Adam Twardoch

Improved Unicode support in FontLab Studio 5

Frankfurt (Oder), April 2005. Version 1.0

The new generation of FontLab's professional font editors (FontLab Studio 5 and AsiaFont Studio 5) includes numerous improvements relevant for Unicode font developers. We present some of the feature highlights: better support for Unicode 4.1, support for advanced layout technologies (improved OpenType Layout support and new AAT support), greatly improved class-based kerning and metrics. We discuss the key improvements in FontLab Studio 5 that are helpful in the process of designing and developing fonts for Unicode environments. We focus on the issues of glyph naming and encoding as the crucial prerequisite for creating a functioning Unicode-compatible font. We present guidelines for glyph naming and encoding and discuss two specific case studies that demonstrate their practical application.

Fontlab Ltd. [1] is an international software vendor that has stayed at the forefront of digital font management by remaining devoted to developing font editors and typography products. Their full line of products is dedicated to solving the most complex typography issues. These products include: FontLab, AsiaFont Studio, ScanFont, TypeTool, SigMaker, TransType, CompoCompiler, BitFonter and FONmaker. The Fontlab Ltd. team works in offices in Canada, Germany, Panama, Russia and USA. With FontLab Studio 5 and AsiaFont Studio 5, software developers and type designers are able to create professional-quality Unicode-savvy OpenType fonts: either from new artwork or by converting their existing font libraries from legacy formats.

Adam Twardoch [2] is Scripting Products and Marketing Manager at Fontlab Ltd. In addition, Adam serves as typographic consultant to MyFonts.com, as OpenType consultant to Linotype Library, and provides consulting services in font tool development, font technology and multilingual typography to other clients worldwide. He regularly writes and lectures about fonts and typography. He is member of the ATypI Board and ATypI country delegate for Poland. Born 1975 in Poland, Adam now lives in Frankfurt (Oder), approximately 80 km east of Berlin.

1 Introduction

In 1975, at the ATypI conference in Warsaw, Peter Karow from the Hamburg-based company URW introduced Ikarus, the world's first digital type design system that worked with outline fonts. Ten years later, Adobe created PostScript and the Type 1 font format, which both became standards in publishing. In the early 1990s, Apple introduced the TrueType font format and the Unicode Consortium published the Unicode Standard. Both initiatives laid the foundations for multilingual text processing and were subsequently implemented in Microsoft Windows and Mac OS. The turn of the millennium brought about OpenType, a significant initiative that unified PostScript, TrueType and Unicode, and added a sophisticated system of advanced typographic features. The year 2005 marks an unusual anniversary: 30 years of digital font technology.

The development of the digital font technology makes it easier for end-users to do text processing, typesetting and layout without sacrificing the typographic quality and logical correctness of the text. But nothing gets lost in Nature: using fonts is getting easier but developing them is more complex. Apart from just drawing letters, a type designer needs to know about encoding, hinting, layout features and various parameters that need to be set inside of a font.

FontLab Studio is an application that assists the font developer in all that: a digital font editor for Mac and Windows that allows the designer to create professional-level fonts from start to end. In 2005, Fontlab Ltd. releases FontLab Studio 5, a new version of the application. This new release of FontLab Studio 5 brings some major improvements that should facilitate the creation of fonts that conform to the Unicode Standard.

In this paper, we will only discuss the issues of creating Unicode-compatible fonts in the TrueType and OpenType format. It should be noted, however, that FontLab Studio also has the ability to create and modify Type 1 and Multiple Master fonts and legacy-encoded fonts. In addition, AsiaFont Studio has the ability to create and modify CID-keyed PostScript and OpenType fonts. A general introduction for using FontLab Studio and other FontLab applications can be found in the book *Learn FontLab Fast* [3] and in the application's reference guide [4].

Whenever we speak of a *font* in this paper, a TrueType or OpenType font is assumed. This paper will not discuss the OpenType font format in detail. For technical details regarding the OpenType format, please consult the OpenType specification [5] and other cited reference works.

One terminological remark needs to be made upfront. The TrueType font format [6] was defined by Apple Computer and further extended by Microsoft Corporation. Subsequently, Adobe Systems contributed some extensions to the format already extended by Microsoft – and the format was renamed to OpenType. In the same time, Apple Computer developed their own extensions independently retaining the name TrueType. This paper uses the term *OpenType* to collectively refer to all fonts that are compatible with either the OpenType specification or the TrueType specifica-

tion. When referring only to the OpenType fonts with PostScript outlines stored in the *CFF* table, we use the term OpenType PS. When referring only to the TrueType fonts and OpenType fonts with TrueType outlines stored in the *glyf* table, we use the term OpenType TT. When we refer to the advanced typographic features of OpenType, we use the term OpenType Layout.

At the time of writing, FontLab Studio 5 has not been released yet. However, the key concepts presented in this paper equally apply to the currently available version, FontLab 4.6. Whenever we refer to FontLab Studio, the same remarks usually apply to the application's high-end cousin AsiaFont Studio. The main difference between FontLab Studio and AsiaFont Studio is that the former can only produce fonts that include less than 6,400 glyphs. AsiaFont Studio can produce fonts with up to 65,535 glyphs.

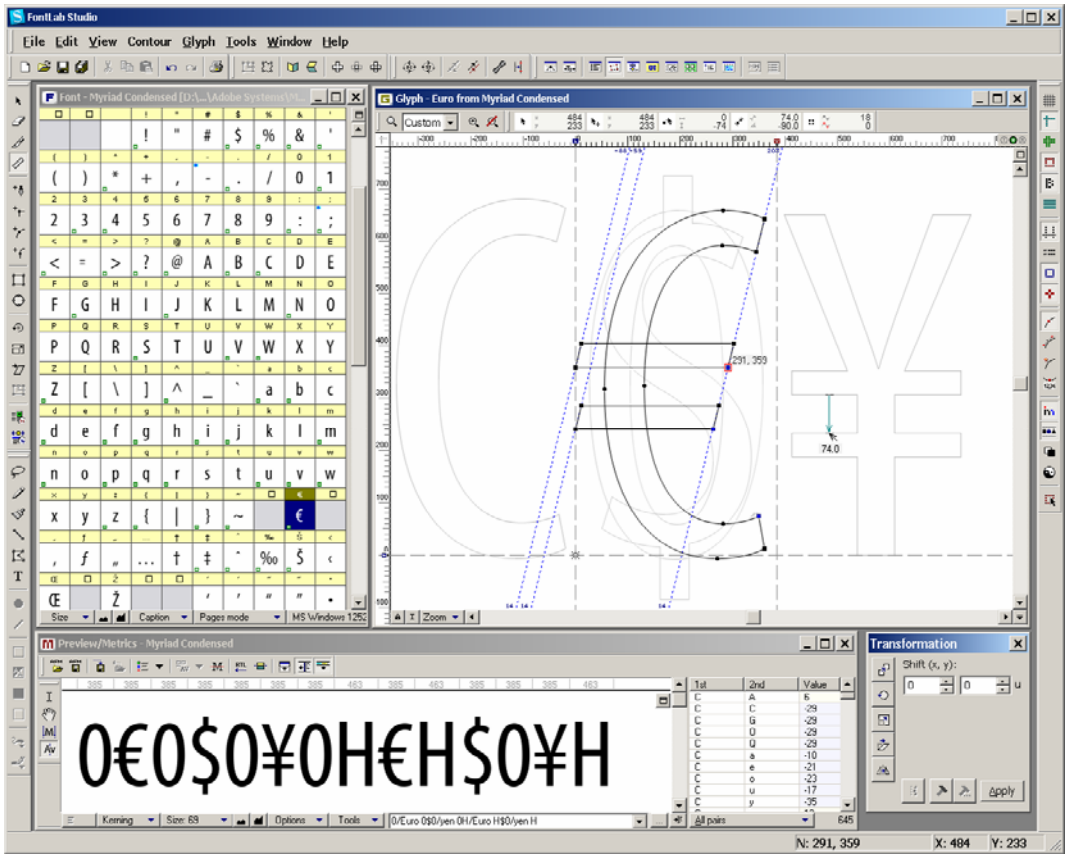

Figure 1. *Three main elements of the FontLab Studio 5 user interface: the Font Window showing the character set of a font, the Glyph Window for designing the glyph shapes, and the Metrics Window for setting metrics and kerning.*

2 Building Blocks of a Font

The three main elements of FontLab Studio's user interface are: the *Font Window*, the *Glyph Window* and the *Metrics Window*. In the Font Window, the user navigates through the font, assigns names and Unicode codepoints to glyphs, views the glyphs in various arrangements, and changes their physical arrangement within the font. In

the Glyph Window, the user creates and modifies the appearance of a glyph, by drawing the glyph's outlines or assembling the glyph from existing parts. In the Metrics Window, the user specifies the appearance of the glyphs when they are arranged on a page, that is, modifies the glyphs' metrics and defines kerning. In additional, FontLab Studio includes a number of panels and toolbars that provide additional functionality.

FontLab Studio's Font Window is the table of contents of a font. All the glyphs included in the font are displayed in a cell grid. Each font is a sequence of glyphs numbered with consecutive indexes (GIDS): 0, 1, 2, 3 etc. The number of glyphs in a font, and their ordering, may differ from font to font. A typical Western font contains between 200 and 300 glyphs, an Asian font can have 50,000 glyphs or more (note that such large fonts can only be processed with AsiaFont Studio, not FontLab Studio). Glyphs in digital fonts are constructed out of outlines (curves and straight segments). A glyph can also include components: a closed portion of the outline can be constructed as a reference to another glyph present in the same font.

As mentioned above, each glyph in a font is uniquely identified by a *glyph index* (GID). FontLab Studio also requires that every glyph in the font has a *glyph name*. The glyph name is visible and can be modified in the glyph's *Properties panel*. To open the Properties panel, click on a glyph and choose from the menu: *Edit / Properties*. Note that the glyph name must not be confused with the Unicode character name! A detailed discussion of glyph naming is included later in this paper.

FontLab Studio follows the character-glyph model defined by the Unicode Standard [7]. Each glyph may be associated with one or more *Unicode codepoints*. The Unicode codepoints are also visible and can be modified in the Properties panel. In FontLab Studio 5, the Unicode codepoints are expressed using hexadecimal digits.

In FontLab 3.0, the Unicode support was limited to the first 65,535 codepoints (the Basic Multilingual Plane, BMP), or to four-digit Unicodes. In FontLab 4.6, experimental support for higher Unicodes (SMP, Supplementary Multilingual Planes) was added, but it never worked correctly. FontLab Studio 5 can finally create fonts that are fully compatible with the Unicode Standard version 4.1 – Unicode codepoints with 4, 5, or 6 digits are equally well supported.

The glyphs in the Font Window can be presented in different arrangements, depending on the Font Window mode selected. The Index mode displays the physical sequence of the glyphs in the font, ordered by GIDS. The Names mode arranges glyphs according to a so-called *"encoding"*. "Encoding" is a list of glyph names ordered in a particular sequence. The term "encoding" is a bit misleading: when developing a Unicode font, the "encoding" selected in the Names mode does not represent the actual encoding of the font. The Names mode is used to help the designer navigate through the glyph repertoire of a font. When a certain "encoding" is selected, the glyphs covered by that encoding are presented at the top of the window, highlighted in yellow color. The designer can easily create custom encodings and use them to switch between different views of the entire glyph repertoire of the font.

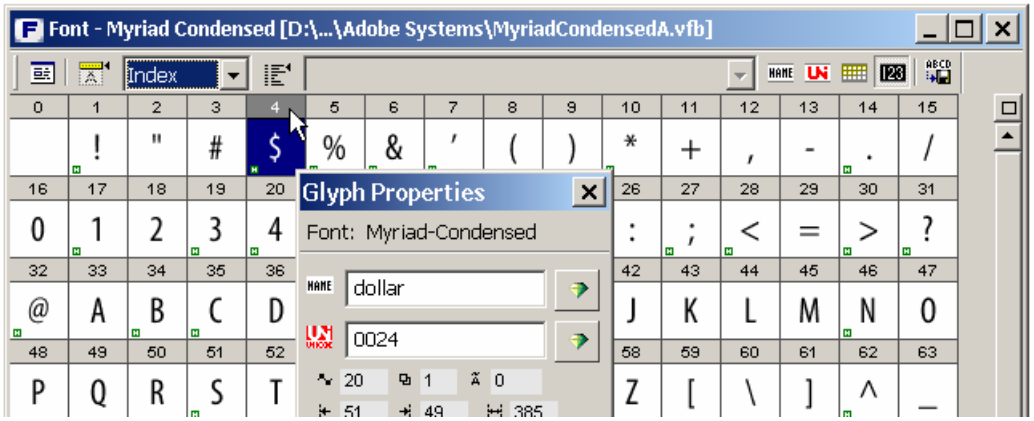

Figure 2. *Index mode of the Font Window. A font is just a sequence of enumerated glyphs, each glyph has a name and (often) a Unicode codepoint assigned.*

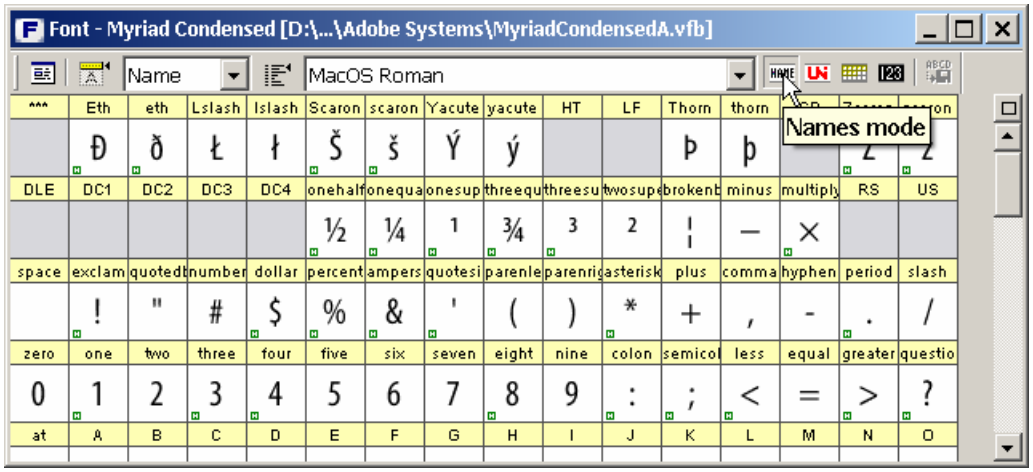

Figure 3. *Names mode of the Font Window. For Type 1 fonts, it represents the font's encoding. For TrueType and OpenType fonts it's just a handy glyph browser.*

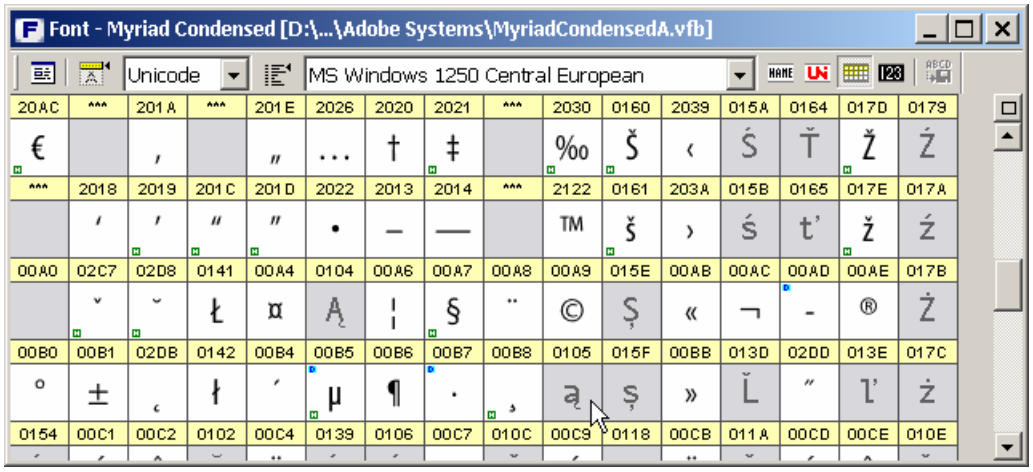

Figure 4. *Codepage mode of the Font Window. When a glyph is missing, a template image is displayed.*

The Font Window has two more modes – the Unicode mode and the Codepage mode – that work analogically to the Names mode. The Unicode mode can filter glyphs by Unicode range, the Codepage holds a huge number of legacy codepages that can be used to view glyphs in a particular arrangement.

3 Glyph Naming and Encoding

3.1 General provisions

Theoretically, the OpenType specification permits the designer not to supply any glyph names at all (at least in case of OpenType TT fonts). However, realistically, it is essential to create fonts with glyph names that fulfill the recommendations detailed below.

When fonts are embedded in electronic documents or sent to a printer, under some circumstances only the information about the glyphs (their GIDs, names and outlines) are retained, while the encoding information (the associated Unicode codepoints) is lost. The electronic document "looks right" but the underlying text streams are obscured or not available. In such cases, meaningfully constructed glyph names can be used as a help to rebuild or at least approximate the original text. A practical example: the user creates a text document that uses an OpenType PS font. The document is printed to a PostScript file. Since PostScript does not support OpenType PS, the font is embedded in the print stream as Type 1. The OpenType information such as layout tables or Unicode codepoints is lost. If Acrobat Distiller is used to convert the PostScript file to a PDF document, the application first tries to locate the original OpenType PS font on the user's system: if the font is found, Distiller is able to use its original Unicode codepoints and embed them in the PDF document. But if the original OpenType PS font is not available to Distiller (for example because the PS-to-PDF conversion happens on a different machine), Distiller embeds the Type 1 font found in the PostScript stream, with no Unicode information. Now, when the text in the PDF document is being searched, copy-pasted or otherwise extracted by an application such as Acrobat or Google, the application can attempt to rebuild the Unicode codepoints basing on glyph names included in the embedded Type 1 font. For Latin or Cyrillic scripts, the recreated text will likely be a very close match of the original; for Thai or Hindi, the text recreated that way will probably be only a crude approximation, with letters arranged in incorrect sequence, and some information missing. But yet, some is often better than nothing.

Users of the Unicode Standard familiar with the character-glyph model know that the relationship between glyphs and characters is not a simple one-to-one mapping.

A glyph in a font can represent the default form of a character. Such glyph needs to have the Unicode codepoint of the represented character assigned. For example, the glyph with the GID 4 representing the character $ (U+0024, DOLLAR SIGN) has the Unicode codepoint *0024* assigned in the Unicode field of the Properties panel. The glyph's name is *dollar*.

A glyph in a font can also represent a variant form of a character. For example, a font can include a glyph that represents the default form of the character a (U+0061, LATIN SMALL LETTER A) as well as glyphs that depict stylistic variants of that character: a small-cap variant, a swash variant, etc. The glyph that represents the default form should have the name *a* and the Unicode codepoint *0061* assigned in the Properties panel. The variant glyphs should different names constructed according to guidelines outlined below. An appropriate OpenType Layout feature [8] should allow the font user to produce the particular variant on the screen – the use of an application and operating system that supports OpenType Layout features is a prerequisite. To allow the user access to the variant glyphs in applications that support Unicode but do not support OpenType Layout features, each of the glyphs may have a PUA (Private Use Area) Unicode codepoint assigned – but they may also remain unencoded.

A glyph in a font can represent the default form of more than one character. Here, several cases need to be differentiated.

First, several characters can have identical appearance, so the same glyph could serve as the default representation of each of those characters. In such case, the Unicode codepoints of all the represented characters are entered in the Unicode field of the Properties panel, separated by spaces. For example, the glyph with the GID 66 with the shape of the letter a could represent two characters: U+0061 (LATIN SMALL LETTER A) and U+0430 (CYRILLIC SMALL LETTER A). In this case, the Unicode field in the Properties panel would have the entry *0061 0430*. The glyph name is *a*. Generally however, assigning multiple Unicode codepoints to one glyph is not recommended, in particular when creating OpenType PS fonts. The designer should rather duplicate the glyphs, assigning no more than one Unicode index to each of them (note that one of the glyphs can refer to the other one as a component).

Another case is that one glyph represents several Unicode characters at a time. For example, the glyph Ẹ́ represents an accented character LATIN CAPITAL LETTER E WITH ACUTE AND DOT BELOW (used in African languages such as Yoruba). This character does not have its own codepoint in the Unicode Standard so it needs to be encoded as a series of characters. It is encoded as *E*, followed by *dot below*, followed by *acute* (U+0045 U+0323 U+0301). Another example is the glyph ffk which is a ligature of *f* followed by *f* followed by *k* (U+0066 U+0066 U+006B). In such cases, again, the appropriate OpenType Layout features must be used to produce the glyphs. The glyph name should be constructed accordingly and the glyph may have a PUA Unicode codepoint assigned.

The following sections present a summary of the Adobe/FontLab glyph naming guidelines. There guidelines unify recommendations by Adobe Systems [9] and those by Fontlab Ltd.

3.2 Glyph name limitations

A glyph name must not be longer than 31 characters. The glyph name consists of a basename, optionally followed by a period (.) which is then followed by a suffix. Both the basename and the suffix may only include: uppercase English letters (A-Z),

lowercase English letters (a-z), European digits (0-9), and underscore (_). Other characters such as spaces are not permitted! A glyph name must start with a letter or the underscore character – with the exception of the special glyph name ".notdef" that starts with the period. For example, "twocents", "a1", and "_" are valid glyph names, while "2cents" and ".twocents" are not.

3.3 Simple glyph names

Review the Adobe Glyph List for New Fonts (AGLFN) [10]. If your glyph represents a character listed in AGLFN, use the glyph name listed there. Instead of using arbitrary names (e.g. "middot"), use standardized names listed in AGLFN ("periodcentered").

Review the Unicode Standard code charts. If your glyph represents a default form of a character encoded in the Unicode Standard but not listed in AGLFN:

a) for BMP codepoints, use the name "uniXXXX", that is lowercase "uni" followed by a four-digit Unicode codepoint written using uppercase hexadecimal digits. Note that "uni" must be lowercase and XXXX must use uppercase letters for hexadecimal digits, so "uni01EB" is a vaild glyph name but "uni01eb" or "Uni01Eb" are not.

b) for SMP codepoints, use the name "uXXXXX" or "uXXXXXX", that is lowercase "u" followed by 5 or 6 uppercase hexadecimal digits representing the codepoint.

3.4 Glyph names with suffix

If your glyph represents an alternate form of a character that is encoded in the Unicode Standard or is listed in AGLFN, use the glyph name of the basic form as the basename, followed by a period, followed by a suffix.

For the suffix, use the name of the OpenType Layout feature that you would most likely access that glyph through.

For example, for a small-caps A, use "A.smcp", for a stylistic alternate R use "R.salt", for a swash Q use "Q.swsh", for a superior m use "m.sups", for a tabular 5 use "five.tnum" etc. If there are multiple OpenType Layout features that can be used to access a glyph, pick one of your likings.

3.5 Compound glyph names

If your glyph represents a "compound character", i.e. a ligature or an accented character that does not have a precomposed Unicode codepoint, and if the character is not explicitly listed in AGLFN or the Unicode Standard, construct the compound glyph name as follows.

For each element of the compound character, take the basename (or the entire glyph name if there is no suffix). Concatenate these using underscore to make the compound basename.

For each element of the compound glyph that has a suffix, concatenate the suffixes using underscore to make the compund suffix. You may eliminate duplicate suffix elements.

For example, for a ligature of the glyphs “c” and “t”, use “c_t” as glyph name. For a ligature of the glyphs “f”, “f” and “i”, use “f_f_i” as glyph name. For a ligature of “longs” and “i” use “longs_i” as glyph name. For a ligature of the glyphs “F.smcp”, “F.smcp” and “I.smcp”, use “F_F_I.smcp” as glyph name. For a ligature of the glyphs “R.salt” and “s.sups”, use “R_s.salt_sups” as glyph name. For the African Ḛ́ character use the glyph name “E_dotbelowcomb_acutecomb”.

If each element of a compound glyph name represents a BMP character, you can use an alternative way of building the basename, which can potentially produce a shorter glyph name. The glyph name starts with “uni” and must be followed by unseparated groups of four uppercase hexadecimal digits representing the BMP codepoint of each element. So instead of “E_dotbelowcomb_acutecomb”, you can use the name “uni004503230301”.

Remember that a glyph name should be no longer than 31 characters, so you may need to abbreviate the name if needed.

3.6 Symbol glyph names

If a glyph does not represent a Unicode character, but rather is an ornament, a non-textual symbol etc., you can use a glyph name of your liking (but adhering to the limitations outlined in 3.2). If you assign PUA codepoints to these glyphs, you can create the glyph names using the “uniXXXX” scheme, where XXXX represents the PUA codepoint.

3.7 Additional naming guidelines

Refer to the Adobe guidelines [9] for additional guidelines on making glyph names, especially for creating complex glyph names that involve “uniXXXX” and “uXXXXX” glyph names as elements.

3.8 Proper Unicode codepoints

Refer to the Unicode Standard code charts and assign proper Unicode codepoints to the glyphs discussed in 3.3.

As discussed above, if the more than one Unicode character share the same glyph shape, two approaches are theoretically possible:

a) create multiple glyphs with identical content but different names, and assign one Unicode codepoint per glyph; for example, create a “periodcentered” glyph and encode it as U+00B7, and create a “uni2219” glyph and encode it as U+2219. One of

the glyphs can refer to the other one as a component. This is the approach recommended by Fontlab Ltd. for OpenType fonts, in particular for OpenType PS fonts.

b) alternatively, either assign multiple Unicode codepoints to your glyph; for example, for "periodcentered", assign U+00B7 and U+2219.

3.9 Private Use Area codepoints

For glyphs discussed in 3.4 – 3.6, you may assign custom codepoints from the Unicode Private Use Area (PUA): from U+E000 to U+F8FF. However, you also may choose to leave these glyphs unencoded (not assign any codepoints).

For some applications (e.g. Microsoft Word 2003 for Windows), assigning PUA codepoints may be the only way to display such glyphs in your font, so it is practical to assign PUA codepoints. On the other hand, PUA codepoints are completely custom, so there is no way that exchangeability of documents can be guaranteed. Also, the text that is set using PUA codepoints is "garbled" (spelling, hyphenation, search & replace won't work). So this is only a short-sighted interim measure.

Some font developers (e.g. Adobe) assign PUA codepoints to glyphs that do not have proper Unicode codepoints, while others (Microsoft, Bitstream, Linotype, Tiro Typeworks) leave the glyphs unencoded.

3.10 Case study I: The tcedilla vs. tcommaaccent confusion

During the development of the Unicode Standard, some confusion was introduced with regard to glyphs that involve the cedilla and the commaaccent acccents. The case study below illustrates the relation between Unicode codepoints, Unicode character names, glyph names and the actual glyph design.

Use C/c with a connecting cedilla accent below for Turkish:
Glyph name: *Ccedilla*, Unicode: *00C7* (LATIN CAPITAL LETTER C WITH CEDILLA)
Glyph name: *ccedilla*, Unicode: *00E7* (LATIN SMALL LETTER C WITH CEDILLA)

Use S/s with a connecting cedilla accent below for Turkish:
Glyph name: *Scedilla*, Unicode: *015E* (LATIN CAPITAL LETTER S WITH CEDILLA)
Glyph name: *scedilla*, Unicode: *015F* (LATIN SMALL LETTER S WITH CEDILLA)

Use S/s with disconnected undercomma accent for Romanian:
Glyph name: *Scommaaccent*, Unicode: *0218* (LATIN CAPITAL LETTER S WITH COMMA BELOW)
Glyph name: *scommaaccent*, Unicode: *0219* (LATIN SMALL LETTER S WITH COMMA BELOW)

Make duplicate glyphs Tcommaaccent/uni021A and tcommaaccent/uni021B with mappings listed below. Use T/t with disconnected undercomma accent for Romanian in both these cases.
Glyph name: *Tcommaaccent*, Unicode: *0162* (LATIN CAPITAL LETTER T WITH CEDILLA)
Glyph name: *uni021A*, Unicode: *021A* (LATIN CAPITAL LETTER T WITH COMMA BELOW)

Glyph name: *tcommaaccent*, Unicode: *0163* (LATIN SMALL LETTER T WITH CEDILLA)
Glyph name: *uni021B*, Unicode: *021B* (LATIN SMALL LETTER T WITH COMMA BELOW)

Alternatively, create *Tcommaaccent* with two Unicode codepoints assigned *(0162, 021A)* and *tcommaaccent* with two Unicode codepoints assigned *(0163, 021B)*.

As an alternative approach, you may choose to design a hybrid disconnected accent that serves as both cedilla and commaaccent. Please refer to the Microsoft Diacritics Design Standards document [11].

Short: call your glyphs "something-cedilla" when they use cedilla and "something-commaaccent" when they use commaaccent. Supply C and S with cedilla for Western and Turkish and G, K, L, N, R, S, T with commaaccent for Romanian and the Baltic languages.

3.11 Case study II: No precomposed Unicode codepoints

As explained in 3.1, characters such as LATIN CAPITAL LETTER E WITH DOT BELOW AND COMBINING ACUTE ACCENT (Ḗ) do not have a precomposed Unicode value. The glyph that represents such a character needs special treatment – the font must include an OpenType Layout feature that maps the composite codepoints to the actual glyph.

To include such a font in your OpenType font, you will need to create the Canonical Composition/Decomposition OpenType feature (*ccmp*) [12]. First, the "ingredients" of the character must be identified. LATIN CAPITAL LETTER E WITH ACUTE AND DOT BELOW consists of the LATIN CAPITAL LETTER E, COMBINING DOT BELOW and COMBINING ACUTE ACCENT:

- U+0045 (LATIN CAPITAL LETTER E), glyph name *E*
- U+0323 (COMBINING DOT BELOW) glyph name *dotbelowcomb*
- U+0301 (COMBINING ACUTE ACCENT) glyph name *acutecomb*

The canonical decomposition of the character, i.e. the Unicode Normalization Form D (NFD) of the character is: U+0045 U+0323 U+0301. This is the „default" form in which the character should be represented in the text. According to the glyph naming guidelines detailed above, the name of the glyph Ḗ could be *uni004503230301* or *E_dotbelowcomb_acutecomb*. We will choose the first one for brevity.

In addition to the canonical components of the character, we should consider that parts of the character do exist in the Unicode standard in precomposed form:

- U+00C9 (LATIN CAPITAL LETTER E WITH ACUTE), glyph name *Eacute*
- U+1EB8 (LATIN CAPITAL LETTER E WITH DOT BELOW) glyph name *uni1EB8*

We should not assume that all Unicode texts in which we want to use our character will exist in canonically decomposed NFD form. Therefore, we should make provisions for all possible combinations of Unicode characters that can represent our LATIN CAPITAL LETTER E WITH DOT BELOW AND COMBINING ACUTE ACCENT character. This means that our font should contain all of the following glyphs: *E, dotbelowcomb, acutecomb, Eacute, uniEB8*.

This also means that the final character can be a combination of the following:

- U+0045 U+0323 U+0301 (glyphs: *E dotbelowcomb acutecomb*)
- U+0045 U+0301 U+0323 (glyphs: *E acutecomb dotbelowcomb*)
- U+00C9 U+0323 (glyphs: *Eacute dotbelowcomb*)
- U+1EB8 U+0301 (glyphs: *uni1EB8 acutecomb*)

To create the glyph for the character LATIN CAPITAL LETTER E WITH DOT BELOW AND COMBINING ACUTE ACCENT in FontLab Studio 5 choose *Glyph / Generate Glyphs* and type in the following:

```
E+dotbelowcomb+acutecomb=uni004503230301
```

Finally, refine the design of the glyph (adjust the positioning of the diacritics as required). Open the OpenType panel (*Window* menu), add a new feature and in the feature definition code field, type in the following:

```
feature ccmp {
  sub E dotbelowcomb acutecomb by uni004503230301;
  sub E acutecomb dotbelowcomb by uni004503230301;
  sub Eacute dotbelowcomb by uni004503230301;
  sub uni1EB8 acutecomb by uni004503230301;
} ccmp;
```

Click on the *Compile* button, then on the *Open Preview Panel* button. In the OpenType Preview panel that opens, activate the *ccmp* feature and type in the following:

```
\u0045\u0323\u0301 \u0045\u0301\u0323 \u00C9\u0323 \u1EB8\u0301
```

to test your new character.

Generate the font as Win TrueType / OpenType TT (.ttf) or as OpenType PS (.otf), install it in the system and in Microsoft Word 2003 for Windows (or newer) and test your new character: type in the hexadecimal codes of the composite codepoints and press *Alt-X* after each of them. Alternatively, use the Character Map application or a custom keyboard driver.

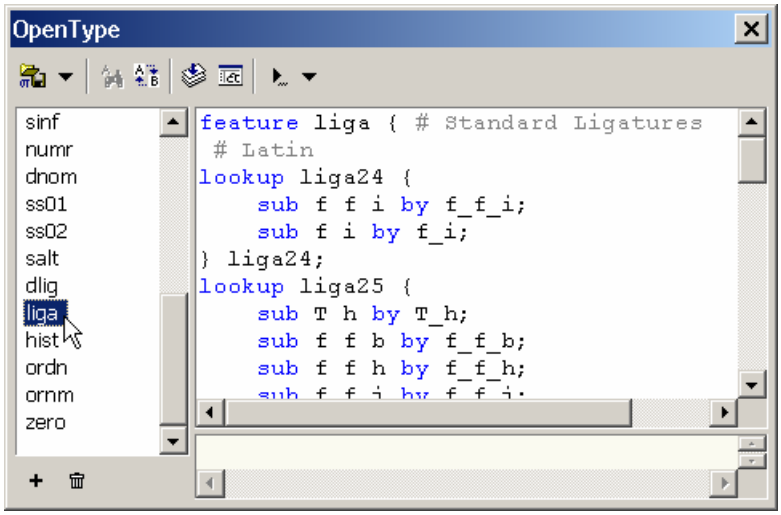

Figure 5. *The OpenType layout feature definitions are stored in the OpenType panel using the Adobe FDK for OpenType notation*

The technique described above should be applied accordingly to other glyphs that represent characters that do not have precomposed codepoints. The same technique should be applied to building ligatures, except that the mapping of the component glyphs to the ligature glyphs should not be done in the *ccmp* feature but in the appropriate ligature features *(liga, dlig, hlig)*. Please refer to the OpenType Layout features registry for details [13].

4 Designing Glyphs

4.1 Glyph Window

The Glyph Window is the place where the designer creates the letterforms. This is where you see the glyph outline (built from nodes, sometimes called points, as well as straight and curve segments between them), hints, glyph and font metrics, guidelines. A number of different toolbars and panels are available to the user: the Tools and Paint toolbars hold various tools for creating, editing and modifying outlines, the meter toolbar displays the position of the currently selected node and the cursor position – in relation to the (0,0) point and, optionally, in relation to a special reference point that the designer can put anywhere within the drawing space.

The Transformation panel (available from the *Window / Panels* submenu) allows for precise geometric transformation (scaling, shifting, slanting, rotating). When you're doing free transformations by just dragging parts of the outline with your mouse, the Transformation will detect if you're scaling, slanting or rotating, and will automatically display the numerical value of what you are currently doing.

The Editing Layers panel allows the designer to edit different elements of the glyph: its outline, the metrics (the left and the right sidebearing), the guidelines, the hints, and the mask, that is, a second outline layer placed in the background.

In the past, type designers used many different tools – Ikarus, Font Studio, Fontographer, and previous versions of FontLab. Many of these users developed very strong habits as for how certain things should work. Since most of these tools are no longer being developed, FontLab Studio 5 includes options that make the look-and-feel more familiar for users of other tools. The user can control many aspects of what the Glyph Window looks like: control the size and color of nodes, the smoothing of outlines, decide whether to show connection marks or to fill the outlines.

Something completely new to FontLab Studio 5 is in-context editing. FontLab 4.6 users know the mask layer where the designer can place an additional outline for reference. In FontLab Studio 5, a new kind of dynamic masks are introduced: shape groups and neighbors. The shape groups layer displays a number of semi-transparent glyphs stacked behind the outline of the current glyph. The neighbors layer shows glyphs left and right of the current outline. The composition of shape groups and neighbors changes automatically for each glyph that you edit, and naturally, this composition can be fully customized.

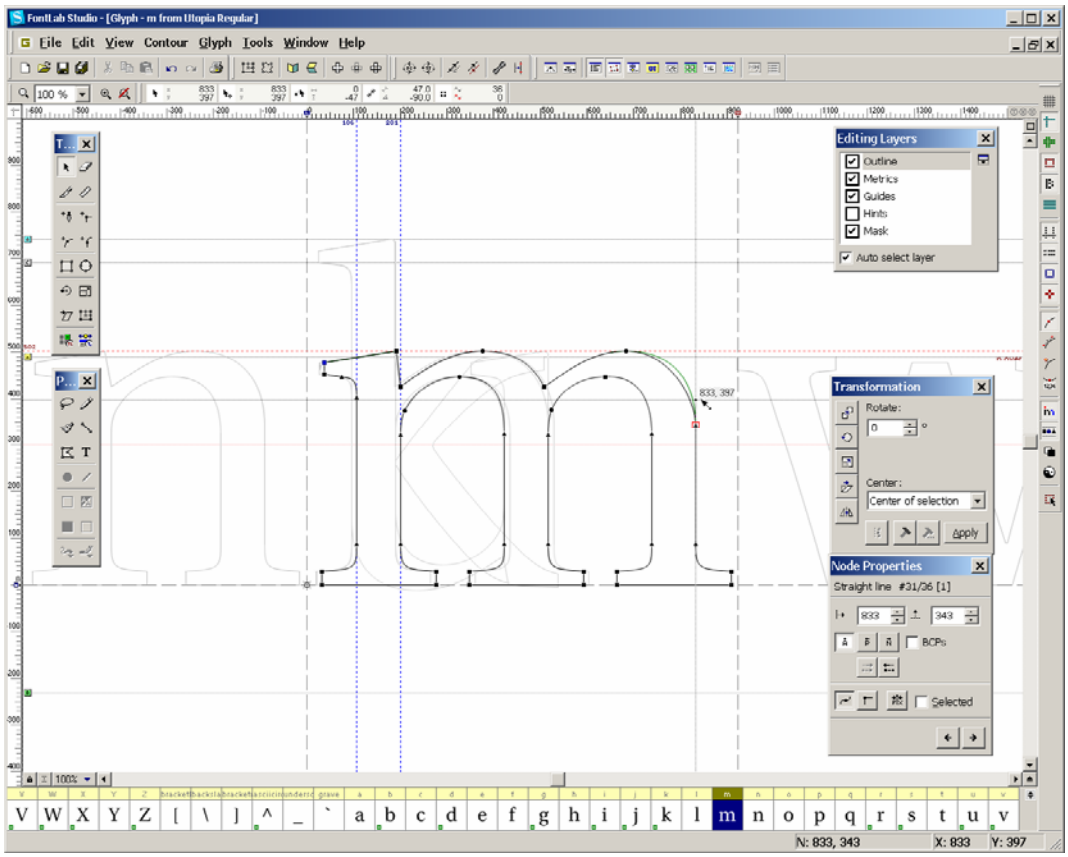

Figure 6. *The Glyph Window is the heart of FontLab Studio 5.*

4.2 Alternative ways to add glyphs

Until now, autotracing was a feature unique to Fontlab Ltd.'s separate product, ScanFont. While ScanFont still remains useful as it can quickly and automatically separate an entire scanned alphabet into single characters, FontLab Studio 5 has a built-in Trace feature. The user can place a black-and-white bitmap of a character into the Background layer and FontLab Studio 5 will convert it into an outline, with customizable precision. FontLab Studio 5 can also import illustrations saved in EPS format. If you prefer to draw in Illustrator or Freehand, remember to save your EPS in Illustrator 6 or Illustrator 8 format rather than any of the newest formats. FontLab will then have less trouble in importing the EPS.

The cells that represent glyphs missing from your font have a gray background and a glyph template image that you can use as orientation for your design. FontLab Studio 5 includes a new, very extensive glyph template font provided by Monotype Imaging [14]. It is a one-size bitmap version of Andale Mono WTG that provides low-resolution character previews for the entire Unicode 3.2 character range.

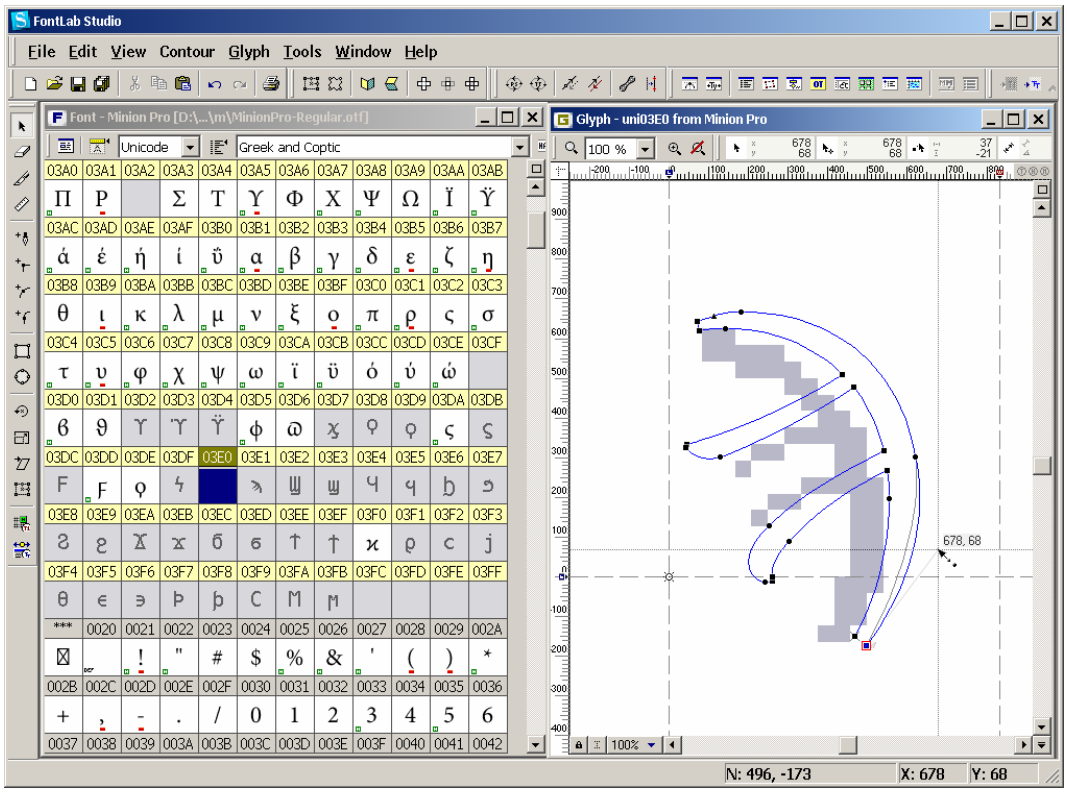

Figure 7. *The new glyph template font (Andale Mono, provided by Monotype Imaging) covers the entire Unicode 3.2 character range and provides small-scale thumbnails for characters that are not included in the current font.*

When adding glyphs to your font, you can use the thumbnail images from the glyph template as guidance. Note that the provided thumbnail templates do not necessarily reflect all the typographical finesse of the respective characters – they should be used as orientation only. Of course, as with most elements of the user interface, this glyph template font can be customized: the user can specify a different font to be displayed as the glyph template font.

FontLab Studio 5 has a new handy tool that allows the designer to quickly create a set of accented characters. Select certain Unicode range (such as Latin Extended-A), right-click on any glyph and choose *Select Encoding* from the context menu, and finally choose *Glyph / Create Glyphs if Empty*". A set of stub accented characters will be automatically created from elements that are present in the font. FontLab Studio 5 comes with a large set of "recipes" on how accented characters should be built – which composites to use and how to place them. The user can modify and extend these recipes.

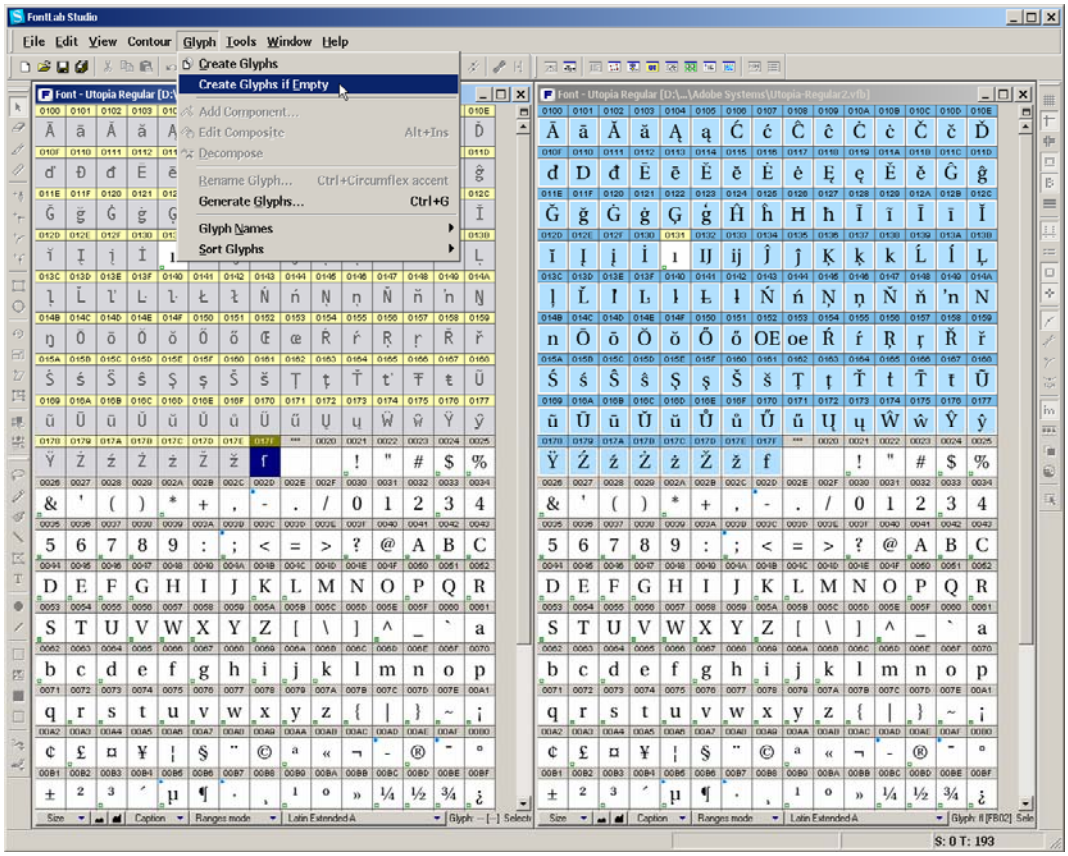

Figure 8. *If your font is missing some accented characters, FontLab Studio 5 will attempt to create them for you.*

5 Letterspacing and Kerning

The Metrics Window probably includes the most important changes in FontLab Studio 5. In the Classes panel, the user can associate similar glyphs into classes. Kerning classes are used to associate glyphs that can share the same kerning values. FontLab Studio 5 includes a brand-new automatic class builder: the software analyzes the shapes of the glyphs included in the font and finds glyphs that are similar on the left side or on the right side. These glyphs are automatically associated into classes. For example, a kerning class *_B* can include a number of glyphs that are similar in shape on their left side (*B, D, F, H, I* etc.). They can use the same kerning value if one of them appears as the second glyph in a kerning pair. With such kerning class in place, the user only needs to define kerning pairs for one of the glyphs (for example *B*, which is the so-called key glyph). The other glyphs will automatically receive the same kerning values on their left side.

Next page: **Figure 9.** *New Metrics and Kerning editing in FontLab Studio 5 (illustration by Luc(as) de Groot)*


FontLab Studio - [Preview/Metrics - TheSans]
FL Kern Manual 0.1 Luc(as)
Click here to sort the list by 1st, 2nd or Value. When sorted by value, exceptions are at the bottom of the list and colored red.
text
preview
metrics
kerning
class kerning on/off
use measurements line
For FOG junkies
show/hide ruler
change interface options & tools at top or bottom
Auto...
Add Pairs...
Reset Kerning...
Edit Kerning...
Assistance...
Generate Classes...
Never trust auto kerning
to re-use text files
for instance to delete kerning pairs below a certain value
the old kerning info dialog
the kerning assistance dialog
to make classes and compact kerning
click here to deselect a pair
click a glyph to select a pair
doubleclick to open glyph in glyph window
Control-Alt double click on the second glyph to make it show up on the left side of the 1st glyph. HZ becomes ZHZ
icon means first in a pair
Key glyph in class
click icon for class info, Ctrl-click for key glyph!
Use arrow up and arrow down to scroll through this list. Pairs are shown in automatic context. (no pair should be selected on the left) Context can be manipulated in Metrics.txt.
Use arrow left and right (with Shift or Control) to change kerning values.
Click the blue square to see expanded pairs. Click it twice to see the class.
red line exception pair
gray line class pair
blue line real pair
type or paste text here. \n stands for new line
show/hide table
open kern text files here
switch font
switch mode
size options
options and tools from the toolbar above
show pairs for selected glyph(s) only
count without zero pairs
count with zero pairs
Pairs: 16824
Axis
Auto Align
Weight
Masters
Synchronize

In place of the old Metrics Window with just one line of text, the new Metrics Window is a multiline text editor so the user can see the whitespace distribution not only between characters but also between lines of text. The Metrics Window has four modes that you choose using the four buttons on the top left of the window: Text, Preview, Metrics and Kerning. The Text mode works like a little text editor: you can type any text, select, copy and paste. The Preview mode removes all unnecessary lines and marks so you can look at your typeface, and print a sample from the window.

6 OpenType Layout support

To create OpenType layout features in FontLab, the user puts the feature definitions in FontLab's OpenType panel using source notation defined by Adobe Systems. When FontLab generates an OpenType PS or OpenType TT, it compiles the notation into binary OpenType tables using programming code licensed from Adobe Systems – the same code that is included in the Adobe FDK for OpenType. Unlike the Adobe FDK, FontLab is able to do the reverse: to open an OpenType font containing the binary OpenType layout tables and to decompile them into the source notation.

When FontLab 4.6 opens an OpenType, it always decompiles the features, which has turned out to be a limitation. The Adobe FDK for OpenType code does not support certain aspects of the OpenType specification such as the mark-to-base and mark-to-mark GPOS positioning lookups that are extensively used in Arabic, Hebrew and Indic OpenType fonts. When a user opens such font in FontLab 4.6, performs some minimal modifications and tries to generate a new font, FontLab will fail. It cannot recompile the features since they include unsupported lookup types. There is no easy workaround for that problem but fortunately, it is fixed in FontLab Studio 5. The new FontLab version has options that control whether to store the OpenType binary tables and whether to interpret them when opening an OpenType font. With both options enabled, FontLab will decompile the feature definitions and also store their binary form. When the user re-generates the font, FontLab Studio asks him which version of the tables (binary or compiled) to include. The user can also enable just one of the options – in this case he will not be asked when generating the final font.

In Microsoft Windows XP, OpenType PS fonts with the *.otf* extension are always displayed using the green OpenType icon ("O"). OpenType TT fonts with the *.ttf* extension are shown using the blue TrueType icon ("TT") by default. Windows will only display the green OpenType icon for a *.ttf* font if the font includes a digital signature. Font developers should add digital signatures to their font to certify the authorship of the font. A user of an OpenType font by, for instance, Linotype Library can be sure that the font is a genuine Linotype font if it contains a Linotype digital signature. Until now, adding digital signatures was cumbersome: it required using a bunch of command-line tools running only on Microsoft Windows. FontLab Studio 5 has built-in support for digital signatures so the font developer only needs to buy a "code signing" digital certificate from a company such as Thawte or Verisign, starting at 200 USD/year,

FontLab Studio 5 also includes another innovative feature: it can convert selected Apple GX/AAT typographic features (if present in the font) to OpenType Layout features [15].

FontLab Studio's abilities to create OpenType Layout features are currently limited to GSUB substitution features, and kerning. Advanced GPOS positioning used in complex scripts (Arabic, Indic scripts) is not directly supported. However, FontLab Studio 5 provides convenient integration with Microsoft Visual OpenType Layout Tool (VOLT) [16], a free application that can be used to create complex OpenType Layout features. The designer can use FontLab Studio to design the glyphs, assign Unicode codepoints and generate the font; later, he can open the font in VOLT and add the OpenType features there. For more information about creating OpenType features for complex scripts, please consult the specifications published at the Microsoft Typography website [17].

7 Other features

A common scenario for creating multilingual OpenType fonts is that the user has several Type 1 fonts designed in the past, each of them including a part of the desired character set. For example, there can be a Roman font, a Central European font, a Cyrillic font, a small caps font and a font with old-style numerals. FontLab Studio includes some new features useful in such a conversion process. In your Type 1 small-cap font, the small cap glyphs are probably named *a, b, c...* but in the OpenType font, such glyph names already exist (for lowercase letters) so your small caps should get different names, e.g. *a.smcp, b.smcp, c.smcp* etc. With *Add Suffix to Name*, the user can append a particular name suffix to selected glyphs. With *Tools / Merge Fonts*, the user can copy all glyphs (or just the ones with the unique names) from one font to another including kerning pairs.

A feature often requested by users of the 4.6 version was improved font proofing. FontLab Studio 5 has greatly extended printing capabilities. The user can print various samples of single glyphs or the entire font including metric information. There is also a new Quick Test feature: FontLab Studio generates the current font in OpenType PS or OpenType TT format (taking the current export Options into account), installs it temporarily and shows a little text editor window. The user can populate the contents of selected codepages or the entire font character set into the Quick Test window, but can also paste arbitrarily long texts from the clipboard. The Quick Test window displays fonts using the system font mechanism, so on Windows XP Service Pack 2 with complex script support enabled in *Control Panel / Regional and Language Options / Languages,* the user can test OpenType layout features such as ligatures or contextual alternates.

The Font Info dialog that acts as the command central of the font has several new handy features. With the *Copy* feature, the user can quickly copy family names, copyright strings, licensing information, metric and encoding information from one font to all the other fonts in a family. A new *Verify names* button in the Names and Copyright section will check for common name problems like too long names or PostScript font

names with two hyphens. The *Embedding* settings have been updated to reflect the recent OpenType specification. The algorithm that automatically checks codepages and Unicode ranges included in the font has been improved. A new *cmap* editor has been added so OpenType and TrueType fonts with very complex encodings can be created.

8 Closing remarks

This paper only covers a small subset of issues regarding the process of creation of Unicode-compatible fonts. A detailed discussion on specific aspects would have not fit in the scope. We shall however mention some of the most important points and limitations. FontLab Studio 5 does not offer a full bidirectional support. However, the key elements of the user interface such as the Metrics Window support right-to-left rendering as well as (to some extent) vertical rendering.

The final Windows version of FontLab Studio 5 is expected to be released in the spring of 2005, with the Mac version to follow a few months later. The information will be announced on `http://www.fontlab.com`

9 References

[1] Fontlab Ltd. http://www.fontlab.com/

[2] Adam Twardoch. http://www.twardoch.com/adam/

[3] Leslie Cabarga: Learn FontLab Fast. Iconoclassics, Los Angeles 2004. http://www.logofontandlettering.com/

[4] Fontlab Ltd.: FontLab Studio User's Manual. http://www.fontlab.com/

[5] Adobe Systems, Microsoft Corp.: The OpenType specification. http://www.microsoft.com/typography/otspec/

[6] Apple Computer: TrueType Reference Manual. http://developer.apple.com/fonts/TTRefMan/

[7] The Unicode Standard. http://www.unicode.org/

[8] Adobe Systems: OpenType, Advanced Typography. http://store.adobe.com/type/opentype/#adv

[9] Adobe Systems: Unicode and Glyph Names. http://partners.adobe.com/public/developer/opentype/index_glyph.html

[10] Adobe Systems: Adobe Glyph List For New Fonts. http://partners.adobe.com/public/developer/en/opentype/aglfn13.txt

[11] Microsoft Corp: Microsoft Character Design Standards. http://www.microsoft.com/typography/developers/fdsspec/

[12] Microsoft Corp.: ccmp OpenType Layout feature. http://www.microsoft.com/typography/otspec/features_ae.htm#ccmp

[13] Microsoft Corp.: OpenType Layout features registry. http://www.microsoft.com/typography/otspec/featurelist.htm

[14] Monotype Imaging. http://www.monotypeimaging.com/

[15] Apple Computer: Comparing GX Line Layout and OpenType layout. http://developer.apple.com/fonts/WhitePapers/GXvsOTLayout.html

[16] Microsoft Corp.: Microsoft VOLT http://www.microsoft.com/typography/developers/volt/

[17] Microsoft Corp.: Typography Specifications http://www.microsoft.com/typography/SpecificationsOverview.mspx