

Fastgraph/Fonts 6.0

for Windows[®]

User's Guide

Copyright © 1996-2003 by Ted Gruber Software, Inc.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without express written permission from Ted Gruber Software. The software described in this publication is furnished under a license agreement and may be used or copied only in accordance with the terms of that agreement.

This publication and its associated software are sold without warranties, either expressed or implied, regarding their merchantability or fitness for any particular application or purpose. The information in this publication is subject to change without notice and does not represent a commitment on the part of Ted Gruber Software. In no event shall Ted Gruber Software be liable for any loss of profit or any other commercial damage, including but not limited to special, incidental, consequential, or other damages resulting from the use of or the inability to use this product, even if Ted Gruber Software has been notified of the possibility of such damages.

Fastgraph/Fonts for Windows version 6.03

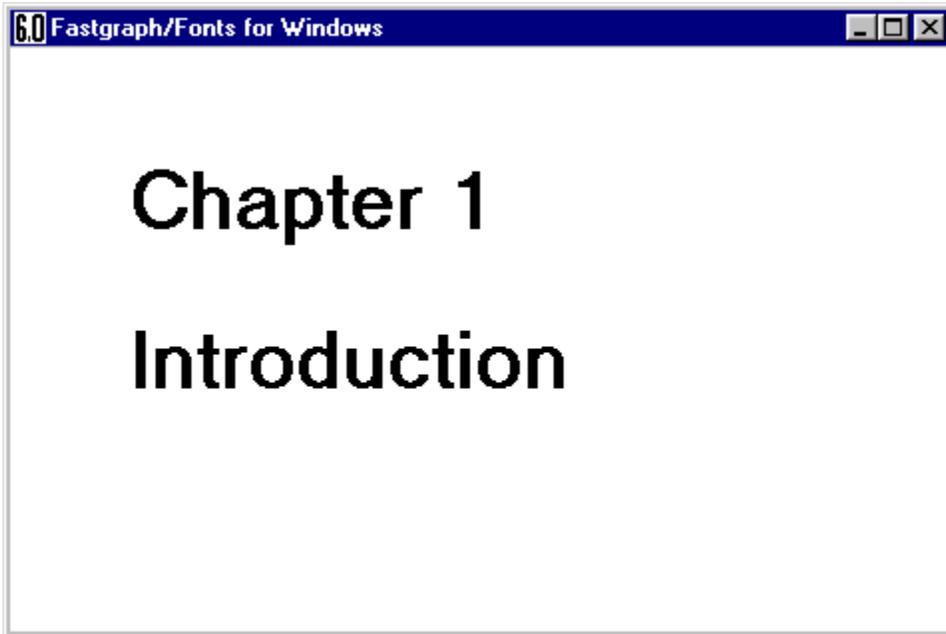
Fastgraph® is a registered trademark of Ted Gruber Software, Inc.

Windows®, Direct3D®, DirectDraw®, and DirectX® are registered trademarks of Microsoft Corporation.

All other brand and product names mentioned in this publication are trademarks or registered trademarks of their respective holders.

Table of Contents

CHAPTER 1 INTRODUCTION	1
ABOUT FASTGRAPH/FONTS.....	2
PREREQUISITE KNOWLEDGE	2
SUPPORTED COMPILERS	2
INSTALLING FASTGRAPH/FONTS	2
THE HELP FILE.....	3
FASTGRAPH/FONTS NAMING CONVENTIONS.....	3
TECHNICAL SUPPORT	3
MAINTENANCE UPDATES.....	4
CHAPTER 2 USING FASTGRAPH/FONTS	5
OVERVIEW	6
COMPILATION AND LINKING	6
HEADER FILES	7
96-CHARACTER AND 256-CHARACTER FONT FILES	7
FONT FILES DISTRIBUTED WITH FASTGRAPH/FONTS	7
LOADING AND UNLOADING FONTS	9
STRING DISPLAY	9
USING MORE THAN ONE FONT.....	23
MANUAL FONT LOADING	29
STRING MEASUREMENT	35
SHADOWED CHARACTERS	47
SUMMARY	52
CHAPTER 3 CREATING AND EDITING FONT FILES	55
OVERVIEW	56
96-CHARACTER FONT FILE STRUCTURE	56
256-CHARACTER FONT FILE STRUCTURE	57
THE FGFEDIT UTILITY	57
THE FGF2FGX UTILITY.....	58
THE SCALFONT UTILITY	58
INDEX	61



Chapter 1

Introduction

About Fastgraph/Fonts

Fastgraph/Fonts lets you easily add bitmapped font support to Fastgraph applications. It includes a function library, font files, and utilities to help create your own custom font files. Fastgraph/Fonts is an add-on product that requires Fastgraph.

Chapter 1 of the *Fastgraph/Fonts 6.0 User's Guide* contains installation instructions and other introductory information. Chapter 2 describes how to compile and link a program that uses Fastgraph/Fonts and takes you on a guided tour of the product's most important features. Chapter 3 discusses the font file structure and how to create and edit font files.

The examples in this manual are presented in C/C++, C++Builder, Delphi, and Visual Basic. The Fastgraph/Fonts SETUP program can install versions of the examples for any of these languages, in addition to MFC, C#, VB.NET, and PowerBASIC. Finally, we'd like to point out that the examples should be read not by themselves, but as part of the surrounding text.

Prerequisite Knowledge

Fastgraph/Fonts is a programming tool, which means programmers are its intended audience. For this reason, we assume you have a knowledge of programming. Further, as Fastgraph/Fonts is an add-on product used with Fastgraph, we also assume you are already familiar with Fastgraph.

Supported Compilers

You can use Fastgraph/Fonts with any Win32 compiler that Fastgraph supports:

- Borland C++ (version 5.0 or later)
- Borland C++Builder (version 1.0 or later)
- Borland Delphi (version 2.0 or later)
- Microsoft Visual Basic (version 4.0 or later)
- Microsoft Visual Basic .NET (version 2002 or later)
- Microsoft Visual C++ (version 2.2 or later)
- Microsoft Visual C# .NET (version 2002 or later)
- PowerBASIC PB/DLL (version 6.0 or later)
- Symantec C++ (version 7.0 or later)
- Watcom C/C++ (version 10.6 or later)

The listed version numbers are the compiler versions under which Fastgraph/Fonts and its example programs have been tested. Fastgraph/Fonts may or may not work with earlier versions of these compilers.

If you need to create Win16 programs for Windows 3.x, the Fastgraph/Fonts 6.0 disk includes the 16-bit Fastgraph/Fonts 5.02 distribution in the Win16 directory. Fastgraph/Fonts 5.02 is the last version that provides Win16 support.

Installing Fastgraph/Fonts

This section explains how to use the SETUP program to load Fastgraph/Fonts and its related files on a hard disk. SETUP lets you select the compilers you wish to use and also gives you the opportunity to load example programs specific to those compilers.

Place the distribution disk in any available diskette drive, select Run from the Windows Start menu, type A:\SETUP, and press Enter (replace "A" with the name of your diskette drive if necessary). Once SETUP begins, just follow the directions on each screen.

The SETUP program will ask you for the compilers you'll use with Fastgraph/Fonts, as well as the directory names for utilities, libraries, and include files. For the utilities, the default directory is C:\FGFW60. For the include files and libraries, we recommend using directories where the compiler you've chosen normally searches for such files. SETUP will automatically try to determine these directories and propose them as defaults.

You can install support for additional compilers at any time. If you choose to do this, you should use the command SETUP /L to avoid copying the files common to all environments.

The Help File

Fastgraph/Fonts does not have its own help file. Rather, the Fastgraph help file (FGW60.hlp) contains the information presented in the *Fastgraph/Fonts 6.0 User's Guide* and the *Fastgraph/Fonts 6.0 Reference Manual*. The reference manual provides summaries of each Fastgraph/Fonts function, with function prototypes or declarations for each supported language, information about the function parameters and return values, restrictions, references to similar functions, and function cross-references to the example programs supplied with Fastgraph/Fonts.

Fastgraph/Fonts Naming Conventions

The names of all Fastgraph/Fonts functions begin with the four characters "fgf_". This prefix helps identify Fastgraph/Fonts functions within a program, and it also reduces the chance of name conflicts that might otherwise occur between Fastgraph/Fonts and other third party libraries.

Fastgraph/Fonts uses a slightly different naming convention for C#. The Fastgraph/Fonts C# function names do not include the leading **fgf_** prefix and are accessed through a class named **fgf**. This means, for example, a C# program would call **fgf.load()** instead of **fgf_load()**. For simplicity, function names presented in this manual will use the **fgf_** convention.

Technical Support

TGS provides a variety of convenient technical support channels for all Fastgraph products:

- **Telephone support** is available weekdays (excluding holidays) between 9:00 a.m. and 5:00 p.m. Pacific time. Call (702) 735-1980.
- **Fax support** is available 24 hours a day. Our fax number is (702) 735-4603.
- **Email support** is available 24 hours a day. Our technical support email address is **support@fastgraph.com**.
- The **Fastgraph web page** <http://www.fastgraph.com> contains product information, answers to common questions, previews to coming attractions, and more. The web page also provides a convenient way to download maintenance updates and additional examples from our ftp site.
- Our [Web BBS](#) contains discussion forums where you can talk to other Fastgraph users.
- Our **Internet ftp** site **ftp.fastgraph.com** provides a convenient way to receive maintenance updates, examples, and other relevant files. Look in the **/fg** and **/fg/Windows** directories once you reach the ftp site.

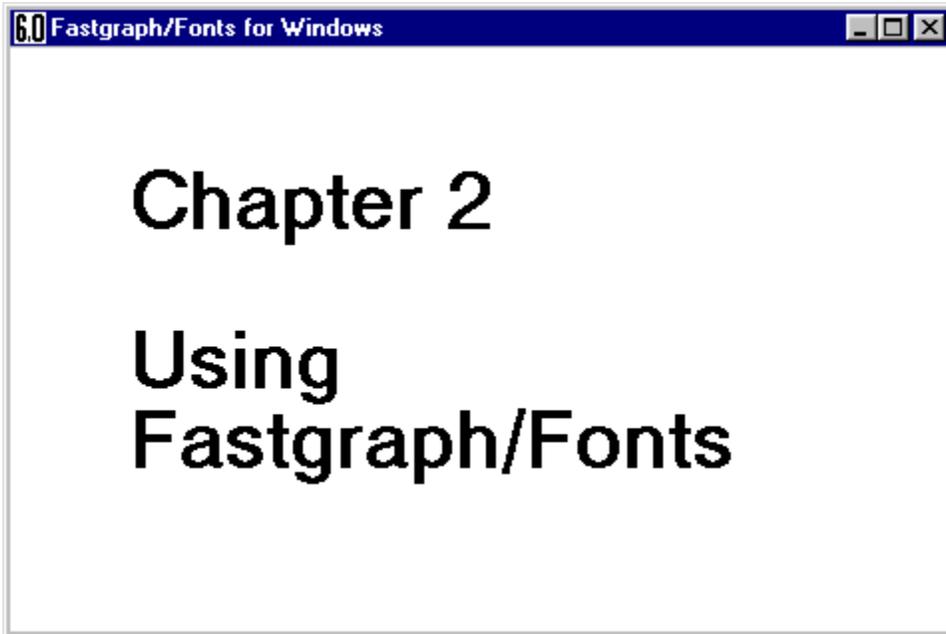
Should you encounter a Fastgraph/Fonts problem, please do the following before contacting TGS for technical support:

- If possible, try running the program on another system with a different display driver. If the problem only occurs on one system, it is most likely a problem with your system's Windows display driver. Contact your video card manufacturer to make sure you have the latest version of their Windows display drivers (and their latest DirectX drivers if it is a DirectX program).
- Isolate the problem down to the smallest possible program that reproduces the error. Often you'll find the source of the problem during this process.

Please note that TGS will be glad to provide Fastgraph/Fonts technical support, but we *cannot provide general Windows programming help*. For Windows programming help, please see the Windows Programming References section in the *Fastgraph 6.0 User's Guide*.

Maintenance Updates

We periodically issue maintenance updates to support new compilers, enhance functionality, and fix known problems. Maintenance updates for all Fastgraph products are available as patches at no charge from our web page and internet ftp site.



Overview

We'll begin this chapter with a discussion of how to compile and link a Fastgraph/Fonts program. Next we'll take a guided tour of the most important Fastgraph/Fonts functions, starting with the basic operations of loading fonts and simple character display and then proceeding to other topics. More detailed information about all Fastgraph/Fonts functions may be found in the *Fastgraph/Fonts 6.0 Reference Manual*.

Compilation and Linking

Fastgraph/Fonts programs are compiled and linked using the same techniques described in Chapter 2 of the *Fastgraph 6.0 User's Guide*. The only difference is that you must specify the appropriate Fastgraph/Fonts library, unit, or module name in addition to those required by Fastgraph. The rest of this section describes how to do this for each supported programming language.

For C and C++ programs, use the following table to determine the appropriate Fastgraph/Fonts library that corresponds to each Fastgraph library. Then specify *both* library names in your project file or on the compile/link command line. Note that Fastgraph/Fonts does not include special DirectX libraries because the same Fastgraph/Fonts libraries work with Fastgraph's native and DirectX libraries.

If linking with this Fastgraph library:	Add this library:
FGWBC32.LIB or FGWBC32D.LIB	FGFWBC32.LIB
FGWSC32.LIB or FGWSC32D.LIB	FGFWSC32.LIB
FGWVC32.LIB or FGWVC32D.LIB	FGFWVC32.LIB
FGWWC32.LIB or FGWWC32D.LIB	FGFWWC32.LIB

For example, if you're using Fastgraph/Fonts with C++Builder, you would link with FGWBC32.LIB *plus* FGFWBC32.LIB.

For Delphi programs, add the unit file FGFWin to your program's `uses` statement (FGFWinD if creating a DirectX program). The `uses` statement must include both the Fastgraph and Fastgraph/Fonts unit names.

For Visual Basic programs, add the module file FGFWin.bas to your project (FGFWinD.bas if creating a DirectX program). The project must include both the Fastgraph and Fastgraph/Fonts modules. You may *not* distribute the Fastgraph/Fonts Visual Basic module (.bas) files.

For Visual Basic .NET programs, add a link to the module file FGFWin.vb to your solution (FGFWinD.vb if creating a DirectX program). The solution must include links to both the Fastgraph and Fastgraph/Fonts modules. By default, the Fastgraph/Fonts SETUP program copies FGFWin.vb and FGFWinD.vb to \FGFW60 and \FGFW60\Examples\VBNET\Common. You may *not* distribute the Fastgraph/Fonts Visual Basic .NET module files (FGFWin.vb and FGFWinD.vb).

For C# programs, add a link to the module file FGFWin.cs to your solution (FGFWinD.cs if creating a DirectX program). The solution must include links to both the Fastgraph and Fastgraph/Fonts modules. By default, the Fastgraph/Fonts SETUP program copies FGFWin.cs and FGFWinD.cs to \FGFW60 and \FGFW60\Examples\VBNET\Common. You may *not* distribute the Fastgraph/Fonts C# module files (FGFWin.cs and FGFWinD.cs).

For PowerBASIC programs, add an `#INCLUDE` metastatement specifying the FGFWin.inc include file (FGFWinD.inc if creating a DirectX program). You may *not* distribute the Fastgraph/Fonts PowerBASIC include (.inc) files.

Header Files

For C and C++ programs, the FGFWIN.H header file contains the Fastgraph/Fonts function prototypes. The statements

```
#include <fgwin.h>
#include <fgfwin.h>
```

must appear in all C or C++ Fastgraph/Fonts programs, and both header files must exist in a directory where the compiler normally searches for header files. Actually, FGFWIN.H includes FGWIN.H if necessary, but we recommend explicitly including both header files for clarity.

96-Character and 256-Character Font Files

Fastgraph/Fonts supports two distinct font file formats, but you use the exact same Fastgraph/Fonts functions with either format. The original font file format contains the 96 standard ASCII printable characters (ASCII values 32 to 127 decimal). 256-character font files contain all standard and extended ASCII characters (values 0 to 255) but of course require more memory than 96-character fonts. By convention, both formats use the FGF file type. We'll describe both the 96-character and 256-character font file formats in detail in Chapter 3.

Font Files Distributed With Fastgraph/Fonts

The next page illustrates sample characters from each font distributed with Fastgraph/Fonts. The Fastgraph/Fonts SETUP program installs these font files in the Fonts subdirectory of the Fastgraph/Fonts utilities directory. The file FONTLIST (in the same directory as the font files) lists the file names for each of these fonts. Note that these are the same font files distributed with Fastgraph/Fonts for DOS.

Austin 36

Birch 24

Birch 35

Block 5
Broadway 18

CARGO 22

CASINO 19

Classic 8

Classic 12

Cognac 10

Cognac 19

Crystal 14

Crystal 26

Crystal 40

Fountain 27

Garden 27

Gothic 24

Kids 30

Modern 28

Plaza 14

Plaza 26

Regal 18

Regal 24

Royal 15

Scotia 12

Scotia 24

Serif 12

Serif 24

Simple 9

Simple 12

Simple 18

Simple 23

Slant 26

Small 11

Sterling 13

Standard 8

Traditional 8

Traditional 13

Wire 12

0123ABCDabcd+~*/#&

0123ABCDabcd+~*/#&

0123ABCDabcd+~*/#&

0123ABCDabcd+~*/#&
0123ABCDabcd+~*/#&

0123ABCDABCD+~*/#&

0123ABCDABCD+~*/#&

0123ABCDabcd+~*/#&

0123ABCDabcd+~*/#&

0123ABCDabcd+~*/#&

0123ABCDabcd+~*/#&

0123ABCDabcd+~*/#&

0123ABCDabcd+~*/#&

0123ABCDabcd+~*/#&

0123ABCDabcd+~*/#&

0123ABCDabcd+~*/#&

0123ABCDabcd+~*/#&

0123ABCDabcd+~*/#&

0123ABCDabcd+~*/#&

0123ABCDabcd+~*/#&

0123ABCDabcd+~*/#&

0123ABCDabcd+~*/#&

0123ABCDabcd+~*/#&

0123ABCDabcd+~*/#&

0123ABCDabcd+~*/#&

0123ABCDabcd+~*/#&

0123ABCDabcd+~*/#&

0123ABCDabcd+~*/#&

0123ABCDabcd+~*/#&

0123ABCDabcd+~*/#&

0123ABCDabcd+~*/#&

0123ABCDabcd+~*/#&

0123ABCDabcd+~*/#&

0123ABCDabcd+~*/#&

0123ABCDabcd+~*/#&

0123ABCDabcd+~*/#&

0123ABCDabcd+~*/#&

0123ABCDabcd+~*/#&

0123ABCDabcd+~*/#&

Loading and Unloading Fonts

The character definitions that make up each font are stored in external files called *font files*. The font files distributed with Fastgraph/Fonts all have names of the form *xxxxxnn.FGF*, where *xxxxxx* is a one- to six-character name describing the font's appearance, and *nn* is a two-digit value that specifies the font's *point size*. The point size is the height in pixels from the top of a normal upper case character to the bottom of the lowest descender character in the font. While the font file naming convention provides consistency, font files may have any valid file name.

To display characters with Fastgraph/Fonts, you must first load the contents of a font file into memory. There are two ways to do this -- *automatic loading* and *manual loading*. Automatic loading uses the **fgf_load()** function to allocate memory for the font and read the font file contents into the allocated memory. Manual loading uses the **fgf_define()** function, which expects your program to perform its own memory management and font file reading. While automatic loading relieves you of these details, manual loading may be preferable if you want to include the font data directly in your program. Most examples we'll present will use automatic loading. In either case, you can have up to 256 font files loaded at the same time.

Both **fgf_load()** and **fgf_define()** expect a single parameter. With **fgf_load()**, the parameter is a null-terminated character string that specifies the font file name. With **fgf_define()**, the parameter is the name of an array into which your program has read the contents of a font file; when using the .NET framework, this array must be a pinned object (this restriction may be lifted in a future version of Fastgraph/Fonts). If successful, each function returns a *font handle* between 1 and 256 that you use to reference the font in later operations. A return value of zero from **fgf_load()** indicates that the font was not loaded because there is not enough memory available for loading the font, the font file wasn't found, or there are no more font handles available. A return value of zero from **fgf_define()** can only occur if there are no more font handles available. When you successfully load a font, that font becomes the *current font*.

If your program uses automatic loading, it should unload all fonts (that is, release the memory allocated for the fonts) before it exits. You also may wish to unload a specific font when you're through using that font. The **fgf_unload()** function is provided for this purpose. It expects a single integer parameter whose value is either a font handle or a negative value. If it is a font handle, only that font is unloaded. If it is negative, all fonts are unloaded.

String Display

After you load a font, use **fgf_print()** or **fgf_printc()** to display characters from that font. The difference between these two functions is that **fgf_printc()** does not display characters or portions of characters outside the clipping region defined by **fg_setclip()**, while **fgf_print()** ignores the clipping limits and is thus faster. Both **fgf_print()** and **fgf_printc()** display a string from the current font in the active virtual buffer (Fastgraph/Fonts always directs output to a virtual buffer, regardless of the **fg_fontdc()** setting). The first parameter passed to each function is the character string to display, while the second parameter is the number of characters to display from that string. Both functions leave the graphics cursor positioned at the bottom right corner of the last character displayed.

By default, **fgf_print()** and **fgf_printc()** display strings so their lower left corner is at the graphics cursor position. The **fgf_justify()** function lets you change this default justification. Its two parameters, *xjust* and *yjust*, control the string positioning about the current graphics position, as shown here.

value of xjust	value of yjust	horizontal justification	vertical justification
-1	-1	left	lower
-1	0	left	center
-1	1	left	upper
0	-1	center	lower
0	0	center	center
0	1	center	upper
1	-1	right	lower
1	0	right	center
1	1	right	upper

In the context of vertical justification, *lower justification* means the bottom of each character (excluding descenders such as "j" and "y") will be at the current graphics y position. *Upper justification* means the top of the tallest character displayed will be at the graphics y position, while *center justification* means the center of the tallest character will be at the graphics y position. Notice the default justification settings are *xjust* = -1 and *yjust* = -1.

Example FGFW1 illustrates the use of **fgf_load()**, **fgf_print()**, and **fgf_justify()**. It uses **fgf_load()** to load a font; it exits if the load operation fails. The first three sets of calls to **fgf_justify()** and **fgf_print()** display the string "FG/Fonts" left justified, centered, and right justified at the top of the virtual buffer. The next three sets of calls display the string in these positions centered vertically, and the last three display it in these positions at the bottom of the virtual buffer. Note how the example uses Fastgraph's **fg_move()** and **fg_setcolor()** functions to control the position and color of the string.

In Visual Basic, prefixing the font file name string with the `App.Path` property makes **fgf_load()** look for the file in the directory where the application's project (VBP) file resides when running the application from Visual Basic's development environment, or from the directory where the EXE file resides when running the application as an executable. If we don't do this, Visual Basic programs will look in the Visual Basic directory when running from the development environment. The PowerBASIC versions of the Fastgraph/Fonts examples include an **AppPath()** function that behaves the same way.

FGFW1: C/C++ Version

```

/*****\
*
* FGFW1.c
*
* Demonstrate the basic Fastgraph/Fonts for Windows operations of font
* loading, character display and justification, and font unloading.
*
\*****/

#include <fgwin.h>
#include <fgfwin.h>

LRESULT CALLBACK WindowProc(HWND,UINT,WPARAM,LPARAM);

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                   PSTR szCmdParam, int iCmdShow)
{
    static char szAppName[] = "FGFW1";
    HWND      hWnd;
    MSG       msg;
    WNDCLASSEX wndclass;

    wndclass.cbSize      = sizeof(wndclass);
    wndclass.style       = CS_HREDRAW | CS_VREDRAW | CS_OWNDC;
    wndclass.lpfnWndProc = WindowProc;
    wndclass.cbClsExtra  = 0;
    wndclass.cbWndExtra  = 0;
    wndclass.hInstance  = hInstance;
    wndclass.hIcon       = LoadIcon(NULL, IDI_APPLICATION);
    wndclass.hCursor     = LoadCursor(NULL, IDC_ARROW);

```

```

wndclass.hbrBackground = NULL;
wndclass.lpszMenuName = NULL;
wndclass.lpszClassName = szAppName;
wndclass.hIconSm = LoadIcon(NULL, IDI_APPLICATION);
RegisterClassEx(&wndclass);

hWnd = CreateWindow(
    szAppName,           // window class name
    "Example FGFW1",    // window caption
    WS_OVERLAPPEDWINDOW, // window style
    CW_USEDEFAULT,      // initial x position
    CW_USEDEFAULT,      // initial y position
    CW_USEDEFAULT,      // initial x size
    CW_USEDEFAULT,      // initial y size
    NULL,               // parent window handle
    NULL,               // window menu handle
    hInstance,          // program instance handle
    NULL);              // creation parameters

ShowWindow(hWnd, iCmdShow);
UpdateWindow(hWnd);

while (GetMessage(&msg, NULL, 0, 0))
{
    TranslateMessage(&msg);
    DispatchMessage(&msg);
}
return msg.wParam;
}

/*****\
*
* WindowProc()
*
*****/

HDC      hDC;
HPALETTE hPal;
int      hVB;
UINT     cxClient, cyClient;

LRESULT CALLBACK WindowProc(HWND hWnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
    PAINTSTRUCT ps;
    int xMax, yMax, xMid, yMid;

    switch (iMsg)
    {
        case WM_CREATE:
            hDC = GetDC(hWnd);
            fg_setdc(hDC);
            hPal = fg_defpal();
            fg_realize(hPal);

            fg_vbinit();
            hVB = fg_vballoc(640, 480);
            fg_vbopen(hVB);
            fg_vbcolors();

            if (fgf_load("AUSTIN36.FGF") == 0)
            {
                MessageBox(hWnd, "Unable to load font file.", "Error",
                    MB_ICONSTOP|MB_OK);
                DestroyWindow(hWnd);
                return 0;
            }

            xMax = fg_getmaxx();
            yMax = fg_getmaxy();
            xMid = xMax / 2;
            yMid = yMax / 2;

            fg_setcolor(255);
            fg_fillpage();

            fg_setcolor(19);

```

```

    fg_move(0,0);
    fgf_justify(-1,1);
    fgf_print("FG/Fonts",8);
    fg_move(xMid,0);
    fgf_justify(0,1);
    fgf_print("FG/Fonts",8);
    fg_move(xMax,0);
    fgf_justify(1,1);
    fgf_print("FG/Fonts",8);

    fg_move(0,yMid);
    fgf_justify(-1,0);
    fgf_print("FG/Fonts",8);
    fg_move(xMid,yMid);
    fgf_justify(0,0);
    fgf_print("FG/Fonts",8);
    fg_move(xMax,yMid);
    fgf_justify(1,0);
    fgf_print("FG/Fonts",8);

    fg_move(0,yMax);
    fgf_justify(-1,-1);
    fgf_print("FG/Fonts",8);
    fg_move(xMid,yMax);
    fgf_justify(0,-1);
    fgf_print("FG/Fonts",8);
    fg_move(xMax,yMax);
    fgf_justify(1,-1);
    fgf_print("FG/Fonts",8);
    return 0;

case WM_PAINT:
    BeginPaint(hWnd,&ps);
    fg_vbscale(0,fg_getmaxx(),0,fg_getmaxy(),0,cxClient-1,0,cyClient-1);
    EndPaint(hWnd,&ps);
    return 0;

case WM_SETFOCUS:
    fg_realize(hPal);
    InvalidateRect(hWnd,NULL,TRUE);
    return 0;

case WM_SIZE:
    cxClient = LOWORD(lParam);
    cyClient = HIWORD(lParam);
    return 0;

case WM_DESTROY:
    fgf_unload(-1);
    fg_vbclose();
    fg_vbfree(hVB);
    fg_vbfin();
    DeleteObject(hPal);
    ReleaseDC(hWnd,hDC);
    PostQuitMessage(0);
    return 0;
}
return DefWindowProc(hWnd,iMsg,wParam,lParam);
}

```

FGFW1: C++Builder Version

```

/*****\
*
*   FGFW1.cpp
*   FGFW1U.cpp
*
*   Demonstrate the basic Fastgraph/Fonts for Windows operations of font
*   loading, character display and justification, and font unloading.
*
*****/

#include <vcl\vcl.h>
#pragma hdrstop

#include "FGFW1U.h"

```

```

//-----
#pragma resource "*.dfm"
TForm1 *Form1;
//-----
__fastcall TForm1::TForm1(TComponent* Owner)
: TForm(Owner)
{
}
//-----
void __fastcall TForm1::FormActivate(TObject *Sender)
{
    fg_realize(hPal);
    Invalidate();
}
//-----
void __fastcall TForm1::FormCreate(TObject *Sender)
{
    int xMax, yMax, xMid, yMid;

    hDC = GetDC(Form1->Handle);
    fg_setdc(hDC);
    hPal = fg_defpal();
    fg_realize(hPal);

    fg_vbinit();
    hVB = fg_vballoc(640,480);
    fg_vbopen(hVB);
    fg_vbcolors();

    if (fgf_load("AUSTIN36.FGF") == 0)
    {
        MessageDlg("Unable to load font file.",mtError,TMsgDlgButtons()<<mbOK,0);
        Close();
        return;
    }

    xMax = fg_getmaxx();
    yMax = fg_getmaxy();
    xMid = xMax / 2;
    yMid = yMax / 2;

    fg_setcolor(255);
    fg_fillpage();

    fg_setcolor(19);
    fg_move(0,0);
    fgf_justify(-1,1);
    fgf_print("FG/Fonts",8);
    fg_move(xMid,0);
    fgf_justify(0,1);
    fgf_print("FG/Fonts",8);
    fg_move(xMax,0);
    fgf_justify(1,1);
    fgf_print("FG/Fonts",8);

    fg_move(0,yMid);
    fgf_justify(-1,0);
    fgf_print("FG/Fonts",8);
    fg_move(xMid,yMid);
    fgf_justify(0,0);
    fgf_print("FG/Fonts",8);
    fg_move(xMax,yMid);
    fgf_justify(1,0);
    fgf_print("FG/Fonts",8);

    fg_move(0,yMax);
    fgf_justify(-1,-1);
    fgf_print("FG/Fonts",8);
    fg_move(xMid,yMax);
    fgf_justify(0,-1);
    fgf_print("FG/Fonts",8);
    fg_move(xMax,yMax);
    fgf_justify(1,-1);
    fgf_print("FG/Fonts",8);

    Application->OnActivate = OnActivate;
}

```

```

}
//-----
void __fastcall TForm1::FormPaint(TObject *Sender)
{
    fg_vbscale(0,fg_getmaxx(),0,fg_getmaxy(),0,cxClient-1,0,cyClient-1);
}
//-----
void __fastcall TForm1::FormResize(TObject *Sender)
{
    cxClient = ClientWidth;
    cyClient = ClientHeight;
    Invalidate();
}
//-----
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
    fgf_unload(-1);
    fg_vbclose();
    fg_vbfree(hVB);
    fg_vbfin();
    DeleteObject(hPal);
    ReleaseDC(Form1->Handle,hDC);
}

```

FGFW1: Delphi Version

```

{*****}
*
*   FGFw1.dpr
*   FGFw1U.pas
*
*   Demonstrate the basic Fastgraph/Fonts for Windows operations of font
*   loading, character display and justification, and font unloading.
*
*****}

unit FgfW1U;

interface

uses
    SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
    Forms, Dialogs, FGWin, FGFWin;

type
    TForm1 = class(TForm)
    procedure AppOnActivate(Sender: TObject);
    procedure FormActivate(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormPaint(Sender: TObject);
    procedure FormResize(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    end;

var
    Form1: TForm1;

implementation

{$R *.DFM}
var
    dc      : hDC;
    hPal    : hPalette;
    hVB     : integer;
    cxClient, cyClient : integer;

procedure TForm1.AppOnActivate(Sender: TObject);
begin
    fg_realize(hPal);
    Invalidate;
end;

procedure TForm1.FormActivate(Sender: TObject);
begin
    fg_realize(hPal);
end;

```

```

procedure TForm1.FormCreate(Sender: TObject);
var
  xMax, yMax, xMid, yMid : integer;
begin
  dc := GetDC(Form1.Handle);
  fg_setdc(dc);
  hPal := fg_defpal;
  fg_realize(hPal);

  fg_vbinit;
  hVB := fg_vballoc(640,480);
  fg_vbopen(hVB);
  fg_vbcolors;

  if fgf_load('AUSTIN36.FGF'+chr(0)) = 0 then
  begin
    MessageDlg('Unable to load font file.', mtError, [mbOK], 0);
    Close;
    Exit;
  end;

  xMax := fg_getmaxx;
  yMax := fg_getmaxy;
  xMid := xMax div 2;
  yMid := yMax div 2;

  fg_setcolor(255);
  fg_fillpage;

  fg_setcolor(19);
  fg_move(0,0);
  fgf_justify(-1,1);
  fgf_print('FG/Fonts',8);
  fg_move(xMid,0);
  fgf_justify(0,1);
  fgf_print('FG/Fonts',8);
  fg_move(xMax,0);
  fgf_justify(1,1);
  fgf_print('FG/Fonts',8);

  fg_move(0,yMid);
  fgf_justify(-1,0);
  fgf_print('FG/Fonts',8);
  fg_move(xMid,yMid);
  fgf_justify(0,0);
  fgf_print('FG/Fonts',8);
  fg_move(xMax,yMid);
  fgf_justify(1,0);
  fgf_print('FG/Fonts',8);

  fg_move(0,yMax);
  fgf_justify(-1,-1);
  fgf_print('FG/Fonts',8);
  fg_move(xMid,yMax);
  fgf_justify(0,-1);
  fgf_print('FG/Fonts',8);
  fg_move(xMax,yMax);
  fgf_justify(1,-1);
  fgf_print('FG/Fonts',8);

  Application.OnActivate := AppOnActivate;
end;

procedure TForm1.FormPaint(Sender: TObject);
begin
  fg_vbscale(0,fg_getmaxx,0,fg_getmaxy,0,cxClient-1,0,cyClient-1);
end;

procedure TForm1.FormResize(Sender: TObject);
begin
  cxClient := ClientWidth;
  cyClient := ClientHeight;
  Invalidate;
end;

```

```
procedure TForm1.FormDestroy(Sender: TObject);
begin
  fgf_unload(-1);
  fg_vbclose;
  fg_vbfree(hVB);
  fg_vbfin;
  DeleteObject(hPal);
  ReleaseDC(Form1.Handle,dc);
end;

end.
```

FGFW1: Visual Basic Version

```
*****
'
' FGFW1.frm
'
' Demonstrate the basic Fastgraph/Fonts for Windows operations of font
' loading, character display and justification, and font unloading.
'
*****

Dim hPal As Long
Dim hVB As Long
Dim cxClient As Long, cyClient As Long

Private Sub Form_Activate()
  Call fg_realize(hPal)
  Refresh
End Sub

Private Sub Form_Load()
  Dim xMax As Long, yMax As Long
  Dim xMid As Long, yMid As Long

  ScaleMode = 3
  Call fg_setdc(hDC)
  hPal = fg_defpal()
  Call fg_realize(hPal)

  Call fg_vbinit
  hVB = fg_vballoc(640, 480)
  Call fg_vbopen(hVB)
  Call fg_vbcolors

  If fgf_load(App.Path & "\AUSTIN36.FGF") = 0 Then
    Call MsgBox("Unable to load font file.", vbCritical, "Error")
    Unload Me
    Exit Sub
  End If

  xMax = fg_getmaxx()
  yMax = fg_getmaxy()
  xMid = xMax / 2
  yMid = yMax / 2

  Call fg_setcolor(255)
  Call fg_fillpage

  Call fg_setcolor(19)
  Call fg_move(0, 0)
  Call fgf_justify(-1, 1)
  Call fgf_print("FG/Fonts", 8)
  Call fg_move(xMid, 0)
  Call fgf_justify(0, 1)
  Call fgf_print("FG/Fonts", 8)
  Call fg_move(xMax, 0)
  Call fgf_justify(1, 1)
  Call fgf_print("FG/Fonts", 8)

  Call fg_move(0, yMid)
  Call fgf_justify(-1, 0)
  Call fgf_print("FG/Fonts", 8)
  Call fg_move(xMid, yMid)
```

```

Call fgf_justify(0, 0)
Call fgf_print("FG/Fonts", 8)
Call fg_move(xMax, yMid)
Call fgf_justify(1, 0)
Call fgf_print("FG/Fonts", 8)

Call fg_move(0, yMax)
Call fgf_justify(-1, -1)
Call fgf_print("FG/Fonts", 8)
Call fg_move(xMid, yMax)
Call fgf_justify(0, -1)
Call fgf_print("FG/Fonts", 8)
Call fg_move(xMax, yMax)
Call fgf_justify(1, -1)
Call fgf_print("FG/Fonts", 8)
End Sub

Private Sub Form_Paint()
Call fg_vbyscale(0, fg_getmaxx(), 0, fg_getmaxy(), 0, cxClient - 1, 0, cyClient - 1)
End Sub

Private Sub Form_Resize()
cxClient = ScaleWidth
cyClient = ScaleHeight
Refresh
End Sub

Private Sub Form_Unload(Cancel As Integer)
Call fgf_unload(-1)
Call fg_vbclose
Call fg_vbfree(hVB)
Call fg_vbfin
End Sub

```

Normally, **fgf_print()** and **fgf_printc()** display a character string in the current color (that is, the color specified in the most recent call to the **fg_setcolor()** function). If you want to display part of a string in one color and part in another color, insert the *option prefix* character (ASCII decimal 127, hex 7F) in the string. When **fgf_print()** or **fgf_printc()** find this character, they treat the byte that follows as a color value and make that color the current color (for direct color virtual buffers, the color value serves as an index into the virtual palette). This means that subsequent characters appear in the new color, as will all graphics output that relies on the current color value.

The option prefix character is also used for underlining characters displayed with **fgf_print()** and **fgf_printc()**. If an underscore character (ASCII decimal 95, hex 5F) follows the option prefix character, characters that follow are underlined. The underlining continues until another option prefix/underscore combination is specified. The underlining is done in the color in effect at the point of the "underline off" sequence and is drawn two pixels beneath the character base (that is, the bottom of non-descending characters). You can change the vertical position of the underlining with **fgf_under()**. As you might expect, **fgf_printc()** performs underlining only within the clipping region, while **fgf_print()** underlines all requested characters regardless of their location.

The horizontal space between characters, called *kerning*, is included on the right side of the actual font bitmaps. The **fgf_kerning()** function provides additional kerning controls. It expects an integer parameter specifying the number of additional pixels between characters displayed with **fgf_print()** and **fgf_printc()**. By default, there is no additional space inserted between characters -- this is equivalent to calling **fgf_kerning(0)**. Note that the value passed to **fgf_kerning()** can be negative to reduce the space between characters. Kerning values set with **fgf_kerning()** remain in effect across font changes.

Example FGFW2 illustrates the use of color selection, underlining, and kerning with **fgf_print()**. It displays a string in two colors (19 and 20) in the upper left corner of the active virtual buffer. The color change resulting from the first **fgf_print()** call carries over to the second call (that is, the string "still color 20" also appears in color 20, not color 19). Following this, the example displays the string "one word is underlined", with the word "word" underlined and characters one additional pixel apart.

FGFW2: C/C++ Version

```

/*****\
*
*   FGFW2.c
*
*   Demonstrate color selection, underlining, and kerning with fgf_print().
*
*****/

#include <fgwin.h>
#include <fgfwin.h>

LRESULT CALLBACK WindowProc(HWND,UINT,WPARAM,LPARAM);

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                   PSTR szCmdParam, int iCmdShow)
{
    static char szAppName[] = "FGFW2";
    HWND       hWnd;
    MSG        msg;
    WNDCLASSEX wndclass;

    wndclass.cbSize      = sizeof(wndclass);
    wndclass.style       = CS_HREDRAW | CS_VREDRAW | CS_OWNDC;
    wndclass.lpfnWndProc = WindowProc;
    wndclass.cbClsExtra  = 0;
    wndclass.cbWndExtra  = 0;
    wndclass.hInstance  = hInstance;
    wndclass.hIcon       = LoadIcon(NULL,IDI_APPLICATION);
    wndclass.hCursor     = LoadCursor(NULL, IDC_ARROW);
    wndclass.hbrBackground = NULL;
    wndclass.lpszMenuName = NULL;
    wndclass.lpszClassName = szAppName;
    wndclass.hIconSm     = LoadIcon(NULL,IDI_APPLICATION);
    RegisterClassEx(&wndclass);

    hWnd = CreateWindow(
        szAppName,           // window class name
        "Example FGFW2",    // window caption
        WS_OVERLAPPEDWINDOW, // window style
        CW_USEDEFAULT,      // initial x position
        CW_USEDEFAULT,      // initial y position
        CW_USEDEFAULT,      // initial x size
        CW_USEDEFAULT,      // initial y size
        NULL,               // parent window handle
        NULL,               // window menu handle
        hInstance,         // program instance handle
        NULL);              // creation parameters

    ShowWindow(hWnd,iCmdShow);
    UpdateWindow(hWnd);

    while (GetMessage(&msg,NULL,0,0))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
    return msg.wParam;
}

/*****\
*
*   WindowProc()
*
*****/

HDC      hDC;
HPALETTE hPal;
int      hVB;
UINT     cxClient, cyClient;

LRESULT CALLBACK WindowProc(HWND hWnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
    PAINTSTRUCT ps;

```

```

switch (iMsg)
{
    case WM_CREATE:
        hDC = GetDC(hWnd);
        fg_setdc(hDC);
        hPal = fg_defpal();
        fg_realize(hPal);

        fg_vbinit();
        hVB = fg_vballoc(640,480);
        fg_vbopen(hVB);
        fg_vbcolors();

        if (fgf_load("AUSTIN36.FGF") == 0)
        {
            MessageBox(hWnd,"Unable to load font file.,"Error",
                MB_ICONSTOP|MB_OK);
            DestroyWindow(hWnd);
            return 0;
        }

        fg_setcolor(255);
        fg_fillpage();

        // display two colors in the same string
        fg_setcolor(19);
        fg_move(20,40);
        fgf_print("color 19\x07F\x014 color 20",19);

        // show current color is the last color used
        fg_move(20,80);
        fgf_print("still color 20",14);

        // display underlined characters
        fg_setcolor(22);
        fg_move(20,120);
        fgf_kerning(1);
        fgf_print("one \x07F_word\x07F_ is underlined",26);
        return 0;

    case WM_PAINT:
        BeginPaint(hWnd,&ps);
        fg_vbscale(0,fg_getmaxx(),0,fg_getmaxy(),0,cxClient-1,0,cyClient-1);
        EndPaint(hWnd,&ps);
        return 0;

    case WM_SETFOCUS:
        fg_realize(hPal);
        InvalidateRect(hWnd,NULL,TRUE);
        return 0;

    case WM_SIZE:
        cxClient = LOWORD(lParam);
        cyClient = HIWORD(lParam);
        return 0;

    case WM_DESTROY:
        fgf_unload(-1);
        fg_vbclose();
        fg_vbfree(hVB);
        fg_vbfin();
        DeleteObject(hPal);
        ReleaseDC(hWnd,hDC);
        PostQuitMessage(0);
        return 0;
}
return DefWindowProc(hWnd,iMsg,wParam,lParam);
}

```

FGFW2: C++Builder Version

```

/*****\
*
* FGFW2.cpp
* FGFW2U.cpp
*

```

```
* Demonstrate color selection, underlining, and kerning with fgf_print(). *
*
\*****/

#include <vcl\vcl.h>
#pragma hdrstop

#include "FGFW2U.h"
//-----
#pragma resource "*.dfm"
TForm1 *Form1;
//-----
__fastcall TForm1::TForm1(TComponent* Owner)
: TForm(Owner)
{
}
//-----
void __fastcall TForm1::FormActivate(TObject *Sender)
{
    fg_realize(hPal);
    Invalidate();
}
//-----
void __fastcall TForm1::FormCreate(TObject *Sender)
{
    HDC = GetDC(Form1->Handle);
    fg_setdc(HDC);
    hPal = fg_defpal();
    fg_realize(hPal);

    fg_vbinit();
    hVB = fg_vballoc(640,480);
    fg_vbopen(hVB);
    fg_vbcolors();

    if (fgf_load("AUSTIN36.FGF") == 0)
    {
        MessageDlg("Unable to load font file.", mtError, TMsgDlgButtons() << mbOK, 0);
        Close();
        return;
    }

    fg_setcolor(255);
    fg_fillpage();

    // display two colors in the same string
    fg_setcolor(19);
    fg_move(20,40);
    fgf_print("color 19\x07F\x014 color 20",19);

    // show current color is the last color used
    fg_move(20,80);
    fgf_print("still color 20",14);

    // display underlined characters
    fg_setcolor(22);
    fg_move(20,120);
    fgf_kerning(1);
    fgf_print("one \x07F_word\x07F_ is underlined",26);

    Application->OnActivate = OnActivate;
}
//-----
void __fastcall TForm1::FormPaint(TObject *Sender)
{
    fg_vbscale(0, fg_getmaxx(), 0, fg_getmaxy(), 0, cxClient-1, 0, cyClient-1);
}
//-----
void __fastcall TForm1::FormResize(TObject *Sender)
{
    cxClient = ClientWidth;
    cyClient = ClientHeight;
    Invalidate();
}
//-----
void __fastcall TForm1::FormDestroy(TObject *Sender)
```

```

{
  fgf_unload(-1);
  fg_vbclose();
  fg_vbfree(hVB);
  fg_vbfin();
  DeleteObject(hPal);
  ReleaseDC(Form1->Handle,hDC);
}

```

FGFW2: Delphi Version

```

{*****}
*
*   FGF2U.dpr
*   FGF2U.pas
*
*   Demonstrate the basic Fastgraph/Fonts for Windows operations of font
*   loading, character display and justification, and font unloading.
*
*****}

unit Fgfw2u;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, FGWin, FGFWin;

type
  TForm1 = class(TForm)
    procedure AppOnActivate(Sender: TObject);
    procedure FormActivate(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormPaint(Sender: TObject);
    procedure FormResize(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}
var
  dc      : hDC;
  hPal    : hPalette;
  hVB     : integer;
  cxClient, cyClient : integer;

procedure TForm1.AppOnActivate(Sender: TObject);
begin
  fg_realize(hPal);
  Invalidate;
end;

procedure TForm1.FormActivate(Sender: TObject);
begin
  fg_realize(hPal);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  dc := GetDC(Form1.Handle);
  fg_setdc(dc);
  hPal := fg_defpal;
  fg_realize(hPal);

  fg_vbinit;
  hVB := fg_vballoc(640,480);
  fg_vbopen(hVB);
  fg_vbcolors;

  if fgf_load('AUSTIN36.FGF'+chr(0)) = 0 then
  begin

```

```

    MessageDlg('Unable to load font file.', mtError, [mbOK], 0);
    Close;
    Exit;
end;

fg_setcolor(255);
fg_fillpage;

{ display two colors in the same string }
fg_setcolor(19);
fg_move(20,40);
fgf_print('color 19'#127#20' color 20',19);

{ show current color is the last color used }
fg_move(20,80);
fgf_print('still color 20',14);

{ display underlined characters }
fg_setcolor(22);
fg_move(20,120);
fgf_kerning(1);
fgf_print('one '#127'_word'#127'_ is underlined',26);

Application.OnActivate := AppOnActivate;
end;

procedure TForm1.FormPaint(Sender: TObject);
begin
    fg_vbyscale(0,fg_getmaxx,0,fg_getmaxy,0,cxClient-1,0,cyClient-1);
end;

procedure TForm1.FormResize(Sender: TObject);
begin
    cxClient := ClientWidth;
    cyClient := ClientHeight;
    Invalidate;
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
    fgf_unload(-1);
    fg_vbclose;
    fg_vbfree(hVB);
    fg_vbfin;
    DeleteObject(hPal);
    ReleaseDC(Form1.Handle,dc);
end;

end.

```

FGFW2: Visual Basic Version

```

*****
'
'   FGFW2.frm
'
'   Demonstrate color selection, underlining, and kerning with fgf_print().
'
*****

```

```

Dim hPal As Long
Dim hVB As Long
Dim cxClient As Long, cyClient As Long

```

```

Private Sub Form_Activate()
    Call fg_realize(hPal)
    Refresh
End Sub

```

```

Private Sub Form_Load()
    ScaleMode = 3
    Call fg_setdc(hDC)
    hPal = fg_defpal()
    Call fg_realize(hPal)

    Call fg_vbinit

```

```

hVB = fg_vballoc(640, 480)
Call fg_vbopen(hVB)
Call fg_vbcolors

If fgf_load(App.Path & "\AUSTIN36.FGF") = 0 Then
    Call MsgBox("Unable to load font file.", vbCritical, "Error")
    Unload Me
    Exit Sub
End If

Call fg_setcolor(255)
Call fg_fillpage

' display two colors in the same string
Call fg_setcolor(19)
Call fg_move(20, 40)
Call fgf_print("color 19 " + Chr$(127) + Chr$(20) + "color 20", 19)

' show current color is the last color used
Call fg_move(20, 80)
Call fgf_print("still color 20", 14)

' display underlined characters
Call fg_setcolor(22)
Call fg_move(20, 120)
Call fgf_kerning(1)
Call fgf_print("one " + Chr$(127) + "_word" + Chr$(127) + "_ is underlined", 26)
End Sub

Private Sub Form_Paint()
    Call fg_vbscale(0, fg_getmaxx(), 0, fg_getmaxy(), 0, cxClient - 1, 0, cyClient - 1)
End Sub

Private Sub Form_Resize()
    cxClient = ScaleWidth
    cyClient = ScaleHeight
    Refresh
End Sub

Private Sub Form_Unload(Cancel As Integer)
    Call fgf_unload(-1)
    Call fg_vbclose
    Call fg_vbfree(hVB)
    Call fg_vbfin
End Sub

```

Notice how we use each language's notation of including non-printable characters within strings to specify the option prefix codes and color selection byte. In the C/C++ version, for example, the option prefix code appears as `\x07F`, and the following byte to select color 20 appears as `\x014`. The string length passed to `fgf_print()` or `fgf_printc()` must include these additional bytes.

Using More Than One Font

The examples presented so far have not used the font handle returned by `fgf_load()` except to check if the font file was loaded successfully. When a program loads just one font, it's not necessary to use the font handle for other purposes because that font is always the current font. However, if we need to use more than one font, the font handle becomes more important because we must use it to reference the different loaded fonts. The `fgf_select()` function, which accepts a font handle as its only parameter, makes the font associated with that handle the current font.

Example FGF3 displays character strings from two different fonts (Austin 36 and Modern 28). After loading both fonts with `fgf_load()`, it calls `fgf_select()`, passing it the handle of the Modern font (this makes it the current font). It then displays a string from the Modern font. Next, the program changes fonts by passing the Austin font handle to `fgf_select()` and then displays a string from the Austin font.

FGFW3: C/C++ Version

```

/*****\

```

```

*
* FGF3.c
*
* Demonstrate how to display strings from two different fonts.
*
\*****/

#include <fgwin.h>
#include <fgfwin.h>

LRESULT CALLBACK WindowProc(HWND,UINT,WPARAM,LPARAM);

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                   PSTR szCmdParam, int iCmdShow)
{
    static char szAppName[] = "FGFW3";
    HWND        hWnd;
    MSG         msg;
    WNDCLASSEX wndclass;

    wndclass.cbSize           = sizeof(wndclass);
    wndclass.style            = CS_HREDRAW | CS_VREDRAW | CS_OWNDC;
    wndclass.lpfnWndProc      = WindowProc;
    wndclass.cbClsExtra       = 0;
    wndclass.cbWndExtra       = 0;
    wndclass.hInstance        = hInstance;
    wndclass.hIcon            = LoadIcon(NULL, IDI_APPLICATION);
    wndclass.hCursor          = LoadCursor(NULL, IDC_ARROW);
    wndclass.hbrBackground   = NULL;
    wndclass.lpszMenuName     = NULL;
    wndclass.lpszClassName    = szAppName;
    wndclass.hIconSm         = LoadIcon(NULL, IDI_APPLICATION);
    RegisterClassEx(&wndclass);

    hWnd = CreateWindow(
        szAppName,           // window class name
        "Example FGF3",     // window caption
        WS_OVERLAPPEDWINDOW, // window style
        CW_USEDEFAULT,      // initial x position
        CW_USEDEFAULT,      // initial y position
        CW_USEDEFAULT,      // initial x size
        CW_USEDEFAULT,      // initial y size
        NULL,               // parent window handle
        NULL,               // window menu handle
        hInstance,         // program instance handle
        NULL);              // creation parameters

    ShowWindow(hWnd,iCmdShow);
    UpdateWindow(hWnd);

    while (GetMessage(&msg,NULL,0,0))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
    return msg.wParam;
}

\*****\
*
* WindowProc()
*
\*****/

HDC        hDC;
HPALETTE   hPal;
int        hVB;
UINT       cxClient, cyClient;
int        hAustin, hModern;

LRESULT CALLBACK WindowProc(HWND hWnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
    PAINTSTRUCT ps;

    switch (iMsg)
    {

```

```

case WM_CREATE:
    hDC = GetDC(hWnd);
    fg_setdc(hDC);
    hPal = fg_defpal();
    fg_realize(hPal);

    fg_vbinit();
    hVB = fg_vballoc(640,480);
    fg_vbopen(hVB);
    fg_vbcolors();

    hAustin = fgf_load("AUSTIN36.FGF");
    hModern = fgf_load("MODERN28.FGF");
    if (hAustin == 0 || hModern == 0)
    {
        MessageBox(hWnd,"Unable to load font file.,"Error",
            MB_ICONSTOP|MB_OK);
        DestroyWindow(hWnd);
        return 0;
    }

    fg_setcolor(255);
    fg_fillpage();

    // display characters from the Modern font in upper left corner
    fg_setcolor(19);
    fgf_select(hModern);
    fgf_justify(-1,1);
    fgf_print("Modern 28",9);

    // display characters from the Austin font in upper right corner
    fgf_select(hAustin);
    fgf_justify(1,1);
    fg_move(fg_getmaxx(),0);
    fgf_print("Austin 36",9);
    return 0;

case WM_PAINT:
    BeginPaint(hWnd,&ps);
    fg_vbscale(0,fg_getmaxx(),0,fg_getmaxy(),0,cxClient-1,0,cyClient-1);
    EndPaint(hWnd,&ps);
    return 0;

case WM_SETFOCUS:
    fg_realize(hPal);
    InvalidateRect(hWnd,NULL,TRUE);
    return 0;

case WM_SIZE:
    cxClient = LOWORD(lParam);
    cyClient = HIWORD(lParam);
    return 0;

case WM_DESTROY:
    fgf_unload(-1);
    fg_vbclose();
    fg_vbfree(hVB);
    fg_vbfin();
    DeleteObject(hPal);
    ReleaseDC(hWnd,hDC);
    PostQuitMessage(0);
    return 0;
}
return DefWindowProc(hWnd,iMsg,wParam,lParam);
}

```

FGFW3: C++Builder Version

```

/*****\
*
* FGFW3.cpp
* FGFW3U.cpp
*
* Demonstrate how to display strings from two different fonts.
*
*****/

```

```
#include <vcl\vcl.h>
#pragma hdrstop

#include "FGFW3U.h"
//-----
#pragma resource "*.dfm"
TForm1 *Form1;
//-----
__fastcall TForm1::TForm1(TComponent* Owner)
: TForm(Owner)
{
}
//-----
void __fastcall TForm1::FormActivate(TObject *Sender)
{
    fg_realize(hPal);
    Invalidate();
}
//-----
void __fastcall TForm1::FormCreate(TObject *Sender)
{
    hDC = GetDC(Form1->Handle);
    fg_setdc(hDC);
    hPal = fg_defpal();
    fg_realize(hPal);

    fg_vbinit();
    hVB = fg_vballoc(640,480);
    fg_vbopen(hVB);
    fg_vbcolors();

    hAustin = fgf_load("AUSTIN36.FGF");
    hModern = fgf_load("MODERN28.FGF");
    if (hAustin == 0 || hModern == 0)
    {
        MessageDlg("Unable to load font file.", mtError, TMsgDlgButtons() << mbOK, 0);
        Close();
        return;
    }

    fg_setcolor(255);
    fg_fillpage();

    // display characters from the Modern font in upper left corner
    fg_setcolor(19);
    fgf_select(hModern);
    fgf_justify(-1,1);
    fgf_print("Modern 28",9);

    // display characters from the Austin font in upper right corner
    fgf_select(hAustin);
    fgf_justify(1,1);
    fg_move(fg_getmaxx(),0);
    fgf_print("Austin 36",9);

    Application->OnActivate = OnActivate;
}
//-----
void __fastcall TForm1::FormPaint(TObject *Sender)
{
    fg_vbscale(0,fg_getmaxx(),0,fg_getmaxy(),0,cxClient-1,0,cyClient-1);
}
//-----
void __fastcall TForm1::FormResize(TObject *Sender)
{
    cxClient = ClientWidth;
    cyClient = ClientHeight;
    Invalidate();
}
//-----
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
    fgf_unload(-1);
    fg_vbclose();
    fg_vbfree(hVB);
}
```

```

    fg_vbfin();
    DeleteObject(hPal);
    ReleaseDC(Form1->Handle,hDC);
}

```

FGFW3: Delphi Version

```

{*****}
*
*   FGFW3.dpr
*   FGFW3U.pas
*
*   Demonstrate how to display strings from two different fonts.
*
*****}

unit Fgfw3u;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, FGWin, FGFWin;

type
  TForm1 = class(TForm)
    procedure AppOnActivate(Sender: TObject);
    procedure FormActivate(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormPaint(Sender: TObject);
    procedure FormResize(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}
var
  dc : HDC;
  hPal : hPalette;
  hVB : integer;
  cxClient, cyClient : integer;
  hModern, hAustin : integer;

procedure TForm1.AppOnActivate(Sender: TObject);
begin
  fg_realize(hPal);
  Invalidate;
end;

procedure TForm1.FormActivate(Sender: TObject);
begin
  fg_realize(hPal);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  dc := GetDC(Form1.Handle);
  fg_setdc(dc);
  hPal := fg_defpal;
  fg_realize(hPal);

  fg_vbinit;
  hVB := fg_vballoc(640,480);
  fg_vbopen(hVB);
  fg_vbcolors;

  hAustin := fgf_load('AUSTIN36.FGF'+chr(0));
  hModern := fgf_load('MODERN28.FGF'+chr(0));
  if (hAustin = 0) or (hModern = 0) then
  begin
    MessageDlg('Unable to load font file.', mtError, [mbOK], 0);
    Close;
  end;
end;

```

```
    Exit;
end;

fg_setcolor(255);
fg_fillpage;

{ display characters from the Modern font in upper left corner }
fg_setcolor(19);
fgf_select(hModern);
fgf_justify(-1,1);
fgf_print('Modern 28',9);

{ display characters from the Austin font in upper right corner }
fgf_select(hAustin);
fgf_justify(1,1);
fg_move(fg_getmaxx,0);
fgf_print('Austin 36',9);

Application.OnActivate := AppOnActivate;
end;

procedure TForm1.FormPaint(Sender: TObject);
begin
    fg_vbscale(0,fg_getmaxx,0,fg_getmaxy,0,cxClient-1,0,cyClient-1);
end;

procedure TForm1.FormResize(Sender: TObject);
begin
    cxClient := ClientWidth;
    cyClient := ClientHeight;
    Invalidate;
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
    fgf_unload(-1);
    fg_vbclose;
    fg_vbfree(hVB);
    fg_vbfin;
    DeleteObject(hPal);
    ReleaseDC(Form1.Handle,dc);
end;

end.
```

FGFW3: Visual Basic Version

```
*****
'
'   FGFW3.frm
'
'   Demonstrate how to display strings from two different fonts.
'
*****

Dim hPal As Long
Dim hVB As Long
Dim hAustin As Long, hModern As Long
Dim cxClient As Long, cyClient As Long

Private Sub Form_Activate()
    Call fg_realize(hPal)
    Refresh
End Sub

Private Sub Form_Load()
    ScaleMode = 3
    Call fg_setdc(hDC)
    hPal = fg_defpal()
    Call fg_realize(hPal)

    Call fg_vbinit
    hVB = fg_vballoc(640, 480)
    Call fg_vbopen(hVB)
    Call fg_vbcolors
```

```

hAustin = fgf_load(App.Path & "\AUSTIN36.FGF")
hModern = fgf_load(App.Path & "\MODERN28.FGF")
If hAustin = 0 Or hModern = 0 Then
    Call MsgBox("Unable to load font file.", vbCritical, "Error")
    Unload Me
    Exit Sub
End If

Call fg_setcolor(255)
Call fg_fillpage

' display characters from the Modern font in upper left corner
Call fg_setcolor(19)
Call fgf_select(hModern)
Call fgf_justify(-1, 1)
Call fgf_print("Modern 28", 9)

' display characters from the Austin font in upper right corner
Call fgf_select(hAustin)
Call fgf_justify(1, 1)
Call fg_move(fgf_getmaxx(), 0)
Call fgf_print("Austin 36", 9)
End Sub

Private Sub Form_Paint()
    Call fg_vbscale(0, fg_getmaxx(), 0, fg_getmaxy(), 0, cxClient - 1, 0, cyClient - 1)
End Sub

Private Sub Form_Resize()
    cxClient = ScaleWidth
    cyClient = ScaleHeight
    Refresh
End Sub

Private Sub Form_Unload(Cancel As Integer)
    Call fgf_unload(-1)
    Call fg_vbclose
    Call fg_vbfree(hVB)
    Call fg_vbfin
End Sub

```

Manual Font Loading

As mentioned earlier, we can load font files automatically with **fgf_load()** or manually with **fgf_define()**. Example FGFW4 illustrates manual font loading. In this example, the font file contents are read into a fixed 8,258-byte array, though in practice it's usually preferable to allocate dynamic memory for this purpose. The size of the array (or allocated memory block) must be at least as large as the font file. The **fgf_define()** function reformats the font data, so you cannot rely on the array's original contents after **fgf_define()** returns.

FGFW4: C/C++ Version

```

/*****\
*
* FGFW4.c
*
* Demonstrate manual font loading.
*
\*****/

#include <fgwin.h>
#include <fgfwin.h>
#include <stdio.h>

LRESULT CALLBACK WindowProc(HWND,UINT,WPARAM,LPARAM);

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdParam, int iCmdShow)
{
    static char szAppName[] = "FGFW4";
    HWND      hWnd;

```

```

MSG          msg;
WNDCLASSEX  wndclass;

wndclass.cbSize      = sizeof(wndclass);
wndclass.style       = CS_HREDRAW | CS_VREDRAW | CS_OWNDC;
wndclass.lpfWndProc  = WindowProc;
wndclass.cbClsExtra  = 0;
wndclass.cbWndExtra  = 0;
wndclass.hInstance   = hInstance;
wndclass.hIcon       = LoadIcon(NULL, IDI_APPLICATION);
wndclass.hCursor     = LoadCursor(NULL, IDC_ARROW);
wndclass.hbrBackground = NULL;
wndclass.lpszMenuName = NULL;
wndclass.lpszClassName = szAppName;
wndclass.hIconSm     = LoadIcon(NULL, IDI_APPLICATION);
RegisterClassEx(&wndclass);

hWnd = CreateWindow(
    szAppName,           // window class name
    "Example FGFW4",    // window caption
    WS_OVERLAPPEDWINDOW, // window style
    CW_USEDEFAULT,      // initial x position
    CW_USEDEFAULT,      // initial y position
    CW_USEDEFAULT,      // initial x size
    CW_USEDEFAULT,      // initial y size
    NULL,               // parent window handle
    NULL,               // window menu handle
    hInstance,          // program instance handle
    NULL);              // creation parameters

ShowWindow(hWnd, iCmdShow);
UpdateWindow(hWnd);

while (GetMessage(&msg, NULL, 0, 0))
{
    TranslateMessage(&msg);
    DispatchMessage(&msg);
}
return msg.wParam;
}

/*****\
*
* WindowProc()
*
*****/

HDC      hDC;
HPALETTE hPal;
int      hVB;
UINT     cxClient, cyClient;

#define FONTSIZE 8258
BYTE FontArray[FONTSIZE];

LRESULT CALLBACK WindowProc(HWND hWnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
    PAINTSTRUCT ps;
    FILE *Stream;

    switch (iMsg)
    {
        case WM_CREATE:
            hDC = GetDC(hWnd);
            fg_setdc(hDC);
            hPal = fg_defpal();
            fg_realize(hPal);

            fg_vbinit();
            hVB = fg_vballoc(640, 480);
            fg_vbopen(hVB);
            fg_vbcolors();

            Stream = fopen("MODERN28.FGF", "rb");
            if (Stream == NULL)
            {

```

```

        MessageBox(hWnd, "Unable to open font file.", "Error",
            MB_ICONSTOP|MB_OK);
        DestroyWindow(hWnd);
        return 0;
    }
    fread(FontArray, 1, FONTSIZE, Stream);
    fclose(Stream);
    fgf_define(FontArray);

    fg_setcolor(255);
    fg_fillpage();

    fg_setcolor(19);
    fg_move(fg_getmaxx()/2, fg_getmaxy()/2);
    fgf_justify(0, 0);
    fgf_print("FG/Fonts", 8);
    return 0;

case WM_PAINT:
    BeginPaint(hWnd, &ps);
    fg_vbyscale(0, fg_getmaxx(), 0, fg_getmaxy(), 0, cxClient-1, 0, cyClient-1);
    EndPaint(hWnd, &ps);
    return 0;

case WM_SETFOCUS:
    fg_realize(hPal);
    InvalidateRect(hWnd, NULL, TRUE);
    return 0;

case WM_SIZE:
    cxClient = LOWORD(lParam);
    cyClient = HIWORD(lParam);
    return 0;

case WM_DESTROY:
    fg_vbclose();
    fg_vbfree(hVB);
    fg_vbfin();
    DeleteObject(hPal);
    ReleaseDC(hWnd, hDC);
    PostQuitMessage(0);
    return 0;
}
return DefWindowProc(hWnd, iMsg, wParam, lParam);
}

```

FGFW4: C++Builder Version

```

/*****\
*
* FGFW4.cpp
* FGFW4U.cpp
* Demonstrate manual font loading.
*
/*****/

#include <vcl\vcl.h>
#pragma hdrstop

#include "FGFW4U.h"
//-----
#pragma resource "*.dfm"
TForm1 *Form1;
//-----
__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}
//-----
void __fastcall TForm1::FormActivate(TObject *Sender)
{
    fg_realize(hPal);
    Invalidate();
}
//-----

```

```

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    FILE *Stream;

    hDC = GetDC(Form1->Handle);
    fg_setdc(hDC);
    hPal = fg_defpal();
    fg_realize(hPal);

    fg_vbinit();
    hVB = fg_vballoc(640,480);
    fg_vbopen(hVB);
    fg_vbcolors();

    Stream = fopen("MODERN28.FGF","rb");
    if (Stream == NULL)
    {
        MessageDlg("Unable to load font file.",mtError,TMsgDlgButtons()<mbOK,0);
        Close();
        return;
    }
    fread(FontArray,1,FontSize,Stream);
    fclose(Stream);
    fgf_define(FontArray);

    fg_setcolor(255);
    fg_fillpage();

    fg_setcolor(19);
    fg_move(fg_getmaxx()/2,fg_getmaxy()/2);
    fgf_justify(0,0);
    fgf_print("FG/Fonts",8);

    Application->OnActivate = OnActivate;
}
//-----
void __fastcall TForm1::FormPaint(TObject *Sender)
{
    fg_vbscale(0,fg_getmaxx(),0,fg_getmaxy(),0,cxClient-1,0,cyClient-1);
}
//-----
void __fastcall TForm1::FormResize(TObject *Sender)
{
    cxClient = ClientWidth;
    cyClient = ClientHeight;
    Invalidate();
}
//-----
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
    fg_vbclose();
    fg_vbfree(hVB);
    fg_vbfin();
    DeleteObject(hPal);
    ReleaseDC(Form1->Handle,hDC);
}

```

FGFW4: Delphi Version

```

{*****}
*
* FGFW4.dpr
* FGFW4U.pas
*
* Demonstrate manual font loading.
*
*****}

unit Fgfw4u;

interface

uses
    SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
    Forms, Dialogs, FGWin, FGFWin;

```

```

type
  TForm1 = class(TForm)
    procedure AppOnActivate(Sender: TObject);
    procedure FormActivate(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormPaint(Sender: TObject);
    procedure FormResize(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}
const
  FONTSIZE = 8258;

var
  dc : HDC;
  hPal : hPalette;
  hVB : integer;
  cxClient, cyClient : integer;
  FontArray : array [1..FONTSIZE] of byte;

procedure TForm1.AppOnActivate(Sender: TObject);
begin
  fg_realize(hPal);
  Invalidate;
end;

procedure TForm1.FormActivate(Sender: TObject);
begin
  fg_realize(hPal);
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  Stream : file;
begin
  dc := GetDC(Form1.Handle);
  fg_setdc(dc);
  hPal := fg_defpal;
  fg_realize(hPal);

  fg_vbinit;
  hVB := fg_vballoc(640,480);
  fg_vbopen(hVB);
  fg_vbcolors;

  {$I-}
  AssignFile(Stream, 'MODERN28.FGF');
  Reset(Stream,1);
  if IOResult <> 0 then
  begin
    MessageDlg('Unable to load font file.', mtError, [mbOK], 0);
    Close;
    Exit;
  end;
  BlockRead(Stream, FontArray, FONTSIZE);
  CloseFile(Stream);
  {$I+}
  fgf_define(FontArray);

  fg_setcolor(255);
  fg_fillpage;

  fg_setcolor(19);
  fg_move(fg_getmaxx div 2, fg_getmaxy div 2);
  fgf_justify(0,0);
  fgf_print('FG/Fonts', 8);

  Application.OnActivate := AppOnActivate;
end;

```

```
procedure TForm1.FormPaint(Sender: TObject);
begin
    fg_vbscale(0,fg_getmaxx,0,fg_getmaxy,0,cxClient-1,0,cyClient-1);
end;

procedure TForm1.FormResize(Sender: TObject);
begin
    cxClient := ClientWidth;
    cyClient := ClientHeight;
    Invalidate;
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
    fg_vbclose;
    fg_vbfree(hVB);
    fg_vbfin;
    DeleteObject(hPal);
    ReleaseDC(Form1.Handle,dc);
end;

end.
```

FGFW4: Visual Basic Version

```
*****
'
'   FGFW4.frm
'
'   Demonstrate manual font loading.
'
*****

Const FontFileSize = 8258

Dim hPal As Long
Dim hVB As Long
Dim cxClient As Long, cyClient As Long

Private Sub Form_Activate()
    Call fg_realize(hPal)
    Refresh
End Sub

Private Sub Form_Load()
    Dim FontArray() As Byte

    ScaleMode = 3
    Call fg_setdc(hDC)
    hPal = fg_defpal()
    Call fg_realize(hPal)

    Call fg_vbinit
    hVB = fg_vballoc(640, 480)
    Call fg_vbopen(hVB)
    Call fg_vbcolors

    On Error Resume Next
    Call FileLen(App.Path & "\MODERN28.FGF")
    If Err.Number > 0 Then
        Call MsgBox("Unable to open font file.", vbCritical, "Error")
        Unload Me
        Exit Sub
    End If
    ReDim FontArray(FontFileSize)
    Open App.Path & "\MODERN28.FGF" For Binary Access Read As #1
    Get #1, , FontArray
    Close #1
    Call fgf_define(FontArray(0))

    Call fg_setcolor(255)
    Call fg_fillpage

    Call fg_setcolor(19)
    Call fg_move(fg_getmaxx() / 2, fg_getmaxy() / 2)
```

```

    Call fgf_justify(0, 0)
    Call fgf_print("FG/Fonts", 8)
    Erase FontArray
End Sub

Private Sub Form_Paint()
    Call fg_vbscale(0, fg_getmaxx(), 0, fg_getmaxy(), 0, cxClient - 1, 0, cyClient - 1)
End Sub

Private Sub Form_Resize()
    cxClient = ScaleWidth
    cyClient = ScaleHeight
    Refresh
End Sub

Private Sub Form_Unload(Cancel As Integer)
    Call fg_vbclose
    Call fg_vbfree(hVB)
    Call fg_vbfin
End Sub

```

If your program uses manual font loading and allocates dynamic memory for a font, it must not free the memory until the font is no longer needed. If you free memory containing a font and later try to display characters from that font, the results are unpredictable. Recall that the font array passed to **fgf_define()** must be a pinned object when using the .NET framework.

When you use **fgf_unload()** to unload a font previously loaded with **fgf_load()**, Fastgraph/Fonts deallocates the font memory and releases the font handle. If you load a font with **fgf_define()**, your program is responsible for the necessary memory management functions to release the font memory, but that does not make the font handle available again. If you want to reclaim a font handle, use **fgf_undefine()**. It's not necessary to use **fgf_undefine()** unless your program must perform more than 256 font loads during its execution.

String Measurement

Fastgraph/Fonts includes two functions, **fgf_width()** and **fgf_height()**, to determine the size in pixels of a character string when displayed in the current font. The width of a string is obvious, but such is not the case for the height. We'll define the height of a string as the height of its tallest character; the height includes extra pixel rows reserved for descender characters, even if those rows aren't used. Both **fgf_width()** and **fgf_height()** expect two parameters: the string to measure and the number of characters to consider in the string. They return the computed width or height as their function value. In this section we'll look at two examples that use these functions.

Example FGF5 displays text centered within a rectangle. It uses **fgf_width()** and **fgf_height()** to determine the size of the rectangle and then centers the text inside it. The rectangle's extremes are chosen to extend four pixels beyond the edge of the text in each direction.

FGFW5: C/C++ Version

```

/*****\
*
* FGF5.c
*
* Demonstrate how to display centered text within a rectangle.
*
\*****/

#include <fgwin.h>
#include <fgfwin.h>

LRESULT CALLBACK WindowProc(HWND, UINT, WPARAM, LPARAM);

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                   PSTR szCmdParam, int iCmdShow)
{
    static char szAppName[] = "FGFW5";

```

```

HWND      hWnd;
MSG       msg;
WNDCLASSEX wndclass;

wndclass.cbSize      = sizeof(wndclass);
wndclass.style       = CS_HREDRAW | CS_VREDRAW | CS_OWNDC;
wndclass.lpfnWndProc = WindowProc;
wndclass.cbClsExtra  = 0;
wndclass.cbWndExtra  = 0;
wndclass.hInstance  = hInstance;
wndclass.hIcon       = LoadIcon(NULL, IDI_APPLICATION);
wndclass.hCursor     = LoadCursor(NULL, IDC_ARROW);
wndclass.hbrBackground = NULL;
wndclass.lpszMenuName = NULL;
wndclass.lpszClassName = szAppName;
wndclass.hIconSm     = LoadIcon(NULL, IDI_APPLICATION);
RegisterClassEx(&wndclass);

hWnd = CreateWindow(
    szAppName,          // window class name
    "Example FGFW5",   // window caption
    WS_OVERLAPPEDWINDOW, // window style
    CW_USEDEFAULT,     // initial x position
    CW_USEDEFAULT,     // initial y position
    CW_USEDEFAULT,     // initial x size
    CW_USEDEFAULT,     // initial y size
    NULL,              // parent window handle
    NULL,              // window menu handle
    hInstance,         // program instance handle
    NULL);             // creation parameters

ShowWindow(hWnd, iCmdShow);
UpdateWindow(hWnd);

while (GetMessage(&msg, NULL, 0, 0))
{
    TranslateMessage(&msg);
    DispatchMessage(&msg);
}
return msg.wParam;
}

/*****\
*
* WindowProc()
*
*****/

HDC      hDC;
HPALETTE hPal;
int      hVB;
UINT     cxClient, cyClient;

LRESULT CALLBACK WindowProc(HWND hWnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
    PAINTSTRUCT ps;
    int dx, dy;
    int xMid, yMid;
    char MessageText[14];

    switch (iMsg)
    {
        case WM_CREATE:
            hDC = GetDC(hWnd);
            fg_setdc(hDC);
            hPal = fg_defpal();
            fg_realize(hPal);

            fg_vbinit();
            hVB = fg_vballoc(640, 480);
            fg_vbopen(hVB);
            fg_vbcolors();

            if (fgf_load("AUSTIN36.FGF") == 0)
            {
                MessageBox(hWnd, "Unable to load font file.", "Error",

```

```

        MB_ICONSTOP|MB_OK);
    DestroyWindow(hWnd);
    return 0;
}

fg_setcolor(255);
fg_fillpage();

xMid = fg_getmaxx() / 2;
yMid = fg_getmaxy() / 2;
lstrcpy(MessageText, "TEXT IN A BOX");
dx = fgf_width(MessageText, 13) / 2 + 4;
dy = fgf_height(MessageText, 13) / 2 + 4;

fg_setcolor(17);
fg_rect(xMid-dx, xMid+dx, yMid-dy, yMid+dy);
fg_setcolor(20);
fg_move(xMid, yMid);
fgf_justify(0, 0);
fgf_print(MessageText, 13);
return 0;

case WM_PAINT:
    BeginPaint(hWnd, &ps);
    fg_vbscale(0, fg_getmaxx(), 0, fg_getmaxy(), 0, cxClient-1, 0, cyClient-1);
    EndPaint(hWnd, &ps);
    return 0;

case WM_SETFOCUS:
    fg_realize(hPal);
    InvalidateRect(hWnd, NULL, TRUE);
    return 0;

case WM_SIZE:
    cxClient = LOWORD(lParam);
    cyClient = HIWORD(lParam);
    return 0;

case WM_DESTROY:
    fgf_unload(-1);
    fg_vbclose();
    fg_vbfree(hVB);
    fg_vbfin();
    DeleteObject(hPal);
    ReleaseDC(hWnd, hDC);
    PostQuitMessage(0);
    return 0;
}
return DefWindowProc(hWnd, iMsg, wParam, lParam);
}

```

FGFW5: C++Builder Version

```

\*****\
*
*   FGF5.cpp
*   FGF5U.cpp
*
*   Demonstrate how to display centered text within a rectangle.
*
\*****/

#include <vcl\vcl.h>
#pragma hdrstop

#include "FGFW5U.h"
//-----
#pragma resource "*.dfm"
TForm1 *Form1;
//-----
__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}
//-----
void __fastcall TForm1::FormActivate(TObject *Sender)

```

```

{
  fg_realize(hPal);
  Invalidate();
}
//-----
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  int dx, dy;
  int xMid, yMid;
  char MessageText[14];

  hDC = GetDC(Form1->Handle);
  fg_setdc(hDC);
  hPal = fg_defpal();
  fg_realize(hPal);

  fg_vbinit();
  hVB = fg_vballoc(640,480);
  fg_vbopen(hVB);
  fg_vbcolors();

  if (fgf_load("AUSTIN36.FGF") == 0)
  {
    MessageDlg("Unable to load font file.",mtError,TMsgDlgButtons()<<mbOK,0);
    Close();
    return;
  }

  fg_setcolor(255);
  fg_fillpage();

  xMid = fg_getmaxx() / 2;
  yMid = fg_getmaxy() / 2;
  lstrcpy(MessageText,"TEXT IN A BOX");
  dx = fgf_width(MessageText,13) / 2 + 4;
  dy = fgf_height(MessageText,13) / 2 + 4;

  fg_setcolor(17);
  fg_rect(xMid-dx,xMid+dx,yMid-dy,yMid+dy);
  fg_setcolor(20);
  fg_move(xMid,yMid);
  fgf_justify(0,0);
  fgf_print(MessageText,13);

  Application->OnActivate = OnActivate;
}
//-----
void __fastcall TForm1::FormPaint(TObject *Sender)
{
  fg_vbscale(0,fg_getmaxx(),0,fg_getmaxy(),0,cxClient-1,0,cyClient-1);
}
//-----
void __fastcall TForm1::FormResize(TObject *Sender)
{
  cxClient = ClientWidth;
  cyClient = ClientHeight;
  Invalidate();
}
//-----
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  fgf_unload(-1);
  fg_vbclose();
  fg_vbfree(hVB);
  fg_vbfin();
  DeleteObject(hPal);
  ReleaseDC(Form1->Handle,hDC);
}

```

FGFW5: Delphi Version

```

{*****
*
* FGF5.dpr
* FGF5U.pas
*
*

```

```

* Demonstrate how to display centered text within a rectangle.          *
*                                                                           *
*****}
unit Fgfw5u;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, FGWin, FGFWin;

type
  TForm1 = class(TForm)
    procedure AppOnActivate(Sender: TObject);
    procedure FormActivate(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormPaint(Sender: TObject);
    procedure FormResize(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}
var
  dc : HDC;
  hPal : hPalette;
  hVB : integer;
  cxClient, cyClient : integer;

procedure TForm1.AppOnActivate(Sender: TObject);
begin
  fg_realize(hPal);
  Invalidate;
end;

procedure TForm1.FormActivate(Sender: TObject);
begin
  fg_realize(hPal);
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  dx, dy : integer;
  xMid, yMid : integer;
  MessageText : string;
begin
  dc := GetDC(Form1.Handle);
  fg_setdc(dc);
  hPal := fg_defpal;
  fg_realize(hPal);

  fg_vbinit;
  hVB := fg_vballoc(640,480);
  fg_vbopen(hVB);
  fg_vbcolors;

  if fgf_load('AUSTIN36.FGF'+chr(0)) = 0 then
  begin
    MessageDlg('Unable to load font file.', mtError, [mbOK], 0);
    Close;
    Exit;
  end;

  fg_setcolor(255);
  fg_fillpage;

  xMid := fg_getmaxx div 2;
  yMid := fg_getmaxy div 2;
  MessageText := 'TEXT IN A BOX';
  dx := fgf_width(MessageText,13) div 2 + 4;

```

```
dy := fgf_height(MessageText,13) div 2 + 4;

fg_setcolor(17);
fg_rect(xMid-dx,xMid+dx,yMid-dy,yMid+dy);
fg_setcolor(20);
fg_move(xMid,yMid);
fgf_justify(0,0);
fgf_print(MessageText,13);

Application.OnActivate := AppOnActivate;
end;

procedure TForm1.FormPaint(Sender: TObject);
begin
  fg_vbyscale(0,fg_getmaxx,0,fg_getmaxy,0,cxClient-1,0,cyClient-1);
end;

procedure TForm1.FormResize(Sender: TObject);
begin
  cxClient := ClientWidth;
  cyClient := ClientHeight;
  Invalidate;
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  fgf_unload(-1);
  fg_vbclose;
  fg_vbfree(hVB);
  fg_vbfin;
  DeleteObject(hPal);
  ReleaseDC(Form1.Handle,dc);
end;

end.
```

FGFW5: Visual Basic Version

```
'*****
'
'   FGFW5.frm
'
'   Demonstrate how to display centered text within a rectangle.
'
'*****

Dim hPal As Long
Dim hVB As Long
Dim cxClient As Long, cyClient As Long

Private Sub Form_Activate()
  Call fg_realize(hPal)
  Refresh
End Sub

Private Sub Form_Load()
  Dim dx As Long, dy As Long
  Dim xMid As Long, yMid As Long

  ScaleMode = 3
  Call fg_setdc(hDC)
  hPal = fg_defpal()
  Call fg_realize(hPal)

  Call fg_vbinit
  hVB = fg_vballoc(640, 480)
  Call fg_vbopen(hVB)
  Call fg_vbcolors

  If fgf_load(App.Path & "\AUSTIN36.FGF") = 0 Then
    Call MsgBox("Unable to load font file.", vbCritical, "Error")
    Unload Me
    Exit Sub
  End If

  Call fg_setcolor(255)
```

```

Call fg_fillpage

xMid = fg_getmaxx() / 2
yMid = fg_getmaxy() / 2
MessageText = "TEXT IN A BOX"
dx = fgf_width(MessageText, 13) / 2 + 4
dy = fgf_height(MessageText, 13) / 2 + 4

Call fg_setcolor(17)
Call fg_rect(xMid - dx, xMid + dx, yMid - dy, yMid + dy)
Call fg_setcolor(20)
Call fg_move(xMid, yMid)
Call fgf_justify(0, 0)
Call fgf_print(MessageText, 13)
End Sub

Private Sub Form_Paint()
Call fg_vbyscale(0, fg_getmaxx(), 0, fg_getmaxy(), 0, cxClient - 1, 0, cyClient - 1)
End Sub

Private Sub Form_Resize()
cxClient = ScaleWidth
cyClient = ScaleHeight
Refresh
End Sub

Private Sub Form_Unload(Cancel As Integer)
Call fgf_unload(-1)
Call fg_vbclose
Call fg_vbfree(hVB)
Call fg_vbfin
End Sub

```

The easiest way to underline characters is to use the special underlining sequences recognized by **fgf_print()** and **fgf_printc()**. Suppose, however, that we instead want to perform the underlining manually, perhaps to draw a thicker underline. We'll do this by drawing a line just beneath the characters, but how do we compute the position of this line? Determining the line length is easy -- it's the same as the width of the string being underlined. We want the vertical position of the line to be two pixels below the characters, except for descender characters (which will extend below the underline).

Earlier in this section, we said the height of a string includes the space reserved for descender characters. This means if we used only the string height as the basis for drawing the underline, it would appear below the tips of the descenders. Fortunately, Fastgraph/Fonts provides an easy solution. The function **fgf_drop()** returns as its function value the number of pixel rows descender characters extend below the other characters. Hence, the vertical coordinate for the underline is the height of the string, less the value of **fgf_drop()**, plus two pixel rows (the two pixel rows provide one blank row between the characters and the underline). Example FGF6 demonstrates this process.

FGFW6: C/C++ Version

```

/*****\
*
* FGF6.c
*
* Demonstrate manual underlining.
*
\*****/

#include <fgwin.h>
#include <fgfwin.h>

LRESULT CALLBACK WindowProc(HWND,UINT,WPARAM,LPARAM);

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                   PSTR szCmdParam, int iCmdShow)
{
    static char szAppName[] = "FGFW6";
    HWND      hWnd;
    MSG       msg;
    WNDCLASSEX wndclass;

```

```

wndclass.cbSize           = sizeof(wndclass);
wndclass.style            = CS_HREDRAW | CS_VREDRAW | CS_OWNDC;
wndclass.lpfWndProc       = WindowProc;
wndclass.cbClsExtra       = 0;
wndclass.cbWndExtra       = 0;
wndclass.hInstance       = hInstance;
wndclass.hIcon            = LoadIcon(NULL, IDI_APPLICATION);
wndclass.hCursor          = LoadCursor(NULL, IDC_ARROW);
wndclass.hbrBackground   = NULL;
wndclass.lpszMenuName    = NULL;
wndclass.lpszClassName   = szAppName;
wndclass.hIconSm         = LoadIcon(NULL, IDI_APPLICATION);
RegisterClassEx(&wndclass);

hWnd = CreateWindow(
    szAppName,           // window class name
    "Example FGF6",     // window caption
    WS_OVERLAPPEDWINDOW, // window style
    CW_USEDEFAULT,      // initial x position
    CW_USEDEFAULT,      // initial y position
    CW_USEDEFAULT,      // initial x size
    CW_USEDEFAULT,      // initial y size
    NULL,               // parent window handle
    NULL,               // window menu handle
    hInstance,         // program instance handle
    NULL);             // creation parameters

ShowWindow(hWnd, iCmdShow);
UpdateWindow(hWnd);

while (GetMessage(&msg, NULL, 0, 0))
{
    TranslateMessage(&msg);
    DispatchMessage(&msg);
}
return msg.wParam;
}

/*****\
*
* WindowProc()
*
*****/

HDC      hDC;
HPALETTE hPal;
int      hVB;
UINT     cxClient, cyClient;

LRESULT CALLBACK WindowProc(HWND hWnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
    PAINTSTRUCT ps;
    int StringHeight, StringWidth;
    char MessageText[16];

    switch (iMsg)
    {
        case WM_CREATE:
            hDC = GetDC(hWnd);
            fg_setdc(hDC);
            hPal = fg_defpal();
            fg_realize(hPal);

            fg_vbinit();
            hVB = fg_vballoc(640, 480);
            fg_vbopen(hVB);
            fg_vbcolors();

            if (fgf_load("AUSTIN36.FGF") == 0)
            {
                MessageBox(hWnd, "Unable to load font file.", "Error",
                    MB_ICONSTOP | MB_OK);
                DestroyWindow(hWnd);
                return 0;
            }
    }
}

```

```

    fg_setcolor(255);
    fg_fillpage();

    lstrcpy(MessageText,"Fastgraph/Fonts");
    StringHeight = fgf_height(MessageText,15) - fgf_drop() + 2;
    StringWidth  = fgf_width(MessageText,15);

    fg_setcolor(20);
    fgf_justify(-1,1);
    fgf_print(MessageText,15);
    fg_rect(0,StringWidth-1,StringHeight,StringHeight+2);
    return 0;

case WM_PAINT:
    BeginPaint(hWnd,&ps);
    fg_vbscale(0,fg_getmaxx(),0,fg_getmaxy(),0,cxClient-1,0,cyClient-1);
    EndPaint(hWnd,&ps);
    return 0;

case WM_SETFOCUS:
    fg_realize(hPal);
    InvalidateRect(hWnd,NULL,TRUE);
    return 0;

case WM_SIZE:
    cxClient = LOWORD(lParam);
    cyClient = HIWORD(lParam);
    return 0;

case WM_DESTROY:
    fgf_unload(-1);
    fg_vbclose();
    fg_vbfree(hVB);
    fg_vbfin();
    DeleteObject(hPal);
    ReleaseDC(hWnd,hDC);
    PostQuitMessage(0);
    return 0;
}
return DefWindowProc(hWnd,iMsg,wParam,lParam);
}

```

FGFW6: C++Builder Version

```

/*****\
*
*  FGFW6.cpp
*  FGFW6U.cpp
*
*  Demonstrate manual underlining.
*
*****/

#include <vcl\vcl.h>
#pragma hdrstop

#include "FGFW6U.h"
//-----
#pragma resource "*.dfm"
TForm1 *Form1;
//-----
__fastcall TForm1::TForm1(TComponent* Owner)
: TForm(Owner)
{
}
//-----
void __fastcall TForm1::FormActivate(TObject *Sender)
{
    fg_realize(hPal);
    Invalidate();
}
//-----
void __fastcall TForm1::FormCreate(TObject *Sender)
{
    int StringHeight, StringWidth;
    char MessageText[16];

```

```

hDC = GetDC(Form1->Handle);
fg_setdc(hDC);
hPal = fg_defpal();
fg_realize(hPal);

fg_vbinit();
hVB = fg_vballoc(640,480);
fg_vbopen(hVB);
fg_vbcolors();

if (fgf_load("AUSTIN36.FGF") == 0)
{
    MessageDlg("Unable to load font file.",mtError,TMsgDlgButtons()<<mbOK,0);
    Close();
    return;
}

fg_setcolor(255);
fg_fillpage();

lstrcpy(MessageText,"Fastgraph/Fonts");
StringHeight = fgf_height(MessageText,15) - fgf_drop() + 2;
StringWidth = fgf_width(MessageText,15);

fg_setcolor(20);
fgf_justify(-1,1);
fgf_print(MessageText,15);
fg_rect(0,StringWidth-1,StringHeight,StringHeight+2);

Application->OnActivate = OnActivate;
}
//-----
void __fastcall TForm1::FormPaint(TObject *Sender)
{
    fg_vbscale(0,fg_getmaxx(),0,fg_getmaxy(),0,cxClient-1,0,cyClient-1);
}
//-----
void __fastcall TForm1::FormResize(TObject *Sender)
{
    cxClient = ClientWidth;
    cyClient = ClientHeight;
    Invalidate();
}
//-----
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
    fgf_unload(-1);
    fg_vbclose();
    fg_vbfree(hVB);
    fg_vbfin();
    DeleteObject(hPal);
    ReleaseDC(Form1->Handle,hDC);
}

```

FGFW6: Delphi Version

```

{*****
*
* FGFW6.dpr
* FGFW6U.pas
*
* Demonstrate manual underlining.
*
*****}

```

```

unit Fgfw6u;

interface

uses
    SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
    Forms, Dialogs, FGWin, FGFWin;

type
    TForm1 = class(TForm)

```

```

    procedure AppOnActivate(Sender: TObject);
    procedure FormActivate(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormPaint(Sender: TObject);
    procedure FormResize(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
end;

var
    Form1: TForm1;

implementation

{$R *.DFM}
var
    dc : HDC;
    hPal : hPalette;
    hVB : integer;
    cxClient, cyClient : integer;

procedure TForm1.AppOnActivate(Sender: TObject);
begin
    fg_realize(hPal);
    Invalidate;
end;

procedure TForm1.FormActivate(Sender: TObject);
begin
    fg_realize(hPal);
end;

procedure TForm1.FormCreate(Sender: TObject);
var
    StringHeight, StringWidth : integer;
    MessageText : string;
begin
    dc := GetDC(Form1.Handle);
    fg_setdc(dc);
    hPal := fg_defpal;
    fg_realize(hPal);

    fg_vbinit;
    hVB := fg_vballoc(640,480);
    fg_vbopen(hVB);
    fg_vbcolors;

    if fgf_load('AUSTIN36.FGF'+chr(0)) = 0 then
    begin
        MessageDlg('Unable to load font file.', mtError, [mbOK], 0);
        Close;
        Exit;
    end;

    fg_setcolor(255);
    fg_fillpage;

    MessageText := 'Fastgraph/Fonts';
    StringHeight := fgf_height(MessageText, 15) - fgf_drop + 2;
    StringWidth := fgf_width(MessageText, 15);

    fg_setcolor(20);
    fgf_justify(-1, 1);
    fgf_print(MessageText, 15);
    fg_rect(0, StringWidth-1, StringHeight, StringHeight+2);

    Application.OnActivate := AppOnActivate;
end;

procedure TForm1.FormPaint(Sender: TObject);
begin
    fg_vbscale(0, fg_getmaxx, 0, fg_getmaxy, 0, cxClient-1, 0, cyClient-1);
end;

procedure TForm1.FormResize(Sender: TObject);
begin

```

```
    cxClient := ClientWidth;
    cyClient := ClientHeight;
    Invalidate;
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
    fgf_unload(-1);
    fg_vbclose;
    fg_vbfree(hVB);
    fg_vbfin;
    DeleteObject(hPal);
    ReleaseDC(Form1.Handle,dc);
end;

end.
```

FGFW6: Visual Basic Version

```
*****
'
'   FGF6.frm
'
'   Demonstrate manual underlining.
'
*****

Dim hPal As Long
Dim hVB As Long
Dim cxClient As Long, cyClient As Long

Private Sub Form_Activate()
    Call fg_realize(hPal)
    Refresh
End Sub

Private Sub Form_Load()
    Dim StringHeight As Long, StringWidth As Long

    ScaleMode = 3
    Call fg_setdc(hDC)
    hPal = fg_defpal()
    Call fg_realize(hPal)

    Call fg_vbinit
    hVB = fg_vballoc(640, 480)
    Call fg_vbopen(hVB)
    Call fg_vbcolors

    If fgf_load(App.Path & "\AUSTIN36.FGF") = 0 Then
        Call MsgBox("Unable to load font file.", vbCritical, "Error")
        Unload Me
        Exit Sub
    End If

    Call fg_setcolor(255)
    Call fg_fillpage

    MessageText = "Fastgraph/Fonts"
    StringHeight = fgf_height(MessageText, 15) - fgf_drop() + 2
    StringWidth = fgf_width(MessageText, 15)

    Call fg_setcolor(20)
    Call fgf_justify(-1, 1)
    Call fgf_print(MessageText, 15)
    Call fg_rect(0, StringWidth - 1, StringHeight, StringHeight + 2)
End Sub

Private Sub Form_Paint()
    Call fg_vbscale(0, fg_getmaxx(), 0, fg_getmaxy(), 0, cxClient - 1, 0, cyClient - 1)
End Sub

Private Sub Form_Resize()
    cxClient = ScaleWidth
    cyClient = ScaleHeight
    Refresh
End Sub
```

```

End Sub

Private Sub Form_Unload(Cancel As Integer)
    Call fgf_unload(-1)
    Call fg_vbclose
    Call fg_vbfree(hVB)
    Call fg_vbfin
End Sub

```

Shadowed Characters

Example FGF7 illustrates a simple method of displaying characters with shadows. It displays the shadowed characters "text with shadow" in the middle of the active virtual buffer. The first call to **fgf_print()** displays the shadow in color 0 (black) offset one pixel to the right and one pixel below the position where the characters will appear. The second call to **fgf_print()** then displays the characters in color 19 where they are supposed to appear (which covers up most of the "shadow"). You can simulate different light sources by displaying the shadow in other positions and at other distances from the actual characters.

FGFW7: C/C++ Version

```

/*****\
 *
 * FGF7.c
 *
 * Demonstrate shadowed characters.
 *
 \*****/

#include <fgwin.h>
#include <fgfwin.h>

LRESULT CALLBACK WindowProc(HWND,UINT,WPARAM,LPARAM);

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdParam, int iCmdShow)
{
    static char szAppName[] = "FGFW7";
    HWND hWnd;
    MSG msg;
    WNDCLASSEX wndclass;

    wndclass.cbSize = sizeof(wndclass);
    wndclass.style = CS_HREDRAW | CS_VREDRAW | CS_OWNDC;
    wndclass.lpfnWndProc = WindowProc;
    wndclass.cbClsExtra = 0;
    wndclass.cbWndExtra = 0;
    wndclass.hInstance = hInstance;
    wndclass.hIcon = LoadIcon(NULL,IDI_APPLICATION);
    wndclass.hCursor = LoadCursor(NULL, IDC_ARROW);
    wndclass.hbrBackground = NULL;
    wndclass.lpszMenuName = NULL;
    wndclass.lpszClassName = szAppName;
    wndclass.hIconSm = LoadIcon(NULL,IDI_APPLICATION);
    RegisterClassEx(&wndclass);

    hWnd = CreateWindow(
        szAppName, // window class name
        "Example FGF7", // window caption
        WS_OVERLAPPEDWINDOW, // window style
        CW_USEDEFAULT, // initial x position
        CW_USEDEFAULT, // initial y position
        CW_USEDEFAULT, // initial x size
        CW_USEDEFAULT, // initial y size
        NULL, // parent window handle
        NULL, // window menu handle
        hInstance, // program instance handle
        NULL); // creation parameters

    ShowWindow(hWnd,iCmdShow);
    UpdateWindow(hWnd);

```

```

while (GetMessage(&msg,NULL,0,0))
{
    TranslateMessage(&msg);
    DispatchMessage(&msg);
}
return msg.wParam;
}

/*****\
*
* WindowProc()
*
\*****/

HDC      hDC;
HPALETTE hPal;
int      hVB;
UINT     cxClient, cyClient;

LRESULT CALLBACK WindowProc(HWND hWnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
    PAINTSTRUCT ps;
    int xMid, yMid;
    char MessageText[17];

    switch (iMsg)
    {
        case WM_CREATE:
            hDC = GetDC(hWnd);
            fg_setdc(hDC);
            hPal = fg_defpal();
            fg_realize(hPal);

            fg_vbinit();
            hVB = fg_vballoc(640,480);
            fg_vbopen(hVB);
            fg_vbcolors();

            if (fgf_load("AUSTIN36.FGF") == 0)
            {
                MessageBox(hWnd,"Unable to load font file.,"Error",
                    MB_ICONSTOP|MB_OK);
                DestroyWindow(hWnd);
                return 0;
            }

            fg_setcolor(255);
            fg_fillpage();

            xMid = fg_getmaxx() / 2;
            yMid = fg_getmaxy() / 2;
            lstrcpy(MessageText,"text with shadow");

            fg_setcolor(0);
            fg_move(xMid+1,yMid+1);
            fgf_justify(0,0);
            fgf_print(MessageText,16);
            fg_setcolor(19);
            fg_move(xMid,yMid);
            fgf_print(MessageText,16);
            return 0;

        case WM_PAINT:
            BeginPaint(hWnd,&ps);
            fg_vbscale(0,fg_getmaxx(),0,fg_getmaxy(),0,cxClient-1,0,cyClient-1);
            EndPaint(hWnd,&ps);
            return 0;

        case WM_SETFOCUS:
            fg_realize(hPal);
            InvalidateRect(hWnd,NULL,TRUE);
            return 0;

        case WM_SIZE:
            cxClient = LOWORD(lParam);
            cyClient = HIWORD(lParam);
    }
}

```

```

        return 0;

    case WM_DESTROY:
        fg_unload(-1);
        fg_vbclose();
        fg_vbfree(hVB);
        fg_vbfin();
        DeleteObject(hPal);
        ReleaseDC(hWnd,hDC);
        PostQuitMessage(0);
        return 0;
    }
    return DefWindowProc(hWnd,iMsg,wParam,lParam);
}

```

FGFW7: C++Builder Version

```

/*****\
 *
 *  FGFW7.cpp
 *  FGFW7U.cpp
 *
 *  Demonstrate shadowed characters.
 *
 *****/

#include <vcl\vcl.h>
#pragma hdrstop

#include "FGFW7U.h"
//-----
#pragma resource "*.dfm"
TForm1 *Form1;
//-----
__fastcall TForm1::TForm1(TComponent* Owner)
: TForm(Owner)
{
}
//-----
void __fastcall TForm1::FormActivate(TObject *Sender)
{
    fg_realize(hPal);
    Invalidate();
}
//-----
void __fastcall TForm1::FormCreate(TObject *Sender)
{
    int xMid, yMid;
    char MessageText[17];

    hDC = GetDC(Form1->Handle);
    fg_setdc(hDC);
    hPal = fg_defpal();
    fg_realize(hPal);

    fg_vbinit();
    hVB = fg_vballoc(640,480);
    fg_vbopen(hVB);
    fg_vbcolors();

    if (fgf_load("AUSTIN36.FGF") == 0)
    {
        MessageDlg("Unable to load font file.",mtError,TMsgDlgButtons()<<mbOK,0);
        Close();
        return;
    }

    fg_setcolor(255);
    fg_fillpage();

    xMid = fg_getmaxx() / 2;
    yMid = fg_getmaxy() / 2;
    lstrncpy(MessageText,"text with shadow");

    fg_setcolor(0);
    fg_move(xMid+1,yMid+1);

```

```

    fgf_justify(0,0);
    fgf_print(MessageText,16);
    fg_setcolor(19);
    fg_move(xMid,yMid);
    fgf_print(MessageText,16);

    Application->OnActivate = OnActivate;
}
//-----
void __fastcall TForm1::FormPaint(TObject *Sender)
{
    fg_vbscale(0,fg_getmaxx(),0,fg_getmaxy(),0,cxClient-1,0,cyClient-1);
}
//-----
void __fastcall TForm1::FormResize(TObject *Sender)
{
    cxClient = ClientWidth;
    cyClient = ClientHeight;
    Invalidate();
}
//-----
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
    fgf_unload(-1);
    fg_vbclose();
    fg_vbfree(hVB);
    fg_vbfin();
    DeleteObject(hPal);
    ReleaseDC(Form1->Handle,hDC);
}

```

FGFW7: Delphi Version

```

{*****
*
* FGFW7.dpr
* FGFW7U.pas
*
* Demonstrate shadowed characters.
*
*****}

unit Fgfw7u;

interface

uses
    SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
    Forms, Dialogs, FGWin, FGFWin;

type
    TForm1 = class(TForm)
    procedure AppOnActivate(Sender: TObject);
    procedure FormActivate(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormPaint(Sender: TObject);
    procedure FormResize(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    end;

var
    Form1: TForm1;

implementation

{$R *.DFM}
var
    dc : hDC;
    hPal : hPalette;
    hVB : integer;
    cxClient, cyClient : integer;

procedure TForm1.AppOnActivate(Sender: TObject);
begin
    fg_realize(hPal);
    Invalidate;

```



```
' FGF7.frm *
' *
' Demonstrate shadowed characters. *
' *
'*****

Dim hPal As Long
Dim hVB As Long
Dim cxClient As Long, cyClient As Long

Private Sub Form_Activate()
    Call fg_realize(hPal)
    Refresh
End Sub

Private Sub Form_Load()
    Dim xMid As Long, yMid As Long

    ScaleMode = 3
    Call fg_setdc(hDC)
    hPal = fg_defpal()
    Call fg_realize(hPal)

    Call fg_vbinit
    hVB = fg_vballoc(640, 480)
    Call fg_vbopen(hVB)
    Call fg_vbcolors

    If fgf_load(App.Path & "\AUSTIN36.FGF") = 0 Then
        Call MsgBox("Unable to load font file.", vbCritical, "Error")
        Unload Me
        Exit Sub
    End If

    Call fg_setcolor(255)
    Call fg_fillpage

    xMid = fg_getmaxx() / 2
    yMid = fg_getmaxy() / 2
    MessageText = "text with shadow"

    Call fg_setcolor(0)
    Call fg_move(xMid + 1, yMid + 1)
    Call fgf_justify(0, 0)
    Call fgf_print(MessageText, 16)
    Call fg_setcolor(19)
    Call fg_move(xMid, yMid)
    Call fgf_print(MessageText, 16)
End Sub

Private Sub Form_Paint()
    Call fg_vbscale(0, fg_getmaxx(), 0, fg_getmaxy(), 0, cxClient - 1, 0, cyClient - 1)
End Sub

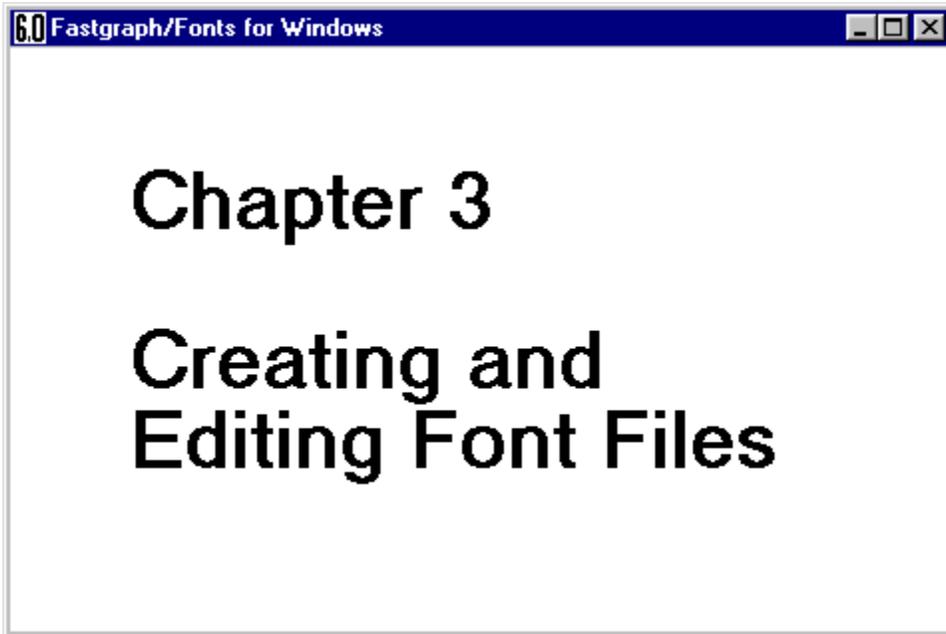
Private Sub Form_Resize()
    cxClient = ScaleWidth
    cyClient = ScaleHeight
    Refresh
End Sub

Private Sub Form_Unload(Cancel As Integer)
    Call fgf_unload(-1)
    Call fg_vbclose
    Call fg_vbfree(hVB)
    Call fg_vbfin
End Sub
```

Summary

This section summarizes the functional descriptions of the Fastgraph/Fonts functions presented in this chapter. More detailed information about these functions, including their parameters and return values, may be found in the *Fastgraph/Fonts 6.0 Reference Manual*.

<code>fgf_define()</code>	Makes externally loaded font data available to Fastgraph/Fonts.
<code>fgf_drop()</code>	Returns the number of pixel rows descender characters extend below the other characters in the current font.
<code>fgf_height()</code>	Determines the height in pixels of a given character string when displayed in the current font.
<code>fgf_justify()</code>	Defines horizontal and vertical justification settings for strings displayed with <code>fgf_print()</code> and <code>fgf_printc()</code> .
<code>fgf_kerning()</code>	Defines the amount of additional space (in pixels) inserted between characters for strings displayed with <code>fgf_print()</code> and <code>fgf_printc()</code> .
<code>fgf_load()</code>	Loads a font from an external file.
<code>fgf_print()</code>	Displays characters from the current, without clipping.
<code>fgf_printc()</code>	Displays characters from the current font, with clipping.
<code>fgf_select()</code>	Makes the selected font the current font.
<code>fgf_undefine()</code>	Releases the handle for a font loaded with <code>fgf_define()</code> .
<code>fgf_under()</code>	Defines the underline position when displaying underlined characters with <code>fgf_print()</code> or <code>fgf_printc()</code> .
<code>fgf_unload()</code>	Releases the handle for a font loaded with <code>fgf_load()</code> and releases the memory used by the font.
<code>fgf_width()</code>	Determines the width in pixels of a given character string when displayed in the current font.



Chapter 3

Creating and Editing Font Files

Overview

This chapter provides information on creating and editing font files for use with Fastgraph/Fonts. The first two sections describe the font file structure for the 96-character and 256-character formats. While it is not necessary to know these formats to use Fastgraph/Fonts, we document them here for those who are interested. The remaining sections describe the utility programs supplied with Fastgraph/Fonts, including an overview of the FGFedit interactive font editor and TrueType conversion utility.

96-Character Font File Structure

The structure of a Fastgraph/Fonts 96-character font file is surprisingly simple. A 96-character font file contains definitions for ASCII characters 32 through 127 decimal, although character 127 isn't used. Each character is stored in a bitmap that is the same size for all characters in the font. The bitmap width is the width of the font's widest character, while its height is the height of the font's tallest character. Given this information, we can summarize the font file structure in the following table.

byte offset	field size	field name	description
0	1	MaxX	The width in bytes of the font's widest character. There are 8 pixels per byte.
1	1	MaxY	The height of the font's tallest character.
2	96	Xsizes	The width in pixels plus one or more kerning pixels for each character (ASCII values 32 to 127).
98	96	Ysizes	The height in pixels for each character (ASCII values 32 to 127).
194	96*size	Bitmap	The actual bitmap for each of the 96 characters. The size of each bitmap is MaxX*MaxY.

Each character's bitmap is stored in rows from top to bottom and includes space for descender characters (even if the character is not a descender). Within each row, the bitmap values are stored from left to right. In addition, each row may include one or more blank pixels (for kerning) appended to the end of the row.

To illustrate the bitmap format, consider the following two bitmaps for the lower case characters "x" and "y". The "x" bitmap is 10 pixels wide (including one pixel of kerning) and 10 pixels high (including three rows for descenders, even though "x" is not a descender). The "y" bitmap is 9 pixels wide and 10 pixels high.

```

* * * * . . * * * . F3 80    * * * . . * * * . E7 00
. . * . . . . * . . 21 00    . * . . . . * . . 42 00
. . . * . . * . . . 12 00    . . * . . . * . . 22 00
. . . * * * . . . . 1C 00    . . * . . * . . . 24 00
. . * . . * . . . . 24 00    . . . * . * . . . 14 00
. * . . . . * . . . 42 00    . . . * * . . . . 18 00
* * * . . * * * * . E7 80    . . . . * . . . . 08 00
. . . . . . . . . . 00 00    . . . * . . . . . 10 00
. . . . . . . . . . 00 00    . . . * . . . . . 10 00
. . . . . . . . . . 00 00    * * * . . . . . . E0 00

```

The hexadecimal values to the right of each row are the bitmap values for the bytes in that row. To compute these values, set the dot (.) pixels to 0 and the asterisk (*) pixels to 1, then convert the resulting binary values to hex. When the rows do not contain a multiple of 8 pixels, logically pad them with zero pixels to compute the hex values.

96-character font files contain bitmaps for the characters whose ASCII decimal values are 32 through 127. This means the offset for a given character is

$$(ASCIIvalue - 32) * size + 194$$

where *ASCIIvalue* is the character's ASCII value, and *size* is the bitmap size for each character (the value 194 is the offset within the file for the first character's bitmap). For this example, let's assume the value of *MaxX* is 2 and the value of *MaxY* is 15, making the bitmap size 30 bytes. The ASCII value of "x" is 120, so its bitmap offset is $(120-32)*30+194$, or 2,834. Similarly, the ASCII value of "y" is 121, so its offset is 2,864. The bitmap values are then stored in the file as shown below.

```

2830   ww ww ww ww F3 80 21 00 12 00
2840   1C 00 24 00 42 00 E7 80 00 00
2850   00 00 00 00 ** ** ** ** ** ** ** **
2860   ** ** ** ** ** E7 00 42 00 22 00
2870   24 00 14 00 18 00 08 00 10 00
2880   10 00 E0 00 ** ** ** ** ** ** ** **
2890   ** ** ** **   zz zz zz zz zz zz

```

The `ww` bytes are the end of the bitmap for the "w" character, and the `zz` bytes are the beginning of the "z" bitmap. The `**` bytes are filler bytes that pad the "x" and "y" bitmaps to their fixed 30-byte size. All such filler bytes must be zero.

256-Character Font File Structure

Like 96-character font files, 256-character font files begin with two bytes that specify the bitmap width and height used in the font. However, the high order bit of the first (*MaxX*) byte is set for 256-character fonts. For instance, a *MaxX* value of 2 means this is a 96-character font with each character bitmap having a width of two bytes. But a *MaxX* of 82 hex indicates a 256-character font, also with each bitmap being two bytes wide.

In a 96-character font file, a *font block* follows the two size bytes. The font block contains the *Xsizes* table, the *Ysizes* table, and the bitmaps for all 96 characters. A 256-character font file has four such font blocks following the size bytes, with each font block containing data for 64 characters. That is, a 64-character font block contains a 64-byte *Xsizes* table, a 64-byte *Ysizes* table, followed by the bitmap data for 64 characters. The first font block contains data for ASCII characters 0 to 63, the second for 64 to 127, the third for 128 to 191, and the fourth for 192 to 255.

The following diagram shows the overall structure of a 256-character font file:

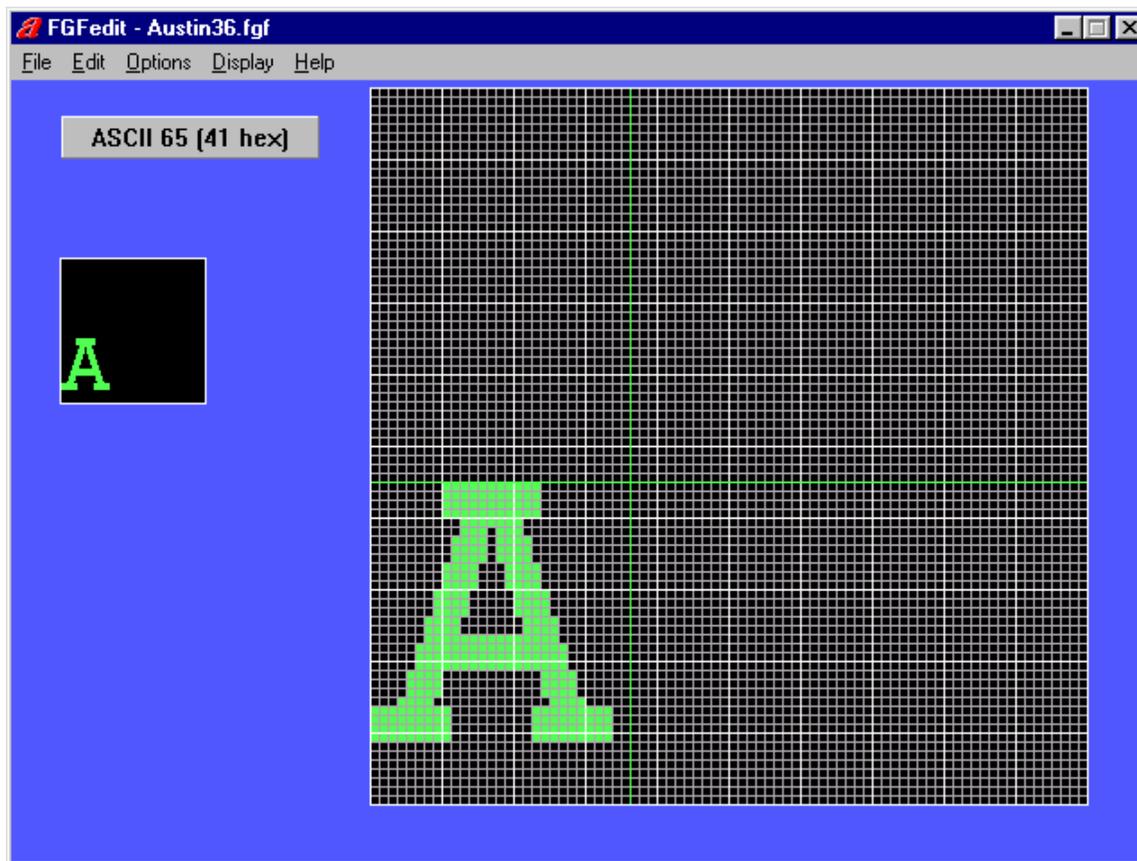
MaxX (1 byte with high order bit set)
MaxY (1 byte)
Font block 1 (characters 0-63)
Font block 2 (characters 64-127)
Font block 3 (characters 128-191)
Font block 4 (characters 192-255)

And each font block looks like this:

Xsizes table (64 bytes)
Ysizes table (64 bytes)
Bitmaps (64 * MaxX * MaxY bytes)

The FGFedit Utility

FGFedit is an interactive font editor and conversion utility. It provides a simple, intuitive way to create new FGF files, modify existing FGF files, and import TrueType (TTF) files and save them as FGF files. Complete information about FGFedit is available through the program's on-line help facility. The FGFedit main window looks like this:



FGFedit works with both 96-character and 256-character FGF files. When you edit an existing FGF file, the edited version will normally have the same number of characters as the original FGF file. For new FGF files, and when importing TTF files that will be saved as FGF files, FGFedit normally creates 256-character font files. You can change this behavior through the "Options|256-Character Font" menu selection.

The FGF2FGX Utility

The FGF2FGX utility is a Win32 console program that converts proportional font files (FGF files) to fixed pitch font files (FGX files). The width of the fixed pitch font is defined to be the width of the font's widest character. Please be aware that many proportional font files look fairly bad when converted to the fixed pitch format! The syntax of the command for running FGF2FGX from the DOS command prompt is

```
FGF2FGX fgf_file fgx_file
```

where *fgf_file* is the name of the proportionally spaced font file, and *fgx_file* is the name of the fixed pitch font file to create. FGF2FGX does not change the *fgf_file* in any way, but it will overwrite the *fgx_file* if an identically named file exists in the current directory.

The SCALFONT Utility

The SCALFONT utility is a Win32 console program that creates a new font file whose characters are smaller or larger than those of an existing font file. The syntax of the command for running SCALFONT from the DOS command prompt is

```
SCALFONT fgf_infile fgf_outfile factor [kerning]
```

where *fgf_infile* is the name of the original font file, *fgf_outfile* is the name of the resulting scaled font file, and *factor* is the scaling factor, expressed as a percentage of the original font size. For instance, a scaling factor of 50 will produce a font whose characters are half the size of the original, and a scaling factor of 120 would create a font whose characters are 20% larger than the original. The optional *kerning* parameter defines the number of blank pixels added to the right edge of each character; the default value is 1.

We do not recommend using SCALFONT on FGF files that were created by importing a TrueType font into FGFedit. You'll get better results by re-importing the TrueType font into FGFedit at a smaller or larger point size.

A

App.Path property, 10
AppPath(), 10
 automatic font loading, 9

B

Borland C++, 2

C

C#. *See* Visual C# .NET
 C++Builder, 2
 clipping, 9
 color change, 17
 compilation, 6
 compilers, 2
 current font, 9

D

Delphi, 2, 6
 DirectX, 6

E

examples
 FGFW1 (C/C++), 10
 FGFW1 (C++Builder), 12
 FGFW1 (Delphi), 14
 FGFW1 (VB), 16
 FGFW2 (C/C++), 18
 FGFW2 (C++Builder), 19
 FGFW2 (Delphi), 21
 FGFW2 (VB), 22
 FGFW3 (C/C++), 23
 FGFW3 (C++Builder), 25
 FGFW3 (Delphi), 27
 FGFW3 (VB), 28
 FGFW4 (C/C++), 29
 FGFW4 (C++Builder), 31
 FGFW4 (Delphi), 32
 FGFW4 (VB), 34
 FGFW5 (C/C++), 35
 FGFW5 (C++Builder), 37
 FGFW5 (Delphi), 38
 FGFW5 (VB), 40

FGFW6 (C/C++), 41
 FGFW6 (C++Builder), 43
 FGFW6 (Delphi), 44
 FGFW6 (VB), 46
 FGFW7 (C/C++), 47
 FGFW7 (C++Builder), 49
 FGFW7 (Delphi), 50
 FGFW7 (VB), 51

F

fg_fontdc(), 9
fg_move(), 10
fg_setclip(), 9
fg_setcolor(), 10, 17
fgf_define(), 9, 29, 35
fgf_drop(), 41
fgf_height(), 35
fgf_justify(), 9, 10
fgf_kerning(), 17
fgf_load(), 9, 10, 23, 29, 35
fgf_print(), 9, 10, 17, 23, 41, 47
fgf_printc(), 9, 17, 23, 41
fgf_select(), 23
fgf_undefine(), 35
fgf_under(), 17
fgf_unload(), 9, 35
fgf_width(), 35
 FGF2FGX, 58
 FGFedit, 56, 57, 58, 59
 FGFWin unit, 6
 FGFWin.bas, 6
 FGFWin.cs, 6
 FGFWIN.H, 7
 FGFWin.inc, 6
 FGFWin.vb, 6
 FGFWinD unit, 6
 FGFWinD.bas, 6
 FGFWinD.cs, 6
 FGFWinD.inc, 6
 FGFWinD.vb, 6
 fixed pitch fonts, 58
 font block, 57
 font file naming convention, 9
 font file structure, 56, 57
 font files, 9
 font handle, 9, 23
 FONTLIST file, 7
 ftp site, 3, 4

H

header files, 7
help file, 3

I

installation, 2

J

justification, 9, 10

K

kerning, 17, 56

L

linking, 6
loading fonts, 9, 10

M

maintenance updates, 4
manual font loading, 9, 35
Microsoft Visual Basic. *See* Visual Basic
Microsoft Visual Basic .NET. *See* Visual Basic .NET
Microsoft Visual C# .NET. *See* Visual C# .NET
Microsoft Visual C++. *See* Visual C++

N

naming conventions, 3

O

option prefix, 17

P

point size, 9
PowerBASIC, 2, 6
prerequisite knowledge, 2

R

releasing font handle, 35

S

SCALFONT, 58, 59
SETUP program, 2
string height, 35
string width, 35
Symantec C++, 2

T

technical support, 3, 4
TrueType files, 57

U

underlining, 17, 41
unloading fonts, 9

V

Visual Basic, 2, 6
Visual Basic .NET, 2, 6
Visual C# .NET, 2, 6
Visual C++, 2

W

Watcom C/C++, 2
web BBS, 3
web page, 3, 4
Win16, 2
Windows 3.x, 2