Font Creation with FontForge

George Williams
444 Alan Rd.
Santa Barbara, Ca. 93109, USA
gww@silcom.com
http://bibliofile.duhs.duke.edu/gww/

Abstract

FontForge is an open source program which allows the the creation and modification of fonts in many standard formats. This talk will start with the basic problem of converting a picture of a letter into an outline image – I shall also address the issue of converting the output of MetaFont into a PostScript font. Then I shall describe the automatic creation of accented characters, and how to add ligatures and kerning pairs to a font. Finally I shall present a few tools for detecting common problems in font design.

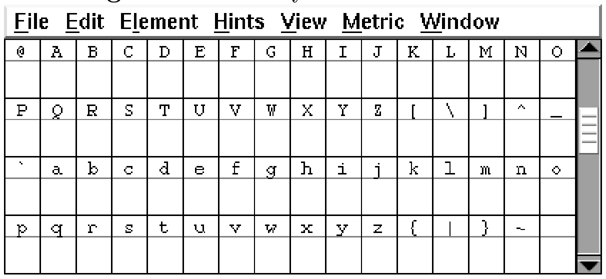
Font creation

You may create an empty font either by invoking FontForge with the `-new` argument on the command line

```
    $ fontforge -new
```

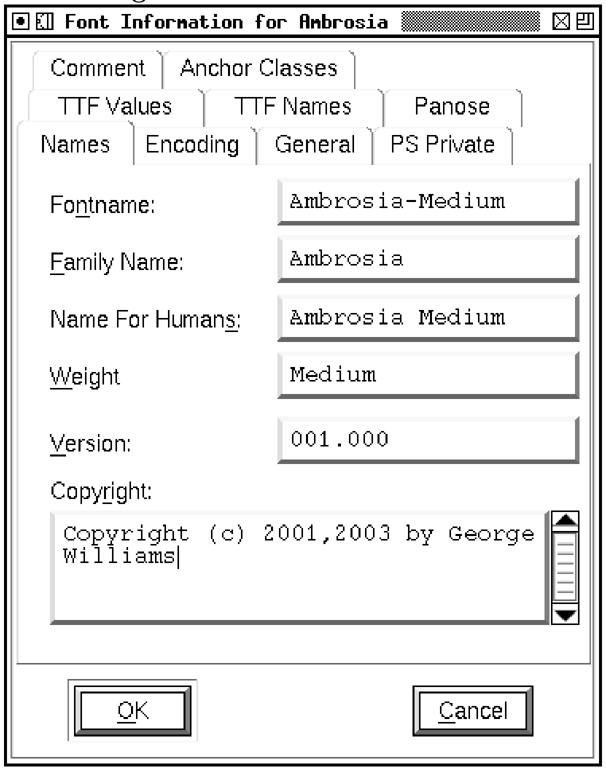
or by invoking the `New` item from the `File` menu. In either case you should end up with a window like this:

Figure 1: A newly created blank font


File Edit Element Hints View Metric Window
@ A B C D E F G H I J K L M N O
P Q R S T U V W X Y Z [\ ] ^ _
` a b c d e f g h i j k l m n o
p q r s t u v w x y z { | } ~

Such a font will have no useful name as yet, and will be encoded with the default encoding (usually Latin1). Use the `Element -> Font Info` menu item to correct these deficiencies. This dialog has several tabbed sub-dialogs, the first one allows you to set the font's various names.

Figure 2: Font name information



Fontname: Ambrosia-Medium
Family Name: Ambrosia
Name For Humans: Ambrosia Medium
Weight: Medium
Version: 001.000
Copyright: Copyright (c) 2001,2003 by George Williams

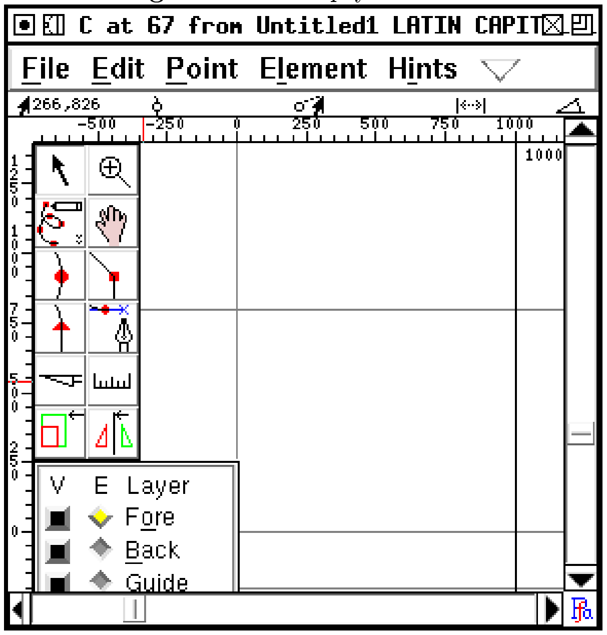
- the family name (most fonts are part of a family of similar fonts)
- the font name, a name for PostScript, usually containing the family name and any style modifiers
- and finally a name that is meaningful to humans

If you wish to change the encoding (to TEX Base or Adobe Standard perhaps) the Encoding tab will present you with a pulldown list of known encodings. If you are making a TrueType font then you should also go to the `General` tab and select an em-size of 2048 (the default coordinate system for TrueType is a little different from that of PostScript).

Character creation

Once you have done that you are ready to start editing characters; for the sake of example, let's create a capital 'C'. Double click on the entry for "C" in the font view above. You should now have an empty Outline Character window:

Figure 3: An empty character


C at 67 from Untitled1 LATIN CAPIT
File Edit Point Element Hints
266,826
-500 -250 0 250 500 750 1000
1000
V E Layer
Fore
Back
Guide

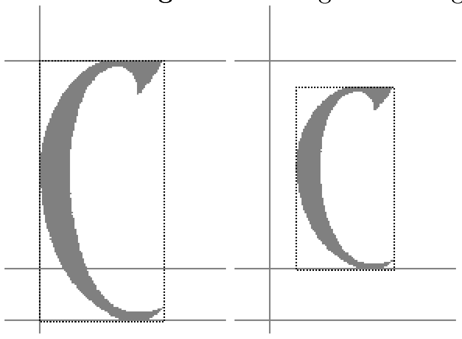
The outline character window contains two palettes snuggled up on the left side of the window. The top palette contains a set of editing tools, and the bottom palette controls which layers of the window are visible or editable.

The foreground layer contains the outline that will become part of the font. The background layer can contain images or line drawings that help you draw this particular character. The guide layer contains lines that are useful on a font-wide basis (such as a line at the x-height). Currently all layers are empty.

This window also shows the character's internal coordinate system with the x and y axes drawn in light grey. A line representing the character's advance width is drawn in black at the right edge of the window. FontForge assigns a default advance width of one em (in PostScript that will usually be 1000 units) to the advance width of a new character.

Select the `File -> Import` menu command to import an image of the character you are creating, assuming that you have one. It will be scaled so that it is as high as the em-square. In this case that's too big and we must rescale the image.

Figure 4: Background image

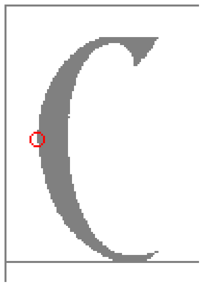


Make the background layer editable (by selecting the `Back` checkbox in the layers palette), move the mouse pointer to one of the edges of the image, hold down the shift key (to constrain the rescale to the same proportion in both dimensions), depress and drag the corner until the image is a reasonable size. Next move the mouse pointer onto the dark part of the image, depress the mouse and drag the image to the correct position.

If you have downloaded the autotrace program you can invoke `Element -> AutoTrace` to generate an outline from the image (You should probably follow this by `Element -> Add Extrema` and `Element -> Simplify`). But I suggest you trace the character yourself (results will be better).

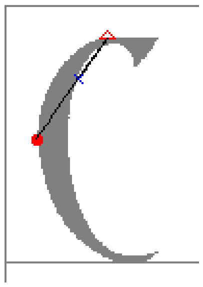
Change the active layer back to foreground (in the layers palette), and select the curve point tool from the tools palette. Then move the pointer to the edge of the image and add a point. I find that it is best to add points at places where the curve is horizontal or vertical, at corners, or where the curve changes inflection (A change of inflection occurs in a curve like "S" where the curve changes from being open to the left to being open on the right. If you follow these rules hinting will work better.

Figure 5: Tracing 1



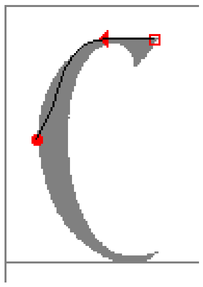
It is best to enter a curve in a clockwise fashion, so the next point should be added up at the top of the image on the flat section. Because the shape becomes flat here, a curve point is not appropriate, rather a tangent point is (this looks like a little triangle on the tools palette). A tangent point makes a nice transition from curves to straight lines because the curve leaves the point with the same slope the line had when it entered.

Figure 6: Tracing 2



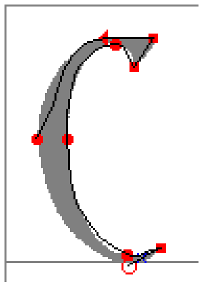
At the moment this "curve" doesn't match the image at all, don't worry about that we'll fix it later, and anyway it will change on its own as we continue. Note that we now have a control point attached to the tangent point (the little blue x). The next point needs to go where the image changes direction abruptly. Neither a curve nor a tangent point is appropriate here, instead we must use a corner point (one of the little squares on the tools palette).

Figure 7: Tracing 3



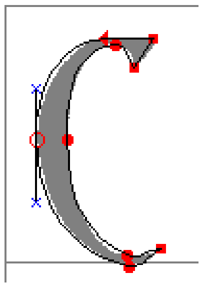
As you see the curve now starts to follow the image a bit more closely. We continue adding points until we are ready to close the path.

Figure 8: Tracing 4



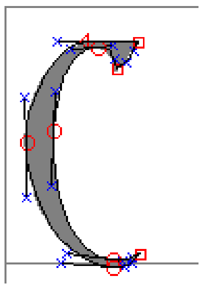
Then we close the path just by adding a new point on top of the old start point

Figure 9: Tracing 5



Now we must make the curve track the image more closely, to do this we must adjust the control points (the blue "x"es). To make all the control points visible select the pointer tool and double-click on the curve. Then move the control points around until the curve looks right.

Figure 10: Tracing 6



Finally we set the advance width. Again with the pointer tool, move the mouse to the width line on the right edge of the screen, depress and drag the line back to a reasonable location.

And we are done with this character.

Navigating to characters

The font view provides one way of navigating around the characters in a font. Simple scroll around it until you find the character you need and then double click on it to open a window looking at that character.

Typing a character will move to that character.

But some fonts are huge (Chinese, Japanese and Korean fonts have thousands or even tens of thousands of characters) and scrolling around the font view is a an inefficient way of finding your character. `View->Goto` provides a simple dialog which will allow you to move directly to any character for which you know the name (or encoding). If your font is a Unicode font, then this dialog will also allow you to find characters by block name (e.g. There is a pulldown list from which you may select Hebrew rather than Alef).

The simplest way to navigate is just to go to the next or previous character. And `View->Next Char` and `View->Prev Char` will do exactly that.

Loading background images better

In the background image of the previous example the bitmap of the letter filled the canvas of the image (with no white borders around it). When FontForge imported the image it needed to be scaled once in the program. But usually when you create the image of the letter you have some idea of how much white space there should be around it. If your images are exactly one em high then FontForge will automatically scale them to be the right size. So in the following examples all the images have exactly the right amount of white space around them to fit perfectly in an em.

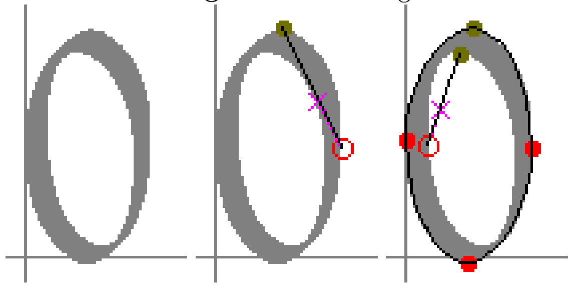
FontForge also has the ability to import an entire bitmap font (for example a "pk" or "gf" font produced by MetaFont) to provide properly scaled background images for all characters in a font. FontForge can even deal with "mf" files to some extent – it will invoke METAFONT on the file to generate a "gf" file and then import that into the background.

Creating the letter "o" – consistent directions

Let us turn our attention to the letter "o" which has a hole (or counter) in the middle. Open the outline view for the letter "o" and import a background image into it.

Notice that the first outline is drawn clockwise and the second counter-clockwise. This change in drawing direction is important. Both PostScript and TrueType require that the outer boundary of a character be drawn in a certain direction (they happen to be opposite from each other, which is a mild annoyance), within FontForge all outer boundaries must be drawn clockwise, while all inner boundaries must be drawn counter-clockwise.

Figure 11: Tracing o



If you fail to alternate directions between outer and inner boundaries you may get results like the one on the left . If you fail to draw the outer contour in a clockwise fashion the errors are more subtle, but will generally result in a less pleasing result once the character has been rasterized.

TECHNICAL AND CONFUSING: the exact behavior of rasterizers varies. Early PostScript rasterizers used a "non-zero winding number rule" while more recent ones use an "even-odd" rule. TrueType uses the "non-zero" rule. The example given above is for the "non-zero" rule. The "even-odd" rule would fill the "o" correctly no matter which way the paths were drawn (though there would probably be subtle problems with hinting).

To determine whether a pixel should be set using the even-odd rule draw a line from that pixel to infinity (in any direction) and count the number of contour crossings. If this number is even the pixel is not filled. If the number is odd the pixel is filled. Using the non-zero winding number rule the same line is drawn, contour crossings in a clockwise direction add 1 to the crossing count, while counter-clockwise contours subtract 1. If the result is 0 the pixel is not filled, any other result will fill it.

The command `Element->Correct Direction` will look at each selected contour, figure out whether it qualifies as an outer or inner contour and will reverse the drawing direction when the contour is drawn incorrectly.

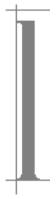
Creating letters with consistent stem widths, serifs and heights.

Many Latin (Greek, Cyrillic) fonts have serifs, special terminators at the end of stems. And in almost all LGC fonts there should only be a small number of stem widths used (ie. the vertical stem of "l" and "i" should probably be the same).

FontForge does not have a good way to enforce consistency, but it does have various commands to help you check for it, and to find discrepancies.

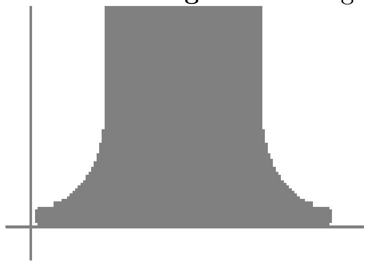
Let us start with the letter "l" and go through the familiar process of importing a bitmap and defining it's outline.

Figure 12: Beginning “l”



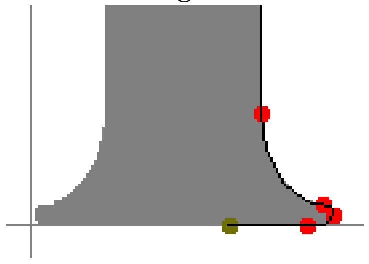
Use the magnify tool to examine the bottom serif, and note that it is symmetric left to right.

Figure 13: Magnified “l”



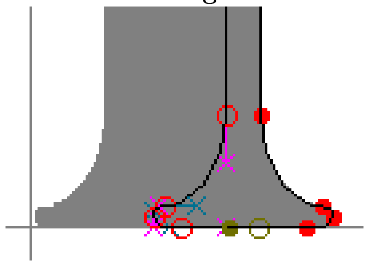
Trace the outline of the right half of the serif

Figure 14: Half traced “l”



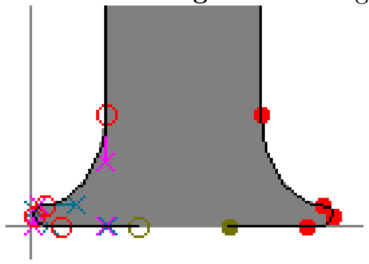
Select the outline, invoke `Edit -> Copy`, then `Edit -> Paste`, and `Element -> Transform` and select `Flip` (from the pull down list) and `check Horizontal`

Figure 15: Pasted “l”



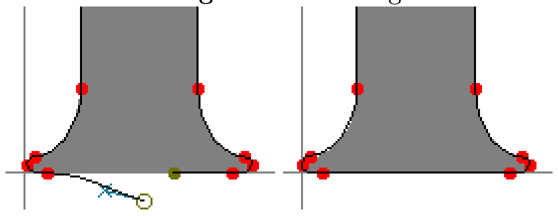
Drag the flipped serif over to the left until it snuggles up against the left edge of the character

Figure 16: Dragged “l”



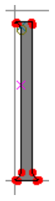
Deselect the path, and select one end point and drag it until it is on top of the end point of the other half.

Figure 17: Joining “l”



Finish off the character.

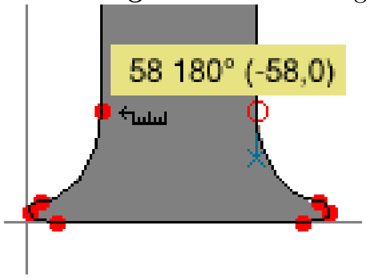
Figure 18: Finished “l”



But there are two more things we should do. First let’s measure the stem width, and second let’s mark the height of the “l.”

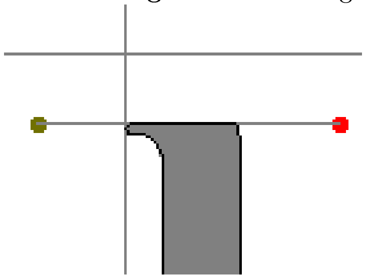
Select the ruler tool from the tool palette, and drag it from one edge of the stem to the other. A little window pops up showing the width is 58 units, the drag direction is 180 degrees, and the drag was -58 units horizontally, and 0 units vertically.

Figure 19: Measuring stem width



Go to the layers palette and select the Guide radio box (this makes the guide layer editable). Then draw a line at the top of the "l", this line will be visible in all characters and marks the ascent height of this font.

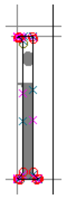
Figure 20: Making a guideline



The “i” glyph looksvery much like a short “l” with a dot on top. So let’s copy the “l” into the “i;” this will automatically give us the right stem width and the correct advance width. The copy may be done either from the font view (by selecting the square with the “l” in it and pressing `Edit -> Copy`) or from the outline view (by `Edit -> Select All` and `Edit -> Copy`). Similarly the Paste may be done either in the font view (by selecting the “i” square and pressing `Edit -> Paste`) or the outline view (by opening the “i” character and pressing `Edit -> Paste`).

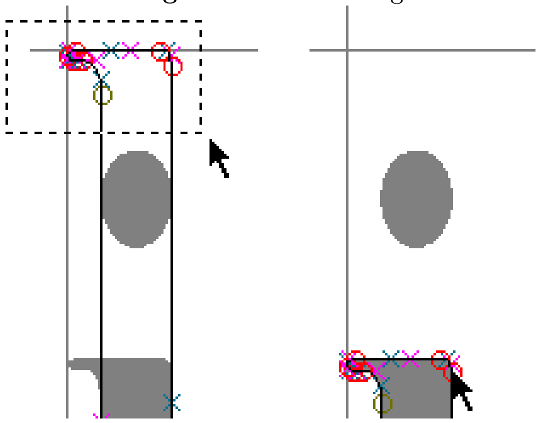
Import the “i” image, and copy and paste the “l” glyph.

Figure 21: Import “i”



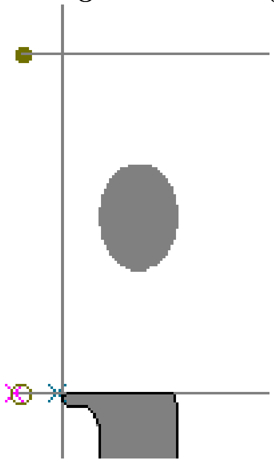
Select the top serif of the outline of the l and drag it down to the right height

Figure 22: Correcting “i”



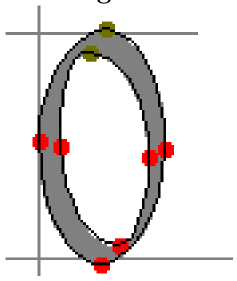
Go to the guide layer and add a line at the x-height

Figure 23: Making another guideline



Looking briefly back at the "o" we built before, you may notice that the "o" reaches a little above the guide line we put in to mark the x-height (and a little below the baseline). This is called overshoot and is an attempt to remedy an optical illusion. A curve actually needs to rise about 3% (of its diameter) above the x-height for it to appear on the x-height.

Figure 24: Comparing "o" to guidelines



Continuing in this manner we can produce all the base glyphs of a font.

Accented letters

Latin, Greek and Cyrillic all have a large complement of accented characters. FontForge provides several ways to build accented characters out of base characters.

The most obvious mechanism is simple copy and paste: `Copy` the letter "A" and `Paste` it to "Ã" then `Copy` the tilde accent and `Paste it Into` "Ã" (N.B. `Paste Into` is subtly different from `Paste`. `Paste` clears out the character before pasting, while `Paste Into` merges the clipboard into the character, retaining the old contents). Then you open up "Ã" and position the accent so that it appears properly centered over the A.

This mechanism is not particularly efficient, if you change the shape of the letter "A" you will need to regenerate all the accented characters built from it. FontForge has the concept of a Reference to a character. So you can `Copy a Reference` to "A", and `Paste` it, the `Copy a Reference` to tilde and `Paste it Into`, and then again adjust the position of the accent over the A.

Then if you change the shape of the A the shape of the A in "Ã" will be updated automagically – as will the width of "Ã".

But FontForge knows that "Ã" is built out of "A" and the tilde accent, and it can easily create your accented characters itself by placing the references in "Ã" and then positioning the accent over the "A". (Unicode provides a database which lists the components of every accented character (in Unicode)). Select "Ã," then apply `Element -> Build -> Build Accented` and FontForge will create the character by pasting references to the two components and positioning them appropriately.

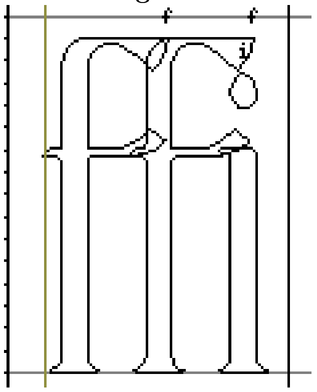
FontForge has a heuristic for positioning accents – most accents are centered over the highest point of the character – sometimes this will produce bad results (if the one of the two stems of "u" is slightly taller than the other the accent will be placed over it rather than being centered over the character), so you should be prepared to look at your accented characters after creating them. You may need to adjust one or two (or you may need to redesign your base characters slightly).

Ligatures

One of the great drawbacks of the standard Type1 fonts from Adobe is that none of them come with "ff" ligatures. Lovers of fine typography tend to object to this. FontForge can help you overcome this flaw (whether it is legal to do so is a matter you must settle by reading the license agreement for your font). FontForge cannot create a nice ligature for you, but what it can do is put all the components of the ligature into the character with `Element -> Build -> Build Composite`. This makes it slightly easier (at least in latin) to design a ligature.

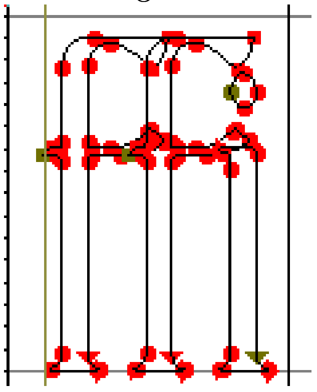
Use the `Element -> Char Info` dialog to name the character (in this case to "ffi". This is a standard name and FontForge recognizes it as a ligature consisting of f, f and i). Apply `Element -> Default ATT -> Common Ligatures` so that FontForge will store the fact that it is a ligature. Then use `Element -> Build -> Build Composite` to insert references to the ligature components.

Figure 25: ffi made of references



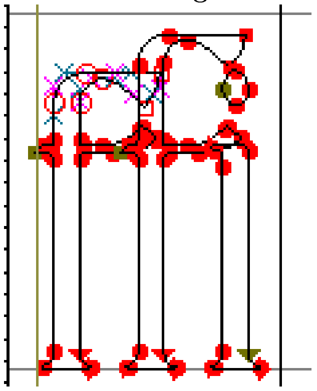
Use `Edit -> Unlink References` to turn the references into a set of contours.

Figure 26: ffi without references



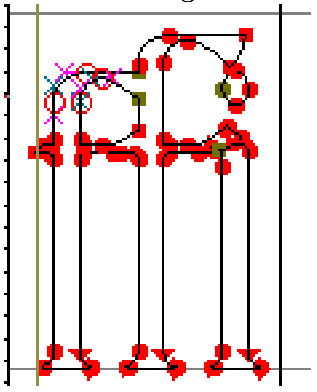
Adjust the components so that they will look better together. Here the stem of the first f has been lowered.

Figure 27: ffi adjusted



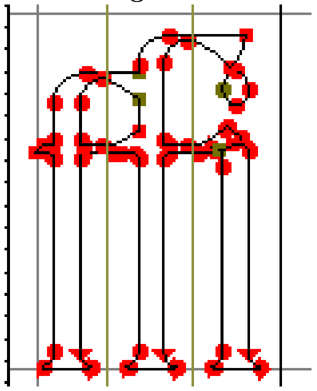
Use `Element -> Remove Overlap` to clean up the character.

Figure 28: ffi cleaned up



Some word processors will allow the text editing caret to be placed inside a ligature (with a caret position between each component of the ligature). This means that the user of that word processor does not need to know s/he is dealing with a ligature and sees behavior very similar to what s/he would see if the components were present. But if the word processor is to be able to do this it must have some information from the font designer giving the appropriate locations of caret positions. As soon as FontForge notices that a character is a ligature it will insert enough caret location lines into it to fit between the ligature's components. FontForge places these on the origin, if you leave them there FontForge will ignore them. But once you have built your ligature you might want to move the pointer tool over to the origin line, press the button and the caret lines to their correct locations. (Only TrueType and OpenType support this).

Figure 29: ffi with ligature carets



Kerning

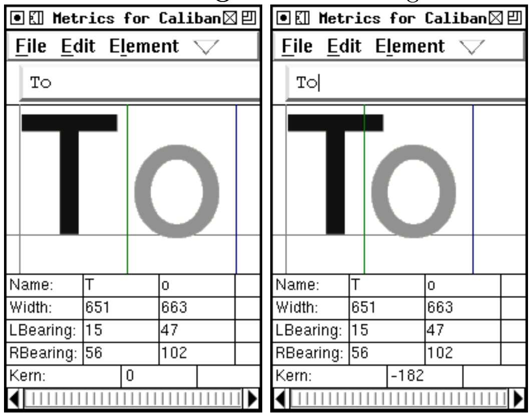
Even in fonts with the most carefully designed metrics there are liable to be some character combinations which look ugly. Some combinations are fixed by building ligatures, but most are best approached by kerning the inter-character spacing for that particular pair.

In the above example the left image shows the unkerned text, the right shows the kerned text. To create a kerned pair, select the two characters, then `File -> Open Metrics View` and click on the right character of the pair, the line (normally the horizontal advance) between the two should go green (and becomes the kerned advance). Drag this line around until the spacing looks nice.

Checking a font for common problems

After you have finished making all the characters in your font you should check it for inconsistencies. FontForge has a command, `Element -> Find`

Figure 30: kerning



Name:	T	o
Width:	651	663
LBearing:	15	47
RBearing:	56	102
Kern:	0	

Name:	T	o
Width:	651	663
LBearing:	15	47
RBearing:	56	102
Kern:	-182	

`Problems` which is designed to find many common problems (as you might guess).

Simply select all the characters in the font and then bring up the Find Problems dialog. Be warned though: Not everything it reports as a problem is a real problem, some may be an element of the font's design that FontForge does not expect.

The dialog can search for many types of problems:

- Stems which are close to but not exactly some standard value
- Points which are close to but not exactly some standard height
- Paths which are almost but not quite vertical or horizontal
- Control points which are in unlikely places
- Points which are almost but not quite on a hint
- and more.

I find it best just to check for a few similar problems at a time, otherwise switching between different types of problems can be distracting.

Generating a font

The penultimate[1] stage of font creation is generating a font. N.B.: FontForge 's `File -> Save` command will produce a format that is only understood by FontForge and is not useful in the real world.

If you plan to use your font with TEX you should insure that the font is encoded with Adobe Standard encoding (Use the `Encoding` tab of `Element -> Font Info` to do this).

Then use `File -> Generate` to convert your font into one of the standard font formats. FontForge presents what looks like a vast array of font formats, but in reality there are just several variants on a few basic font formats: PostScript Type 1, TrueType, OpenType (and for CJK fonts, also CID-keyed fonts).

[1] The final stage of font creation would be installing the font. Installing a PostScript font for TEX is beyond the scope of this document but is well described at: `http://www.ctan.org/tex-archive/info/Type1fonts/fontinstallationguide.pdf`.

Appendix: Additional features

FontForge provides many more features, further descriptions of which may be found at

`http://fontforge.sf.net/overview.html`

Here is a list of some of the more useful of them:

- Users may edit characters composed of either third order Bézier splines (for PostScript fonts) or second order Béziers (for TrueType fonts) and may convert from one format to another.
- FontForge will retain both PostScript and TrueType hints, and can automatically hint PostScript fonts.
- FontForge allows you to modify most features of OpenType's GSUB, GPOS and GDEF tables, and some features of Apple's morx, kern, lcar and prop tables. Moreover it can convert from one format to another.
- FontForge has support for Apple's font formats. It can read and generate Apple font files both on and off a Macintosh. It can generate the FOND resource needed for the Mac to place a set of fonts together as one family.
- FontForge allows you to manipulate strikes of bitmap fonts as well as outline fonts. It has support for many formats of bitmap fonts (including TrueType's embedded bitmap – both the format prescribed by Apple and that specified by MicroSoft).
- FontForge has several means of examining fonts it produces, from the ability to generate PostScript print samples, to the ability to view a text sample on the screen using the FreeType rasterizer.
- FontForge can interpolate between two fonts (subject to certain constraints) to yield a third font between the two (or even beyond). For instance given a "Regular" and a "Bold" variant it could produce a "DemiBold" or even a "Black" variant.
- FontForge also has a command (which often fails miserably) which attempts to change the weight of a font.
- FontForge can automatically guess at widths for characters, and even produce kerning pairs automatically.
- FontForge has some support for fonts with vertical metrics.
- FontForge has a scripting language which allows batch processing of many fonts at once.