1 Preface

FireFly allows the use of controls that were written and contributed by mere mortals. But interfacing these controls to FireFLy has been a mystery due to the absence of documentation. I wanted to remedy this situation so I looked very carefully at custom controls that were created by others. I believe that I figured out most of the mystery, but not all.

Any help in improving this document would be greatly appreciated.

1.1 Author

Name Robert A. Rioja
e-mail RobRioja@gmail.com

1.2 With a little help from

Name: SeaVipe

1.3 Need Help

All text in **RED** should eventually be eliminated. Your help is needed to make this happen.

2 Table of Contents

1 Preface....1
1.1 Author....1
1.2 With a little help from....1
1.3 Need Help....1
2 Table of Contents....2
3 Folders and files....4
4 .ctl File Format....5
4.1 Title Line....5
4.2 Control Section....5
4.2.1 controlname....5
4.2.2 description....5
4.2.3 author....6
4.2.4 copyright....6
4.2.5 version....6
4.2.6 dll_filename....6
4.2.7 dll_required....6
4.2.8 sll_filename....6
4.2.9 sll_required....6
4.2.10 source_filename....6
4.2.11 toolbox_bitmap....6
4.2.12 toolbox_cursor....6
4.2.13 toolbox_tooltip....7
4.2.14 delimiter....7
4.2.15 uniqueid....7
4.2.16 use_loadlibrary....7
4.2.17 use_initialize_action....7
4.2.18 initialize_action_designer....7
4.2.19 initialize_action_code....7
4.2.20 property_count....7
4.2.21 message_count....7
4.2.22 allow_system_colors....8
4.2.23 isgroupbox....8
4.2.24 create_action....8
4.2.25 Other Control Section parameters....10
4.3 Property Section....11
4.3.1 name....11
4.3.2 curvalue....11
4.3.3 itemtype....11
4.4 Properties....12
4.4.1 Name....12
4.4.2 Styles....12
4.4.3 About....12
4.4.4 ControlIndex....12
4.4.5 Font....12
4.4.6 Height....13
4.4.7 Left....13

4.4.8 Locked....13
4.4.9 Top....13
4.4.10 Width....13
4.4.11 BackColor....14
4.4.12 Caption....14
4.4.13 ResizeRules....14
4.5 Messages Sections....15
4.5.1 Custom Functions....15
4.5.2 Windows Functions....15
5 Custom Control Creation....16
6 LOGFONT Structure....17
6.1 Members....17
6.1.1 lfHeight....17
6.1.2 lfWidth....18
6.1.3 lfEscapement....18
6.1.4 lfOrientation....18
6.1.5 lfWeight....18
6.1.6 lfItalic....18
6.1.7 lfUnderline....19
6.1.8 lfStrikeOut....19
6.1.9 lfCharSet....19
6.1.10 lfOutPrecision....20
6.1.11 lfClipPrecision....20
6.1.12 lfQuality....21
6.1.13 lfPitchAndFamily....22
6.1.14 lfFaceName....22

3 Folders and files

FireFly custom controls are stored in a folder called **CustomControls** within the installation folder. It contains sub-folders named after the authors of the individual custom controls. I call these folders **Author Folders**. Each of these sub-folders contain sub-sub-folders, one for each custom control written by an author. I call these folders **Control Folders**.

Each **Control Folder** contains the files needed by a custom control. Some of these files are absolutely required. Others are optional. For example, let's say that there is a control called **YourControl**. Its folder will contain the required:

```
C_YourControl.cur
N_YourControl.bmp
YourControl.ctl
YourControl.dll      or     YourControl.sll
YourControl.inc
```

The **.cur** file is a graphic, in standard Windows cursor format. It contains the image that will appear in the **Tools** tab of the **FireFly Workspace** window, where all of the controls are listed. The standard name of this file is "**C_**", followed by the **name** of your control, followed by "**.cur**".

The **.bmp** file is a graphic, in standard Windows bitmap format. It contains the same image as the cursor file. The standard name of this file is "**N_**", followed by the **name** of your control, followed by "**.bmp**". **I do not know the purpose for the bitmap file.**

The **.ctl** file is a text file which gives FireFly all the information it needs to create and manage the control. Its format is described in another section. The standard name of this file is the **name** of your control, followed by "**.ctl**".

The **.dll** file contains the compiled code that you wrote to make your control work. It is typically written in PowerBasic and compiled into a dynamically linked library. The standard name of this file is the **name** of your control, followed by "**.dll**".

Alternately, you can compile your code into a statically linked library. The standard name of this file is the **name** of your control, followed by "**.sll**".

The **.inc** file is a PowerBasic source file which can provide your FireFly program with data it will need to know to use your control. This might include defined constants, etc. The standard name of this file is the **name** of your control, followed by "**.inc**". This file could also contain all of your control's code, so that neither a DLL nor a SLL would be needed.

Other files which might be needed by your control should also be placed in this **Control Folder**.
Here we could use a description of other types of files.

4 .ctl File Format

The .ctl file is a text file that can be created with any text editor. However, its contents, organization and format should be followed as described here.

The file contains a title line followed sections which are named by specific section names enclosed in brackets. Blank lines are used to separate the sections only to make the text more readable. Each section contains parameters which are formatted as:

```
   parameter = value
```

where `parameter` is the name of a parameter and `value` is the value assigned to that parameter. Each parameter name is predefined.

4.1 Title Line

The first line is not a named section and always contains the following text:

```
#FireFly_Custom_Control#
```

4.2 Control Section

The Control Section is the first section. There can only be one Control Section. It contains general information about the control. Here is an example:

```
[Control]
controlname   = MyControl
description   = This control is a custom thingymagiggy.
author        = John Doe
...
```

Here is a list of control section parameters:

4.2.1 controlname

Name of the control. It will also be shown in the About box if your control has one. Example:
`controlname = MyControl`

4.2.2 description

A short description of the control. It will also be shown in the About box if your control has one. Example:
`description = This control is a custom thingymagiggy.`

4.2.3 author

Your name. It will also be shown in the About box if your control has one. Example:
```
author = John Doe
```

4.2.4 copyright

Your copyright message. It will also be shown in the About box if your control has one. Example:
```
copyright = John Doe, All Rights Reserved 2016
```

4.2.5 version

Your version number. It will also be shown in the About box if your control has one. Example:
```
version - 1.23
```

4.2.6 dll_filename

Name of your .dll file. It will also be shown in the About box if your control has one. Example:
```
dll_filename = MyControl.dll
```

4.2.7 dll_required

1 means that the control's code is in a DLL file. 0 means that a DLLis not required. Example:
```
dll_required = 1
```

4.2.8 sll_filename

Name of your .sll file. It will also be shown in the About box if your control has one. Example:
```
sll_filename = MyControl.sll
```

4.2.9 sll_required

1 means that the control's code is in a SLL file. 0 means that a SLLis not required. Example:
```
sll_required = 1
```

4.2.10 source_filename

Name of your .inc file. It will also be shown in the About box if your control has one. Example:
```
source_filename = MyControl.inc
```

4.2.11 toolbox_bitmap

Name of your .bmp file. Example:
```
toolbox_bitmap = N_MyControl.bmp
```

4.2.12 toolbox_cursor

Name of your .cur file. Example:

```
toolbox_cursor = C_MyControl.cur
```

4.2.13 toolbox_tooltip

Short description of your control. **Not sure where this is used.** Example:
```
toolbox_tooltip = My custom control
```

4.2.14 delimiter

Character used as delimiter for later parameters that require more than one value. Example:
```
delimiter = |
```

4.2.15 uniqueid

A unique identifier shown in braces. Shown in About. **I don't know why this is needed, nor how to obtain it.** Example:
```
uniqueid = {12345678-1234}
```

4.2.16 use_loadlibrary

Value of 0 or 1. **I don't know what this is for.** Example:
```
use_loadlibrary = 1
```

4.2.17 use_initialize_action

Value of 0 or 1. **I don't know what this is for.** Example:
```
use_initialize_action = 0
```

4.2.18 initialize_action_designer

I don't know what this is for. Example:
```
initialize_action designer =
```

4.2.19 initialize_action_code

I don't know what this is for. Example:
```
initialize_action code =
```

4.2.20 property_count

I don't know what this is for. Example:
```
property_count = 10
```

4.2.21 message_count

I don't know what this is for.
```
message_count = 10
```

4.2.22 allow_system_colors

Value of 0 or 1. **I don't know what this is for.** Example:
`allow_system_colors = 0`

4.2.23 isgroupbox

Value of 0 or 1. **I don't know what this is for.** Example:
`isgroupbox = 0`

4.2.24 create_action

Specifies a function (and its parameters) which tells Windows to create your control. It's value is a string containing several parameters separated by the delimiter specified by `delimiter`. An example is:

```
USER32.DLL|CreateWindowEx|CreateWindowExA|BYVAL LONG //CTRL_EXSTYLE//|
BYVAL ASCIIZ MYCONTROL|BYVAL ASCIIZ //CTRL_CAPTION//|BYVAL LONG
//CTRL_STYLE//|BYVAL LONG //CTRL_LEFT//|BYVAL LONG //CTRL_TOP//|BYVAL
LONG //CTRL_WIDTH//|BYVAL LONG //CTRL_HEIGHT//|BYVAL LONG
//CTRL_PARENT//|BYVAL LONG //CTRL_ID//|BYVAL LONG //CTRL_INST//|BYVAL
ANY %NULL
```

The string does not contain any actual data. It just has place holders for the data. It is interpreted as follows:

Parameter	Description
`USER32.DLL`	Name of DLL containing the function which will create the control.
`CreateWindowEx`	Name of function which will create the control.
`CreateWindowExA`	Name of function which will create the control. **I don't know why we need this twice.**
`BYVAL LONG //CTRL_EXSTYLE//`	Extended style flags.
`BYVAL ASCIIZ MYCONTROL`	**Explain?**
`BYVAL ASCIIZ //CTRL_CAPTION//`	**Explain?**
`BYVAL LONG //CTRL_STYLE//`	Style flags.
`BYVAL LONG //CTRL_LEFT//`	Left property.
`BYVAL LONG //CTRL_TOP//`	Top property.
`BYVAL LONG //CTRL_WIDTH//`	Width property.
`BYVAL LONG //CTRL_HEIGHT//`	Height property.
`BYVAL LONG //CTRL_PARENT//`	Windows handle for the control's parent, usually the form that the control is in.
`BYVAL LONG //CTRL_ID//`	Window handle for the control.

Parameter	Description
BYVAL LONG //CTRL_INST//	**Explain?**
BYVAL ANY %NULL	**Explain?**

Another example:

```
MYCONTROL.DLL|MYCONTROL_CREATE|MYCONTROL_CREATE|BYVAL LONG
//CTRL_PARENT//|BYVAL LONG //CTRL_ID//|BYVAL LONG //CTRL_LEFT//|BYVAL
LONG //CTRL_TOP//|BYVAL LONG //CTRL_WIDTH//|BYVAL LONG //CTRL_HEIGHT//|
BYVAL LONG //CTRL_STYLE//|BYVAL LONG //CTRL_EXSTYLE//
```

Parameter	Description
MYCONTROL.DLL	Name of DLL containing the function which will create the control.
MYCONTROL_CREATE	Name of function which will create the control.
MYCONTROL_CREATE	Name of function which will create the control. **I don't know why we need this twice.**
BYVAL LONG //CTRL_PARENT//	Windows handle for the control's parent, usually the form that the control is in.
BYVAL LONG //CTRL_ID//	Window handle for the control.
BYVAL LONG //CTRL_LEFT//	Left property.
BYVAL LONG //CTRL_TOP//	Top property.
BYVAL LONG //CTRL_WIDTH//	Width property.
BYVAL LONG //CTRL_HEIGHT//	Height property.
BYVAL LONG //CTRL_STYLE//	Style flags.
BYVAL LONG //CTRL_EXSTYLE//	Extended style flags.

```
CreateWindow(
   lpClassName  as LPCTSTR, ' pointer to registered class name
   lpWindowName as LPCTSTR, ' pointer to window name
   dwStyle      as DWORD,   ' window style
   x            as integer, ' horizontal position of window
   y            as integer, ' vertical position of window
   nWidth       as integer, ' window width
   nHeight      as integer, ' window height
```

```
   hWndParent   as HWND,    ' handle to parent or owner window
   hMenu        as HMENU,   ' handle to menu or child-window identifier
   hInstance    as HANDLE,  ' handle to application instance
   lpParam      as LPVOID   ' pointer to window-creation data
   ) as DWORD
```

4.2.25 Other Control Section parameters

There may be other Control Section parameters that I do not know about.

4.3 Property Section

The **Properties** tab of the **FireFly Workspace** window shows the properties of each control. You can specify which of your control's properties are available by defining them in Properties Sections. You can have as many Properties Sections as there are properties in your control. For each property, you must specify its name, current value and type. For certain properties, you will also have to specify other parameters. Examples:

```
[Property]
name        = name|(Name)
curvalue    = mycMyControl
itemtype    = Edit|1

[Property]
name        = windowstyles|(WindowStyles)
curvalue    =
itemtype    = Styles

[Property]
name        = locked|Locked
curvalue    = False
itemtype    = Combo
cmbitems    = False|True
equates     = False|True
```

4.3.1 name

The first parameter contains two delimited values. The first specifies the property. The second specifies the text that will appear in the first column of the **Properties** tab in the **FireFly Workspace.** Example:
`name = name|(Name)`
The name of the property is `name`. **This is confusing but I don't know a better way to explain it.** The text that will be shown is `(Name)`.

Another example:
`name = windowstyles|(WindowStyles)`
The name of the property is `windowstyles`, the text displayed is `(WindowStyles)`.

4.3.2 curvalue

Current (or default) value of the property. Example:
`curvalue = mycMyControl`

4.3.3 itemtype

This parameter defines the type of interface between the user and the property. It may contain more than one value, separated by the delimiter. Example:

```
itemtype = Edit|1
```
This is an editable text box. **I don't know what the second value does.**

4.4 Properties

The following are examples of common properties:

4.4.1 Name

Control name. This property uses an editable text box.

```
[Property]
name        = name|(Name)
curvalue    = mycMyControl
itemtype    = Edit|1
```

4.4.2 Styles

Window styles. This property uses a pop-up window full of check boxes.

```
[Property]
name        = windowstyles|(WindowStyles)
curvalue    =
itemtype    = Styles
```

4.4.3 About

About box. This shows the user some information about the control in a pop-up window.

```
[Property]
name        = about|About
curvalue    =
itemtype    = About
```

4.4.4 ControlIndex

Allows user to enter an index for a control array member.

```
[Property]
name        = controlindex|ControlIndex
curvalue    = 0
itemtype    = Edit|1
```

4.4.5 Font

Control's font. This property uses the Windows standard font selection dialog. For an explanation of the parameters, see section 6.

```
[Property]
name        = font|Font
curvalue    = Tahoma,-13,0,0,0,400,0,0,0,0,3,2,1,34
itemtype    = Font
```

4.4.6 Height

Control's height.

```
[Property]
name        = height|Height
curvalue    =
itemtype    = Edit|True
```

4.4.7 Left

Control's left coordinate.

```
[Property]
name        = left|Left
curvalue    =
itemtype    = Edit|1
```

4.4.8 Locked

Control's lock. This property uses a combo box. Therefore the `cmbitems` and `equates` parameters are used.

```
[Property]
name        = locked|Locked
curvalue    = False
itemtype    = Combo
cmbitems    = False|True
equates     = False|True
```

4.4.9 Top

Control's left coordinate.

```
[Property]
name       = top|Top
curvalue    =
itemtype    = Edit|1
```

4.4.10 Width

Control's width

```
[Property]
name        = width|Width
curvalue    =
itemtype    = Edit|1
```

4.4.11 BackColor

Control's back color. **Need description of prop_action.**

```
[Property]
name          = backstyle|BackStyle
curvalue      = 0 - Transparent
itemtype      = Combo
cmbitems      = 0 - Transparent|1 - Opaque
equates       = 1|2
prop_action   = USER32.DLL|SendMessage|SendMessageA|BYVAL LONG
//CTRL_HWND//|BYVAL LONG 1028|BYVAL LONG //PROP_VALUE//|BYVAL LONG 1
```

4.4.12 Caption

Control's caption.

```
[Property]
name        = caption|Caption
curvalue    = RRButton
itemtype    = Edit|0
```

4.4.13 ResizeRules

Set control's resizing rules.

```
[Property]
name        = resizerules|ResizeRules
curvalue    = 0
itemtype    = ResizeRules
```

4.5 Messages Sections

Your control will have callback functions (event handlers) that the user will need to have access to. There should be one Message Section for each such function, describing the function's interface. Standard Windows functions can also be specified, although their interfaces are already defined and need not be described.

4.5.1 Custom Functions

For example, let's assume that your control has a function like this:

```
MyControl_Clicked ( _
                  ControlIndex As Long,
                  hWndForm As Dword,
                  hWndControl As Dword,
                  ByVal lpButon As Long
                  ) As Long
```

The Message Section might then look like this:

```
[Message]
name         = MyControl_Clicked
declare      = (ControlIndex As Long, hWndForm As Dword, hWndControl As Dword, BYVAL lpButon As long) As Long
call         = FLY_nResult = //MESSAGE// (FLY_ControlIndex, hWndForm, @FLY_pNotify.hwndFrom, lParam)
notification = Notify_Notification
```

4.5.2 Windows Functions

If you are using a Windows function like `WM_SIZE`, then its Message Section would look like this:

```
[Message]
name = WM_SIZE
```

5 Custom Control Creation

Looking for ideas.

6 LOGFONT Structure

This section was excerpted from the Microsoft MSDN web site. You will need this information to specify fonts, as in section 4.4.5.

The LOGFONT structure defines the attributes of a font. The C++ syntax is as follows:

```
typedef struct tagLOGFONT {
  LONG      lfHeight;
  LONG      lfWidth;
  LONG      lfEscapement;
  LONG      lfOrientation;
  LONG      lfWeight;
  BYTE      lfItalic;
  BYTE      lfUnderline;
  BYTE      lfStrikeOut;
  BYTE      lfCharSet;
  BYTE      lfOutPrecision;
  BYTE      lfClipPrecision;
  BYTE      lfQuality;
  BYTE      lfPitchAndFamily;
  TCHAR  lfFaceName[LF_FACESIZE];
} LOGFONT, *PLOGFONT;
```

6.1 Members

6.1.1 lfHeight

The height, in logical units, of the font's character cell or character. The character height value (also known as the em height) is the character cell height value minus the internal-leading value. The font mapper interprets the value specified in **lfHeight** in the following manner.

Value	Meaning
> 0	The font mapper transforms this value into device units and matches it against the cell height of the available fonts.
0	The font mapper uses a default height value when it searches for a match.
< 0	The font mapper transforms this value into device units and matches its absolute value against the character height of the available fonts.

For all height comparisons, the font mapper looks for the largest font that does not exceed the requested size. This mapping occurs when the font is used for the first time. For the MM_TEXT mapping mode, you can use the following formula to specify a height for a font with a specified point size:

C++ syntax:

```
lfHeight = -MulDiv(PointSize, GetDeviceCaps(hDC, LOGPIXELSY), 72);
```

6.1.2 lfWidth

The average width, in logical units, of characters in the font. If **lfWidth** is zero, the aspect ratio of the device is matched against the digitization aspect ratio of the available fonts to find the closest match, determined by the absolute value of the difference.

6.1.3 lfEscapement

The angle, in tenths of degrees, between the escapement vector and the x-axis of the device. The escapement vector is parallel to the base line of a row of text. When the graphics mode is set to GM_ADVANCED, you can specify the escapement angle of the string independently of the orientation angle of the string's characters. When the graphics mode is set to GM_COMPATIBLE, **lfEscapement** specifies both the escapement and orientation. You should set **lfEscapement** and **lfOrientation** to the same value.

6.1.4 lfOrientation

The angle, in tenths of degrees, between each character's base line and the x-axis of the device.

6.1.5 lfWeight

The weight of the font in the range 0 through 1000. For example, 400 is normal and 700 is bold. If this value is zero, a default weight is used. The following values are defined for convenience.

Value	Weight
FW_DONTCARE	0
FW_THIN	100
FW_EXTRALIGHT	200
FW_ULTRALIGHT	200
FW_LIGHT	300
FW_NORMAL	400
FW_REGULAR	400
FW_MEDIUM	500
FW_SEMIBOLD	600
FW_DEMIBOLD	600
FW_BOLD	700
FW_EXTRABOLD	800
FW_ULTRABOLD	800
FW_HEAVY	900
FW_BLACK	900

6.1.6 lfItalic

An italic font if set to **TRUE**.

6.1.7 lfUnderline

An underlined font if set to **TRUE**.

6.1.8 lfStrikeOut

A strikeout font if set to **TRUE**.

6.1.9 lfCharSet

The character set. The following values are predefined.

```
ANSI_CHARSET
BALTIC_CHARSET
CHINESEBIG5_CHARSET
DEFAULT_CHARSET
EASTEUROPE_CHARSET
GB2312_CHARSET
GREEK_CHARSET
HANGUL_CHARSET
MAC_CHARSET
OEM_CHARSET
RUSSIAN_CHARSET
SHIFTJIS_CHARSET
SYMBOL_CHARSET
TURKISH_CHARSET
VIETNAMESE_CHARSET
```

Korean language edition of Windows:

```
JOHAB_CHARSET
```

Middle East language edition of Windows:

```
ARABIC_CHARSET
HEBREW_CHARSET
```

Thai language edition of Windows:

```
THAI_CHARSET
```

The OEM_CHARSET value specifies a character set that is operating-system dependent. DEFAULT_CHARSET is set to a value based on the current system locale. For example, when the system locale is English (United States), it is set as ANSI_CHARSET. Fonts with other character sets may exist in the operating system. If an application uses a font with an unknown character set, it should not attempt to translate or interpret strings that are rendered with that font. This parameter is important in the font mapping process. To ensure consistent results, specify a specific character set. If you specify a typeface name in the **lfFaceName** member, make sure that

the **lfCharSet** value matches the character set of the typeface specified in **lfFaceName**.

6.1.10 lfOutPrecision

The output precision. The output precision defines how closely the output must match the requested font's height, width, character orientation, escapement, pitch, and font type. It can be one of the following values.

Value	Meaning
OUT_CHARACTER_PRECIS	Not used.
OUT_DEFAULT_PRECIS	Specifies the default font mapper behavior.
OUT_DEVICE_PRECIS	Instructs the font mapper to choose a Device font when the system contains multiple fonts with the same name.
OUT_OUTLINE_PRECIS	This value instructs the font mapper to choose from TrueType and other outline-based fonts.
OUT_PS_ONLY_PRECIS	Instructs the font mapper to choose from only PostScript fonts. If there are no PostScript fonts installed in the system, the font mapper returns to default behavior.
OUT_RASTER_PRECIS	Instructs the font mapper to choose a raster font when the system contains multiple fonts with the same name.
OUT_STRING_PRECIS	This value is not used by the font mapper, but it is returned when raster fonts are enumerated.
OUT_STROKE_PRECIS	This value is not used by the font mapper, but it is returned when TrueType, other outline-based fonts, and vector fonts are enumerated.
OUT_TT_ONLY_PRECIS	Instructs the font mapper to choose from only TrueType fonts. If there are no TrueType fonts installed in the system, the font mapper returns to default behavior.
OUT_TT_PRECIS	Instructs the font mapper to choose a TrueType font when the system contains multiple fonts with the same name.

Applications can use the OUT_DEVICE_PRECIS, OUT_RASTER_PRECIS, OUT_TT_PRECIS, and OUT_PS_ONLY_PRECIS values to control how the font mapper chooses a font when the operating system contains more than one font with a specified name. For example, if an operating system contains a font named Symbol in raster and TrueType form, specifying OUT_TT_PRECIS forces the font mapper to choose the TrueType version. Specifying OUT_TT_ONLY_PRECIS forces the font mapper to choose a TrueType font, even if it must substitute a TrueType font of another name.

6.1.11 lfClipPrecision

The clipping precision. The clipping precision defines how to clip characters that are partially outside the clipping region. It can be one or more of the following values. For more information about the orientation of coordinate systems, see the description of the *nOrientation* parameter.

Value	Meaning

CLIP_CHARACTER_PRECIS	Not used.
CLIP_DEFAULT_PRECIS	Specifies default clipping behavior.
CLIP_DFA_DISABLE	**Windows XP SP1:** Turns off font association for the font. Note that this flag is not guaranteed to have any effect on any platform after Windows Server 2003.
CLIP_EMBEDDED	You must specify this flag to use an embedded read-only font.
CLIP_LH_ANGLES	When this value is used, the rotation for all fonts depends on whether the orientation of the coordinate system is left-handed or right-handed. If not used, device fonts always rotate counterclockwise, but the rotation of other fonts is dependent on the orientation of the coordinate system.
CLIP_MASK	Not used.
CLIP_DFA_OVERRIDE	Turns off font association for the font. This is identical to CLIP_DFA_DISABLE, but it can have problems in some situations; the recommended flag to use is CLIP_DFA_DISABLE.
CLIP_STROKE_PRECIS	Not used by the font mapper, but is returned when raster, vector, or TrueType fonts are enumerated. For compatibility, this value is always returned when enumerating fonts.
CLIP_TT_ALWAYS	Not used.

6.1.12 lfQuality

The output quality. The output quality defines how carefully the graphics device interface (GDI) must attempt to match the logical-font attributes to those of an actual physical font. It can be one of the following values.

Value	**Meaning**
ANTIALIASED_QUALITY	Font is always antialiased if the font supports it and the size of the font is not too small or too large.
CLEARTYPE_QUALITY	If set, text is rendered (when possible) using ClearType antialiasing method. See Remarks for more information.
DEFAULT_QUALITY	Appearance of the font does not matter.
DRAFT_QUALITY	Appearance of the font is less important than when PROOF_QUALITY is used. For GDI raster fonts, scaling is enabled, which means that more font sizes are available, but the quality may be lower. Bold, italic, underline, and strikeout fonts are synthesized if necessary.
NONANTIALIASED_QUALITY	Font is never antialiased.
PROOF_QUALITY	Character quality of the font is more important than exact matching of the logical-font attributes. For GDI raster fonts, scaling is disabled and the font closest in size is chosen. Although the chosen font size may not be mapped exactly when PROOF_QUALITY is used, the quality of the font is high and there is no distortion of appearance. Bold, italic, underline, and strikeout fonts are synthesized if necessary.

If neither ANTIALIASED_QUALITY nor NONANTIALIASED_QUALITY is selected, the font is antialiased only if the user chooses smooth screen fonts in Control Panel.

6.1.13 lfPitchAndFamily

The pitch and family of the font. The two low-order bits specify the pitch of the font and can be one of the following values.

- DEFAULT_PITCH
- FIXED_PITCH
- VARIABLE_PITCH

Bits 4 through 7 of the member specify the font family and can be one of the following values.

- FF_DECORATIVE
- FF_DONTCARE
- FF_MODERN
- FF_ROMAN
- FF_SCRIPT
- FF_SWISS

The proper value can be obtained by using the Boolean OR operator to join one pitch constant with one family constant.

Font families describe the look of a font in a general way. They are intended for specifying fonts when the exact typeface desired is not available. The values for font families are as follows.

Value	**Meaning**
FF_DECORATIVE	Novelty fonts. Old English is an example.
FF_DONTCARE	Use default font.
FF_MODERN	Fonts with constant stroke width (monospace), with or without serifs. Monospace fonts are usually modern. Pica, Elite, and CourierNew are examples.
FF_ROMAN	Fonts with variable stroke width (proportional) and with serifs. MS Serif is an example.
FF_SCRIPT	Fonts designed to look like handwriting. Script and Cursive are examples.
FF_SWISS	Fonts with variable stroke width (proportional) and without serifs. MS Sans Serif is an example.

6.1.14 lfFaceName

A null-terminated string that specifies the typeface name of the font. The length of this string must not exceed 32 **TCHAR** values, including the terminating **NULL**. The EnumFontFamiliesEx function can be used to enumerate the typeface names of all currently available fonts. If **lfFaceName** is an empty string, GDI uses the first font that matches the other specified attributes.