specification. For **QBWinFont**, the contents of font arrays loaded are assumed to be known to the program, and the main elements stored are values which need to be rapidly accessed for character display.

6.3 Customizing the Routines For Speed and Memory

QBWinFont can be optimized to improve memory or speed if one limits the types of fonts handled or writes custom versions of the routines. Of course, the appropriate solution to the speed problem is Assembly or C routines, but that's another matter.

About the only improvement to make in the speed of the routines in QB is at the expense of memory. When loading the fonts, each byte of bitmap data could be shifted into the upper byte of an integer and stored in an array element. This would allow line patterns to be taken directly from font array elements instead of using `PEEK`. Of course, this would double the memory required to store the font's bitmap data.

Recall in fixed fonts, all characters are the same width. If your program only uses fixed fonts, the character width and bitmap offset information could be removed from the font array to save space. The maximum character width entry in the font array could then be used as the character width. Since all character's bitmaps would be the same length, the bitmap offset could be calculated by multiplying the character number by the bitmap length per character.

If memory is a problem, consider writing a routine to remove unused characters from a font. For instance, if a program only uses the upper case letters of a font for a title screen, all the remaining characters could be removed. To do this, update the first, last, and default characters in the font array header. Then relocate all the character widths and offsets

downward in the font array and recompute the bitmap offsets to reflect the removed characters. Modifyng the **LoadRsrcFileFont** routine to only load a certain range of characters might be an easier alternative.

Fixed fonts less than a byte (8 pixels) wide can be installed as the graphics font using the video bios. Then the fonts would automatically be used by `PRINT` statements. This would lead to much faster display, although only at row, column locations and not at any x,y location.

6.4 Unsupported Font Resources

The font resources established by Windows have several features which are not supported by **QBWinFont**. Only file-based fonts are supported, not fonts stored at an absolute memory address, as with fonts realized by device drivers.

Most importantly, **QBWinFont** does not support font resources stored in Version 3.x format. The main new feature of the Version 3.x format is the ability to store font resources larger than 64K. Supporting these in QB would require a good bit more complexity in the display routines, and `BSAVE` and `BLOAD` could not be used directly to save and load files. This restriction will hopefully not be a problem for users, since during program development no Version 3.x bitmap font resources were encountered. Besides, a number of Windows programs also do not support font resources over 64K. (This does not mean that the resource file cannot be over 64K, only that an individual font resource cannot be over 64K.)

6.5 Bugs, Limitations, and Improvements

Known bugs and limitations include:

* Routines do not handle Version 3.x bitmap fonts, even if the fonts are under 64K.
* Load routines do not provide an error message for attempting to load non-file-based fonts.

Needed improvements include:

* Some fonts do not correctly set the default character. The routines should attempt to handle this more gracefully.
* A routine is needed to remove unwanted characters from a font array to save memory.
* Program should be able to write a FNT file from a resource file font.

7 References

Rimmer, Steve. *Super VGA Graphics: Programming Secrets*. New York: Windcrest, 1993.
Swan, Tom. *Inside Windows File Formats*. Sams Publishing, 1993.

8 Acknowledgements

The sample font resource file **sample.fon** included with this package was created using

SysFon from the TrueType Kashmir font by Adam Albright and modified by Matt Hill. The sample font file **sample.fnt** was created using **RX** from the fonts included in the **woafon14.zip** package by Steve Karrer. All files were downloaded by anonymous ftp from **ftp.cica.indiana.edu**.

Appendix A
List Of Routines

BLOADFont

Subroutine to BLOAD the given font array from the specified file. The size of the file to load is obtained first, so that the font array can be redimensioned to the size of the data. If the file is of size zero, the return code is set to True (-1), otherwise the return code is set to False (0).

To allow redimensioning of the array, the font array must be DYNAMIC. This can be done by using the $DYNAMIC metacommand in QB, or more simply, by creating the original array with REDIM instead of DIM. (See the example below.)

The file to load with BLOAD must have been created with BSAVE. BSAVE adds a 7-byte header containing size info, which is used by BLOAD to determine the amount of data to load.

Format:

```
DECLARE SUB BLOADFont(FlName$, FontArray%(), RetCode%)
```

Arguments:

```
    FlName$ = file name from which to BLOAD the font array
FontArray% = integer array containing font data, must be
             a DYNAMIC array to allow redimensioning
  RetCode% = return code, set to True (-1) for file size zero
```

Example:

The following example demonstrates using BLOAD to load a font array from a disk file (**dtch_bld.bin**) previously created with BSAVE.

```
REM:  EX_BLDFT.BAS, Unregistered Version 1.0
REM:  Example of using BLOAD to load a font array.

DECLARE SUB BLOADFont(FlName$, FontArray%(), RetCode%)
DECLARE SUB FastString(Text$, FClr%, X%, Y%, FontArray%())

'...setup a VGA screen mode...
SCREEN 12

'...dimen array for font data (use REDIM so its DYNAMIC)...
REDIM FontArray%(1)

PRINT : PRINT "BLOAD'ing a font from DTCH_BLD.BIN..."

'...load the font in one of the example BIN files...
CALL BLOADFont("DTCH_BLD.BIN", FontArray%(), RetCode%)

'...check return code, catches non-existent file...
IF (RetCode% <> 0) THEN STOP

'...get the height for below...
Ht% = FontArray%(0)
```

```
'...display the capitals, lower case, & numbers...
CALL FastString("ABCDEFGHIJKLMNOPQRSTUVWXYZ", 4, 80, 100, FontArray%())
CALL FastString("abcdefghijklmnopqrstuvwxyz", 4, 80, 100 + Ht%, _
        FontArray%())
CALL FastString("1234567890", 4, 80, 100 + 2 * Ht%, FontArray%())

END
```

BSAVEFont

Subroutine to BSAVE the given font array to the specified file. Using BSAVE to store a file allows the font to be stored individually in a smaller file than the original FON file. Since the font data has already been loaded into the array elements, a font array stored with BSAVE also allows for faster loading.

Format:

```
DECLARE SUB BSAVEFont(FlName$, FontArray%())
```

Arguments:

```
    FlName$ = file name in which to BSAVE the font array
FontArray% = integer array containing font data
```

Example:

The following example demonstrates using BSAVE to save a font array to a disk file. The font array is loaded from the first font in the example FON file **sample.fon**, a sample of the font is displayed, and then the font is stored in the file **test.bin** using BSAVE.

```
REM:  EX_BSVFT.BAS, Unregistered Version 1.0
REM:  Example of using BSAVE to store a font array to disk.

DECLARE SUB BSAVEFont(FlName$, FontArray%())
DECLARE SUB FastString(Text$, FClr%, X%, Y%, FontArray%())
DECLARE SUB LoadRsrcFileFont(FlName$, FontNum%, FontArray%(), _
        RetCode%, RetMsg$)

'...setup a VGA screen mode...
SCREEN 12

'...dim array for font data (use REDIM so its DYNAMIC)...
REDIM FontArray%(1)

PRINT : PRINT "Loading a font from SAMPLE.FON..."

'...load the font in the example FON file...
CALL LoadRsrcFileFont("SAMPLE.FON", 1, FontArray%(), RetCode%, _
        RetMsg$)

IF (RetCode% <> 0) THEN
  PRINT "***** ERROR: RetCode% = "; RetCode%
  PRINT "***** "; RetMsg$
  STOP
END IF

'...display a sample of the font...
CALL FastString("Sample of Font ", 4, 10, 60, FontArray%())

LOCATE 8, 1: PRINT "BSAVE font array to file TEST.BIN...";
```

```
'...save the file using BSAVE for quick future access...
CALL BSAVEFont("TEST.BIN", FontArray%())

PRINT "Completed!": PRINT

PRINT "See the example EX_BLDFT.BAS for an example of loading"
PRINT "a font file created with BSAVE."

END
```

DispChar

Subroutine to display a single character using the given font array at any (x,y) in the given foreground and background colors. The x-coordinate is returned set to the x-location of the next character.

When calling the routines to display characters or strings, the current screen mode must be a graphics mode. The coordinates and colors should be within the range available for the current screen mode. If the foreground or background color is negative, the character foreground or background is not drawn.

To draw a character in only a foreground color, the routine **FastChar** is faster. To display a string, see the routines **DispString** and **FastString**.

Format:

```
DECLARE SUB DispChar(Char%, FClr%, BClr%, X%, Y%, FontArray%())
```

Arguments:

```
      Char% = ASCII code of character to display
      FClr% = foreground color (negative to not draw foreground)
      BClr% = background color (negative to not draw background)
         X% = x-coordinate of upper-left corner of character,
              returned pointing to start of next character
         Y% = y-coordinate of upper-left corner of character
 FontArray% = integer array containing font data
```

Example:

The following example demonstrates using **DispChar** to display single characters in combinations of foreground and background colors.

```
REM:  EX_DCHAR.BAS, Unregistered Version 1.0
REM:  Example of using DispChar to display characters.

DECLARE SUB DispChar(Char%, FClr%, BClr%, X%, Y%, FontArray%())
DECLARE SUB LoadRsrcFileFont(FlName$, FontNum%, FontArray%(), _
        RetCode%, RetMsg$)

'...setup a VGA screen mode...
SCREEN 12

'...dim array for font data (use REDIM so its DYNAMIC)...
REDIM FontArray%(1)

PRINT : PRINT "Loading a font from SAMPLE.FON..."

'...load the font in the example FON file...
CALL LoadRsrcFileFont("SAMPLE.FON", 1, FontArray%(), RetCode%, _
        RetMsg$)
```

```
'...check return code, catches non-existent file...
IF (RetCode% <> 0) THEN STOP

'...display an "A" in foreground and background color...
CALL DispChar(ASC("A"), 4, 7, 100, 100, FontArray%())

'...display a "B" in foreground color only...
CALL DispChar(ASC("B"), 4, -1, 100, 130, FontArray%())

'...display a "C" in background color only...
CALL DispChar(ASC("C"), -1, 7, 100, 160, FontArray%())

END
```

DispString

Subroutine to display a string using the given font array at any (x,y) in the given foreground and background colors. The x-coordinate is returned set to the x-location of the next character.

When calling the routines to display characters or strings, the current screen mode must be a graphics mode. The coordinates and colors should be within the range available for the current screen mode. If the foreground or background color is negative, the character foreground or background is not drawn.

To draw a string in only a foreground color, the routine **FastString** is faster. To display only a single character, see the routines **DispChar** and **FastChar**.

Format:

```
DECLARE SUB DispString(Text$, FClr%, BClr%, X%, Y%, _
   FontArray%())
```

Arguments:

```
     Text$ = text to display
     FClr% = foreground color (negative to not draw foreground)
     BClr% = background color (negative to not draw background)
        X% = x-coordinate of upper-left corner of character,
             returned pointing to start of next character
        Y% = y-coordinate of upper-left corner of character
FontArray% = integer array containing font data
```

Example:

The following example demonstrates using **DispString** to display text in combinations of foreground and background colors.

```
REM:  EX_DSTRG.BAS, Unregistered Version 1.0
REM:  Example of using DispString to display text.

DECLARE SUB DispString (Text$, FClr%, BClr%, X%, Y%, FontArray%())
DECLARE SUB LoadRsrcFileFont (FlName$, FontNum%, FontArray%(), _
        RetCode%,

'...setup a VGA screen mode...
SCREEN 12

'...dim array for font data (use REDIM so its DYNAMIC)...
REDIM FontArray%(1)

PRINT : PRINT "Loading a font from SAMPLE.FON..."

'...load the font in the example FON file...
CALL LoadRsrcFileFont("SAMPLE.FON", 1, FontArray%(), RetCode%, RetMsg$)
```

```
'...check the return code...
IF (RetCode% <> 0) THEN STOP

'...display string in foreground and background color...
CALL DispString(" Foreground & Background ", 4, 7, 100, 100, _
        FontArray%())

'...text foreground only, negative color forces no background...
CALL DispString(" Foreground Only ", 4, -1, 100, 135, FontArray%())

'...text background only, negative color forces no foreground...
CALL DispString(" Background Only ", -1, 7, 100, 170, FontArray%())

END
```

FastChar

Subroutine to display a single character using the given font array at any (x,y) in the given foreground color. The x-coordinate is returned set to the x-location of the next character.

When calling the routines to display characters or strings, the current screen mode must be a graphics mode. The coordinates and colors should be within the range available for the current screen mode.

This routine is faster than **DispChar**, since the main loop can be optimized for foreground color only. To draw a single character in both foreground and background colors, see the routine **DispChar.** To display a string, see the routines **DispString** and **FastString**.

Format:

```
DECLARE SUB FastChar(Char%, FClr%, X%, Y%, FontArray%())
```

Arguments:

```
      Char% = ASCII code of character to display
      FClr% = foreground color
         X% = x-coordinate of upper-left corner of character,
              returned pointing to start of next character
         Y% = y-coordinate of upper-left corner of character
 FontArray% = integer array containing font data
```

Example:

The following program does a timing comparison between **DispChar** and **FastChar**.

```
REM:  EX_FSTCH.BAS, Unregistered Version 1.0
REM:  Example of using FastChar to display characters.

DECLARE SUB BLOADFont(FlName$, FontArray%(), RetCode%)
DECLARE SUB DispChar(Char%, FClr%, BClr%, X%, Y%, FontArray%())
DECLARE SUB FastChar(Char%, FClr%, X%, Y%, FontArray%())

'...setup a VGA screen mode...
SCREEN 12

'...dimen array for font data (use REDIM so its DYNAMIC)...
REDIM FontArray%(1)

PRINT : PRINT "BLOAD'ing a font from DTCH_BLD.BIN..."

'...load the font in one of the example BIN files...
CALL BLOADFont("DTCH_BLD.BIN", FontArray%(), RetCode%)

'...check return code, catches non-existent file...
IF (RetCode% <> 0) THEN STOP

'...draw A to Z, 25 times using DispChar (fgnd & bgnd)...
```

```
Start1 = TIMER
FOR Dupl% = 1 TO 25
  X% = 50 + Dupl%: Y% = 60 + Dupl%
  FOR Char% = 65 TO 90
    CALL DispChar(Char%, 7, 8, X%, Y%, FontArray%())
  NEXT Char%
NEXT Dupl%
End1 = TIMER

'...draw A to Z, 25 times using DispChar (fgnd only)...
Start2 = TIMER
FOR Dupl% = 1 TO 25
  X% = 50 + Dupl%: Y% = 100 + Dupl%
  FOR Char% = 65 TO 90
    CALL DispChar(Char%, 7, -1, X%, Y%, FontArray%())
  NEXT Char%
NEXT Dupl%
End2 = TIMER

'...display letters A to Z, 25 times using FastChar...
Start3 = TIMER
FOR Dupl% = 1 TO 25
  X% = 50 + Dupl%: Y% = 140 + Dupl%
  FOR Char% = 65 TO 90
    CALL FastChar(Char%, 7, X%, Y%, FontArray%())
  NEXT Char%
NEXT Dupl%
End3 = TIMER

'...skip down a ways and print results...
LOCATE 15, 1: PRINT "Approximate times for 25 displays: "
PRINT USING "   DispChar (fgnd, bgnd): ##.##"; End1 - Start1
PRINT USING "    DispChar (fgnd only): ##.##"; End2 - Start2
PRINT USING "                FastChar: ##.##"; End3 - Start3

END
```

FastString

Subroutine to display a string using the given font array at any (x,y) in a foreground color only. The x-coordinate is returned set to the x-location of the next character.

When calling the routines to display characters or strings, the current screen mode must be a graphics mode. The coordinates and colors should be within the range available for the current screen mode.

This routine is faster than **DispString**, since the main loop is optimized for foreground color only. To display text in both foreground and background colors, see the routine **DispString.**

Format:

```
DECLARE SUB FastString(Text$, FClr%, X%, Y%, FontArray%())
```

Arguments:

```
      Text$ = text to display
      FClr% = foreground color
         X% = x-coordinate of upper-left corner of character,
              returned pointing to start of next character
         Y% = y-coordinate of upper-left corner of character
 FontArray% = integer array containing font data
```

Example:

The following example demonstrates using **FastString** to display text from several of the fonts in **\windows\system\sseriff.fon**, a font file distributed with Windows 3.x. (You may need to change the directory for your system.)

```
REM:  EX_FSTST.BAS, Unregistered Version 1.0
REM:  Example of using FastString to display text.

DECLARE SUB FastString(Text$, FClr%, X%, Y%, FontArray%())
DECLARE SUB LoadRsrcFileFont(FlName$, FontNum%, FontArray%(), _
        RetCode%, RetMsg$)

'...setup a VGA screen mode...
SCREEN 12

'...set file name (change to match your system if necessary)...
FlName$ = "\windows\system\SSERIFF.FON"

'...dim arrays for font data (use REDIM so they're DYNAMIC)...
REDIM Serif8pt%(1), Serif10pt%(1), Serif12pt%(1)

PRINT : PRINT "Loading fonts from "; FlName$; "..."

'...load the eight-point font (#1 in file)...
CALL LoadRsrcFileFont(FlName$, 1, Serif8pt%(), RetCode%, RetMsg$)
IF (RetCode% <> 0) THEN STOP
```

```
'...load the ten-point font (#2 in file)...
CALL LoadRsrcFileFont(FlName$, 2, Serif10pt%(), RetCode%, RetMsg$)
IF (RetCode% <> 0) THEN STOP

'...load the twelve-point font (#3 in file)...
CALL LoadRsrcFileFont(FlName$, 3, Serif12pt%(), RetCode%, RetMsg$)
IF (RetCode% <> 0) THEN STOP

'...display a sample string in each font...
CALL FastString(" MS Sans Serif  8-point ", 2, 20, 70, Serif8pt%())
CALL FastString(" MS Sans Serif 10-point ", 2, 20, 90, Serif10pt%())
CALL FastString(" MS Sans Serif 12-point ", 2, 20, 116, Serif12pt%())

END
```

LoadFontFile

Creates a font array from font data stored in a Windows' font (FNT) file. The font number to load is not specified, since FNT files can only contain a single font.

To allow redimensioning, the font array must be `DYNAMIC`. This can be done by using the `$DYNAMIC` metacommand in QB, or more simply, by creating the original array with `REDIM` instead of `DIM`. (See the example below.)

Return codes and messages are as follows, zero being font loaded successfully. (Note certain return codes are skipped to be consistent with the codes returned by the **LoadRsrcFileFont** subroutine.)

```
0       Null [font successfully loaded]
1       Invalid font file, file too short or not present
5       Bad value of first and/or last font character
6       Font is a vector font, not a bitmap font
7       Font is version number 3, not supported
8       Number of bytes needed negative or greater than 64K
```

Format:

```
DECLARE SUB LoadFontFile(FlName$, FontArray%(), RetCode%, _
        RetMsg$)
```

Arguments:

```
    FlName$ = name of font (FNT) file
FontArray% = integer array for storing font data, must be
             a DYNAMIC array to allow redimensioning
  RetCode% = return code
   RetMsg$ = return message
```

Example:

The following example demonstrates creating a font array from the font data stored in a Windows' font (FNT) file. The user is prompted for the name of the font file, and a short sample of text is displayed. As an example, try the file **sample.fnt** supplied with this program.

```
REM:  EX_LDFNT.BAS, Unregistered Version 1.0
REM:  Example of loading a font from Windows' font (FNT) file.

DECLARE SUB FastString(Text$, FClr%, X%, Y%, FontArray%())
DECLARE SUB LoadFontFile(FlName$, FontArray%(), RetCode%, RetMsg$)

'...setup a VGA screen mode...
SCREEN 12

'...prompt for a file name and file number...
INPUT "Enter Windows' font file (FNT) name (try SAMPLE.FNT): ", FlName$
```

```
'...dim array for font data (use REDIM so its DYNAMIC)...
REDIM FontArray%(1)

PRINT : PRINT "Loading font from "; FlName$; "..."

'...load the specified font from the file...
CALL LoadFontFile(FlName$, FontArray%(), RetCode%, RetMsg$)

IF (RetCode% <> 0) THEN

  PRINT
  PRINT "***** ERROR: RetCode% = "; RetCode%
  PRINT "***** "; RetMsg$

ELSE

  '...display a sample of the font...
  CALL FastString("Sample of Font ", 2, 20, 80, FontArray%())

END IF

END
```

LoadRsrcFileFont

Creates a font array from font data stored in a Windows' resource file, typically an FON file.

Since font resource files can contain multiple fonts, the font number to load must be specified. The routine counts fonts in the order encountered in the resource file starting at one, until the desired font number is reached. If the font number is not reached, an error is returned.

To allow redimensioning, the font array must be `DYNAMIC`. This can be done by using the `$DYNAMIC` metacommand in QB, or more simply, by creating the original array with `REDIM` instead of `DIM`. (See the example below.)

Return codes and messages are as follows, zero being font loaded successfully:

```
0       Null [font successfully loaded]
1       Invalid rsrc file, file too short or not present
2       Invalid rsrc file, ExeHdr signature (MZ) not present
3       Invalid rsrc file, WinInfo signature (NE) not present
4       Desired font number (##) not found
5       Bad value of first and/or last font character
6       Desired font number is a vector font, not a bitmap font
7       Desired font is version number 3, not supported
8       Number of bytes needed negative or greater than 64K
```

Format:

```
DECLARE SUB LoadRsrcFileFont(FlName$, FontNum%, FontArray%(), _
        RetCode%, RetMsg$)
```

Arguments:

```
   FlName$ = name of resource file (typically FON file)
  FontNum% = number of font to load
FontArray% = integer array for storing font data, must be
             a DYNAMIC array to allow redimensioning
  RetCode% = return code
   RetMsg$ = return message
```

Example:

The following example demonstrates creating font arrays from font data stored in a resource file. The user is prompted for the name of the resource file. As an example, try some of the FON files supplied with Windows or **sample.fon** supplied with this program.

```
REM:  EX_LRSRC.BAS, Unregistered Version 1.0
REM:  Example of loading a font from Windows resource files.

DECLARE SUB FastString(Text$, FClr%, X%, Y%, FontArray%())
DECLARE SUB LoadRsrcFileFont(FlName$, FontNum%, FontArray%(), _
        RetCode%, RetMsg$)
```

```
'...setup a VGA screen mode...
SCREEN 12

'...prompt for a file name and file number...
INPUT "Enter resource file name (typically FON file): ", FlName$
INPUT "Enter font number from file to load (try 1): ", FontNum%

'...dim array for font data (use REDIM so its DYNAMIC)...
REDIM FontArray%(1)

PRINT : PRINT "Loading font from "; FlName$; "..."

'...load the specified font from the file...
CALL LoadRsrcFileFont(FlName$, FontNum%, FontArray%(), RetCode%, _
        RetMsg$)

IF (RetCode% <> 0) THEN

  PRINT
  PRINT "***** ERROR: RetCode% = "; RetCode%
  PRINT "***** "; RetMsg$

ELSE

  '...display a sample of the font...
  CALL FastString("Sample of Font ", 2, 20, 80, FontArray%())

END IF

END
```

WidthString%

Function which returns the width in pixels of the given string in the specified font. This allows the calling program to check to insure the text fits the display before drawing.

Format:

```
DECLARE FUNCTION WidthString%(Text$, FontArray%())
```

Arguments:

```
      Text$ = text of which to compute width
 FontArray% = integer array containing font data
```

Example:

The following example uses **WidthString** to display text between two x-limits. The text is broken between words to fit the display area.

```
REM:  EX_WIDTH.BAS, Unregistered Version 1.0
REM:  Example of using WidthString to break a line.

DECLARE FUNCTION WidthString%(Text$, FontArray%())

DECLARE SUB BLOADFont(FlName$, FontArray%(), RetCode%)
DECLARE SUB FastString(Text$, FClr%, X%, Y%, FontArray%())

'...setup a VGA screen mode...
SCREEN 12

'...dimension array for font data (use REDIM so they're DYNAMIC)...
REDIM FontArray%(1)

'...load in a fonts and check the return code...
CALL BLOADFont("DTCH_BLD.BIN", FontArray%(), RetCode%)
IF (RetCode% <> 0) THEN STOP

'...define x-limits in which to display, y start, and text color...
XMin% = 100: XMax% = 300: YRow% = 100: FClr% = 7

Text$ = "This is a simple demonstration of breaking text between "
Text$ = Text$ + "words to fit on a line."

'...compute line width...
BxWidth% = XMax% - XMin% + 1

'...start at the first char and a min x-coord...
i% = 1: X% = XMin%

'...set row spacing - vertical spacing + any vertical padding...
DRow% = FontArray%(7) + FontArray%(10)

DO
```

```
    '...find the next word break (the next space)...
    j% = INSTR(i%, Text$, " ")

    '...if no space found before the end, skip to the end...
    IF (j% <= 0) THEN j% = LEN(Text$)

    '...pull the word...
    Word$ = MID$(Text$, i%, j% - i% + 1)

    '...compute its width...
    WordWidth% = WidthString%(Word$, FontArray%())

    '...will this word exceed the length of the box or end of text...
    IF (CurrLine% + WordWidth% > BxWidth%) THEN

      '...display the text for the line...
      CALL FastString(LineText$, FClr%, X%, YRow%, FontArray%())

      '...set to rwo start and add line spacing to next row...
      X% = XMin%: YRow% = YRow% + DRow%

      '...the word we just found starts the new line...
      LineText$ = Word$: CurrLine% = WordWidth%

    ELSE

      '...just add the text to the line and the width counter...
      LineText$ = LineText$ + Word$: CurrLine% = CurrLine% + WordWidth%

    END IF

    '...if we reached end of string, display any remaining text...
    IF (j% >= LEN(Text$)) THEN
      CALL FastString(LineText$, FClr%, X%, YRow%, FontArray%())
    END IF

    '...set i% to the start of the next word...
    i% = j% + 1

  LOOP UNTIL (i% > LEN(Text$))

  END
```

Appendix B
Registration Instructions

QBWinFont is shareware. If you continue to use this product after a trial period, you must register it.

Registered users will receive the most recent version of the **QBWinFont** package on disk, which includes all documentation in Windows Write (WRI) format. To keep the price lower for the basic registration, printed, unbound copies of the manual are sold separately for a small fee.

Any questions you may have about the **QBWinFont** package before ordering are welcomed at the address below or at the e-mail address: **joe@galcit.caltech.edu**.

Fill in the order form on the following page and send with a check or money order in US dollars for the total amount indicated to:

Nybble Software
P.O. Box 7213
Tuscaloosa, AL 35486

Foreign Orders: Please enclose only checks or money orders in US dollars drawn on a US bank. Checks drawn on banks outside the US will be returned, since they cost more to cash in bank fees than the registration amount.

QBWinFont Registration Form

NAME _____

STREET _____

CITY _____

STATE _____ **ZIP** _____

IF DESIRED: PHONE _____

E-MAIL _____

HOW DID YOU FIND OUR PROGRAM _____

[ ] QBWinFont Registration ..... ____8.00_
Includes most recent version on disk with complete documentation in Windows' Write (WRI) format. Please specify disk type:
[ ] 3 1/2"[ ] 3 1/2" 720K only
[ ] 5 1/4"[ ] 5 1/4" 360K only

[ ] Printed, unbound copy of documentation ..... Add $2.00
_____.00_

Foreign Orders, Add $2.00 Shipping & Handling _____.00_

Total Amount Enclosed US Dollars _____.00_

Mail Completed Form With Check or Money Order To: **Nybble SoftWare**
P.O Box 7213
Tuscaloosa, AL 35486