

Comments on Ligature Control

John H. Jenkins
Apple Computer, Inc.
jenkins@apple.com

General

This document examines proposals to add a ZERO WIDTH LIGATOR (henceforward ZWL) and ZERO WIDTH NON-LIGATOR (ZWNL) to the Unicode Standard. This document examines the issue in light of two shipping technologies which implement the character-glyph model, namely OpenType (OT) from Adobe and Microsoft and Apple Advanced Typography (AAT) from Apple.

OpenType is currently available in Adobe's **InDesign** page layout software for the Macintosh and Windows, in Windows 2000 and the most recent version of **Office** for Windows. AAT is supported through Mac OS system software through Apple Type Services for Unicode™ Imaging (ATSUI) and Multilingual TextEdit (MLTE). Future versions of Apple's Java virtual machine (MRJ) will be using ATSUI and common low-end text editing engines are being converted to use MLTE.

We would like to thank Adobe and Bitstream for supplying OpenType fonts to use. It should also be noted that Adobe joins Apple in feeling that proposals to add ZWL to Unicode are unnecessary and ill-advised for Latin text.

Answers to Some Arguments

Some fundamental points in favor of adding ZWL are:

- 1) If ligation control is left to higher-level protocols, then all ligation information will be lost at conversion to plain text, with the potential for linguistically incorrect display or, at the least, loss of information contained in the original text.
- 2) Similarly, if ligation control is left to higher-level protocols, then the ability to transfer ligated text from one application to another (or from one platform to another) will require an agreed-upon markup format used at both ends which contains this information.
- 3) Other scripts have ligation control in plain-text using the Unicode Standard; why not Latin, Greek, Cyrillic, and so on?

We will respond to these points in that order.

1) If ligation control is left to higher-level protocols, then all ligation information will be lost at conversion to plain text, with the potential for linguistically incorrect display or, at the least, loss of information contained in the original text.

This is, of course, true. It is also true for other information used in text display, and is an inherent limitation derived from the decision to base Unicode character encoding upon “minimal legibility”—that level of information required to do basic display of text in a given script. The decision to restrict Unicode encoding to minimal legibility means inevitably that there are varieties of text-based information exchange for which plain text is inadequate.

It’s important to realize that this decision has ramifications beyond the loss of typeface specification, or point-size specification, or use of italics when such provides a semantic distinction. Whenever Unicode unifies visually distinct glyphs for writing a particular character, loss of the out-of-band information that distinguishes them may result in an “incorrect” display.

UniHan may be a prototypical example for this sort of thing—Japanese readers seeing their email displayed with glyphs for ideographs appropriate for China and not Japan—but it isn’t the only one. The unification of Greek with Coptic means that a copy of a Coptic document might be displayed in a mixture of Greek and Coptic glyphs if the language tag is lost, or a document intended to be displayed using a Serto style of Syriac might end up being displayed with Estrangela if the script-style information is lost. A scholar attempting to recreate the *form* of a particular Etruscan inscription would be limited by Unicode to its *content*.

In and of itself, this is an insufficient argument for extending the standard. One need not be able to reproduce Gutenberg’s Bible in plain text.

2) Similarly, if ligation control is left to higher-level protocols, then the ability to transfer ligated text from one application to another (or from one platform to another) will require an agreed-upon markup format used at both ends which contains this information.

The same response is appropriate here. Apple defines a standard interchange format which would make it possible for any two AT&T-based applications to interchange styled text. This is the heir of Apple’s original standard format for exchanging styled text information for QuickDraw and TextEdit; the new specification, however, includes sufficient style information to include ligature control. Other standard ways of specifying data such as typeface and point size are being defined.

Again, this point in and of itself is insufficient for extending the standard. The fact that some information which may have semantic ramifications is not interchangeable except in higher-level protocols is irrelevant to encoding within Unicode.

3) Other scripts have ligation control in plain-text using the Unicode Standard; why not Latin, Greek, Cyrillic, and so on?

The point comes to the core of the philosophy of encoding within Unicode: Unicode deals with the information required in text for minimal legibility. Both Arabic and Indic scripts cannot be properly rendered in a legible fashion without sophisticated character-glyph mapping processes. The same is not true for Latin, Greek, and Cyrillic. It is also untrue for most East Asian scripts.

In any event, the only scripts for which actual ligation control is available in plain-text Unicode are the Indic scripts, and they use a visible character, halant, to exert that control. Complex Arabic writing styles which make heavy use of ligatures are also not supported in plain-text Unicode.

It is not incumbent upon the UTC to guarantee that the mechanisms required for rendering some scripts be available for all.

How Things Work Right Now

Michael Everson's document, ISO/IEC JTC1/SC2/WG2 N2147, summarizes well the situation in terms of how most font vendors structure their typefaces. Some points should be observed.

1) *Combining accented letters are handled either through normalization to precomposed forms, on-the-fly composition or through higher-level protocols.*

Obviously handling ligatures by limiting access to precomposed forms is unacceptable. It is certainly the way, however, that most accented European letters are handled right now.

As an example of the on-the-fly composition, there is Linguist Software's Graeca font for New Testament Greek. It does not have individual glyphs for every potential combination of vowel-breathing mark-accent-iota subscript. Rather, it contains individual glyphs for iota subscript and for the various combinations of breathing mark and accent. These glyphs have zero width and a negative left side bearing; that is, they hang leftward over the previous character. Thus typical words can be typed as:

'+E+ν+ +α'+ρ+χ+η+ϕ+τ,+ +η+ϕ'+ν+ +ο+ϕ'+ +λ+ο+ϕ'+γ+ο+ς

And it looks like

Ἐν ἀρχῇ ἦν ὁ λόγος

Note that this method will not infrequently result in less-than-optimum display of accented letters. It is possible within OpenType to overcome this difficulty, but in order to do so an application would be required to add OpenType support.

In any event, on-the-fly creation of ligatures is not possible. A font which doesn't have an "ff" ligature glyph won't be able to create one by designing the "f" and "t" glyphs so that they overlap automatically, at least not without disastrous results.

The other mechanism involves mapping the characters to predefined glyphs for accented letters. This requires the font vendor to anticipate the accented letters desired by the people who use their fonts. Moreover, this process can be done directly or indirectly; that is, the application can either map a series of characters to a single glyph, or it can map a series of characters to a series of corresponding glyphs and then manipulate the series of glyphs. This indirect process is precisely what is envisioned by OpenType (OT) and Apple Advanced Typography (AAT).

The TrueType specification, which is the basis for text display on both Windows and the Mac OS, only allows a one-one initial mapping of characters to glyphs. Not only does this provide the simplest design for character-glyph mapping, but it satisfies most needs. The presumption from the beginning of TrueType design was that other, ancillary information would be available in the font to handle the manipulation of arbitrary sequences of glyphs.

That is, support for accented glyphs using combining marks is typically either not optimal or not supported in existing applications.

2) Most of the typefaces that include large sets of ligatures also include large sets of alternate glyph shapes which cannot be accessed in plain text.

There are two very common forms of glyphic variation which typeface designers include are support for old-style numerals or small caps.

The former refers to the fact that numerals used in accounting are usually designed so that each numeral occupies the same horizontal space, making the lining up of columns automatic. Numerals designed for uses such as dates should be designed for the best visual appearance. Hence one could have \$ 1 23.45 but 15 June 1215.

Many word processors provide a small caps feature; however, this is generally done by taking regular capitals and displaying them at a slightly smaller size. This can result in typographical infelicities; type designers will therefore provide a set of small cap letters to use for this. Hence one would see SMALL CAPS and not SMALL CAPS.

In the current world, where glyph selection is possible only through changing characters, this is typically done by a type designer including additional fonts for their face. Many typefaces, such as ITC's Legacy Serif design, include an SC font for SMALL CAPS and a OS font for old-style numerals; this is in addition to the four fonts for Roman, *italics*, **boldface**, and **boldface-italics**. Indeed, there are even more fonts for this single typeface to allow variation **in weight**, as well. This is not at all an atypical approach. A truly versatile typeface such as Adobe's Minion might well be sold as literally dozens of fonts.

OT and AAT both provide for support for old-style numerals and small cap variants within a single font file. Bitstream's Chianti, for example, allows one to set SMALL CAPS directly in the face through OpenType. Apple's Hoefler Text does THE SAME using AAT, as well as providing access to both **1215** and **\$1,215**; Adobe's Tekton Pro does BOTH using OpenType. (Hoefler also includes in the same font a set of ENGRAVED capital-letter glyphs for fancy effects.)

Typefaces—particularly calligraphic faces such as Linotype’s *Zapfino*—provide for non-contextual variant shapes. These would typically be swashes or other special shapes whose selection is entirely at the whim of the user (as opposed to variant shapes required by the writing system). Thus we could have *Zapfino*, *Zapfino*, *Zapfino*, or *Zapfino*.

Again, the traditional way to provide support for these is through multiple fonts for the typeface; Zapfino itself is currently shipped by Linotype as a series of four fonts which contain the same glyphic repertoire—no variation for boldface or italics—but with different shapes in each font. There is an additional font for the fancy ligatures and swashes.

Again, both OT and AAT provide for combining all these alternate glyph shapes in a single fonts.

Ligature Control


The ability to provide support for ligatures is also allowed through AAT and OT support. Alternate mechanisms for full ligature control is generally not available, however, because we aren’t looking for alternate shapes for a given character. On the Mac OS, because “fi” and “fl” are included in the standard character set, they are commonly used, but other ligatures are generally not made available.

The ability to use standard ligatures depends on the software and on the typeface. **Adobe InDesign** will automatically handle the formation of “fi” from “fi” if the font is properly set up. Hence we see “fish” in Legacy Serif but “fish” in Hoefler Text.

Other applications (such as **Microsoft Word**) provide no automatic support for ligature formation. The user has to explicitly select “fi” in order to see it.

Applications which provide for support for ligatures may do so by hard-coding assumptions about the character set—**Quark XPress** does this, as does **InDesign**. **InDesign**, however, also provides by default support for some types of ligatures as defined in the font’s OpenType data. This allows for some very interesting possibilities in Chianti. Thus do we see “Flashy Fish, @., Inc.” if we are so inclined. Note, by the way, that in Chianti, “fi” and “fi” are hard to tell apart; Bitstream has effectively removed from Chianti the fi-ligature required by the character set by making it look simply like “fi”. That is, in Latin, the choice of ligatures to provide in a typeface is typeface-specific.

This point must be borne in mind. There is not in Latin typography a set of ligatures which all typefaces must support in order to conform to user wishes. Setting “fish” as “fish” in Courier just doesn’t make much typographic sense, no matter what the user wants, whereas setting it as “fish”

does make sense in Chianti, as do “fish” and “fish” in .

Here, for example, are the ligature suites of the typefaces under discussion.

Tekton Pro (Adobe)
fi fl ff ffi ffl fj ffj tt

Chianti (Bitstream)

fi fl ffi ffi ff FI FL FI FL Cr Æ tt ch sh gg ck gy QU tʒ © € M^c Mc Qu LA LL F MD TT TY KA HE R EA ME
QU F R HE L MD ME QU TT TY ffi ft st ct

Hoefler Text (Apple)

fi fl fb ff fh fj fk ffb ffh ffi ffk flb fh fi fk fl ff ft ffi ct st fl

Zapfino (Linotype)

*fi fl fi fl fi fl fi fl th t uz nra Co M^{rs} Ltd Th di dr es es ex fff fl ft ge gg gʒ is ll ll ou ph ph
pp ri rt sh sp ss s st tt fi fl tt fi fl*

Courier

fi fl

Courier has exactly the two ligatures required for MacRoman.

Tekton Pro provides a simple set of standard Latin ligatures; Hoefler a somewhat larger set of standard ligatures such as one would find in high-quality Latin typography.

Chianti, however, designs the standard fi- and fl-ligatures away while providing a somewhat unexpected and more playful set.

Finally, Zapfino provides a rich set which includes not only a large number of variant shapes for the fi- and fl-ligatures, some “abbreviation” ligatures such as Chianti provides but a number of elegant forms for common character combinations. Tekton Pro does the same thing with a “tt” ligature.

Note the impact of this approach. Even in common English texts, the use of the “tt” ligature for words such as “chatter” look subtly if unexpectedly better. The effect is substantially more significant; *Apple* and *Microsoft* look better than *Apple* and *Microsoft*. Or maybe not. The typographer makes the default decisions; the user overrides them.

The fact that Roman typefaces will vary so much in the suite of ligatures they provide illustrates a difficulty with the ZWL approach to ligation control in Latin. A user could not be expected to know that, as a rule, Tekton Pro will look best if every time “tt” is found in a text, then should insert a ZWL to get the esthetically better “tt.” The keyboard or input method should not have the burden of knowing that if text is being typed in Tekton Pro, “t+ZWL+t” should be inserted into the text whenever the user types “t+t.” Certainly when typing early drafts of this paper, I was unaware of Tekton Pro’s “tt” ligature and would have been annoyed if I had been required to do anything to activate it.

The argument in favor of ZWL is that it can be inserted into the text where the user would like a ligature if it is available for a given typeface. The requirement that ZWL be inserted in text to form a ligature places an unreasonable burden on users or input methods/keyboards to provide default availability of unexpected ligatures provided by the typeface designer.

There is no requirement in Unicode that access be provided for every glyph in a font through plain text controls. Nor is there any requirement that any type designer provide support for every glyph the user may want. The set of desired ligatures may be small for some typefaces but large for others; this is an esthetic issue, and the fundamental arbiter for a typeface's esthetics is its designer.

Ligature control is generally obtained in OT and AAT systems by dividing ligatures into a series of classes, such as required, optional, and rare. Exactly how these are divided up depend on the typeface. For example, Hoefler marks “**fi**” and “**fl**” as required and “**ct**” as optional. This is not unreasonable. Most people typesetting an English document would expect to see the former two but probably not the latter. If someone *does* want the latter, they probably won't want to explicitly insert ZWL between every “ct” pair in order to see it. It would be much easier to select all the text, turn rare ligatures on, and have done with it, or turn rare ligatures on in the default style sheet.

Both OT and AAT systems allow complete ultimate control over which ligatures are actually found in a document to be the user's. A user who wants “Flashy Fish, Co., Inc.” in Chianti can get it. Displaying “fi” instead of “fi” throughout this document has been done by selecting the pair and turning ligatures off. There is no advantage to be gained by having a ZWL available in this case.

It is certainly true that adding ZWL support is possible within the AAT/OT framework. However, there are two issues.

One is that because of the limitations of the TrueType spec, ligature control through ZWL won't be available except through OT and AAT. If it is done through AAT and OT, then we have the difficulty of having two disparate models for ligature control operating simultaneously.

That is, it isn't simply a matter of revising fonts to flag a combination as “f+ZWL+i” as a potential candidate for ligation. Even if the font does so and the user provides “f+ZWL+i” in their text, “fi” won't show up unless the ligation feature is turned on. And “fi” may not turn into “fi” simply by deleting the ZWL.

This, by the way, is rather different from the situation regarding ZWNL. Care would have to be taken that ZWNL doesn't show up on the screen as a blank box unexpectedly, but as its insertion would *disrupt* the character stream expected for the ligature, it would have the effect of preventing a ligature formation whether or not the ligature typographic feature is turned on. It would still have the ramification of providing two different models for turning a ligature off. A user who wants to undo the action of turning a ligature off might be confused when selecting the Ligatures menu item has no effect on their particular “fi” pair.

One other point to keep in mind is that ligatures interact with typography in other ways. As the point size of the text increases, ligatures become less necessary and may be turned off altogether. As the text is stretched to fit on a justified line, ligatures become disruptive and eventually are broken up. It is difficult to see how these processes would work with a ZWL-based ligature.

When Will It Show Up in My Software?

The technology to support ligature control via smart fonts has been available for years, ever since Apple introduced QuickDraw GX. GX, however, did not win wide acceptance, for a number of reasons.

One of them is simply a performance issue. For most high-end word processing or typesetting applications, performance is a significant problem, and they have shown a historical willingness to cut down on the scripts they support in order to enhance that performance. Adoption of GX (or ATSUI) would allow them to support a number of languages they don't support, and to collapse existing font file suites into a single font file—but they can live without supporting Indic languages, and they can live with multiple-font-file suites. The less elegant way of providing typographic power is the price they are willing to pay for better performance.

There is now a shipping application that supports OpenType ligature controls, namely Adobe's **InDesign**. **InDesign** is implemented through a set of plug-ins, which means that either Adobe or another developer could enhance its OT support by writing a new plug-in. Thus, although the ligature control in the current version of **InDesign** is limited, that could change quickly. Moreover, Adobe is planning to enhance **InDesign** and use the CoolType engine it uses for rendering in future versions of its other products—thus, **InDesign** represents only the thin edge of the wedge in Adobe's OT support.

Microsoft is also starting to use OT support in its products, such as the latest versions of **Office**. Here, however, the initial support is limited to what is absolutely required for rendering of complex scripts. This doesn't reflect Microsoft's long-term strategy or goals, but it was a conscious and deliberate decision. The mere availability of the technology does not mean that applications will utilize it.

Apple has provided AAT functionality since the early 1990's. This support is becoming more pervasive on Apple systems, now that it is de-linked from QuickDraw GX. For example, future versions of Apple's Java engine, MRJ, will be based on AAT. Apple has also centered its new built-in text editing engine, MLTE, around AAT so that applications that use it will pick up AAT support automatically.

The point is that the promise of ligature control through smart fonts is less empty than it has been in the past. Over the course of 1999, there were substantial improvements along this front.

In any event, on both Windows and the Mac OS, it is true that whether ligatures are controlled via ZWL and ZWNL or through AAT and OpenType, that support will be available in most end-user applications only as they adopt AAT and OpenType. Legacy applications will not pick up ligature control via ZWL automatically, even if fonts are updated to support it. Other end-user applications which don't depend on system library support but use their own text layout engines will have to be rewritten in either case. (There is also the issue of how an application which doesn't use OT or AAT is going to know about the availability of a particular ligature in a font.)

ZWL will also require fonts to be revised—although this is a minimal problem as only now are font vendors starting to convert their fonts to use OT and few shipping fonts use AAT. Current “smart” fonts will need to have their smarts enhanced to know about ZWL and use it appropriately. There may also be revisions to system software. For example, the Mac OS includes code to guarantee that certain characters such as directionality overrides are not rendered by blank boxes even if the font

selected doesn't know how to render them. This will have to be revised. If Apple chose to guarantee that fonts without support for ZWL will work properly if ZWL is present in the text stream, considerable effort would be required.

(As a side note, adding OT and AAT support to TrueType fonts is relatively easy. Microsoft is working on its Visual OpenType Layout Table [VOLT] program for use on Windows, Adobe is providing tools for adding OpenType support to PostScript fonts, and Apple provides for free a tool to add AAT support to a font.)

In short, introduction of ZWL won't speed up support for advanced ligatures in Latin text and might actually slow down the process.

Other Scripts

*This document deliberately limits itself to discussion of Latin. The situation in Greek and Cyrillic is similar. In all three cases, the script itself does not require any ligation *per se*, although elaborate ligatures are desirable in certain writing styles or typefaces. In particular, the issue of whether or not particular runic scripts require ligature control in plain text is not addressed.*

Both OT and AAT provide a model for how ligature control can be achieved. This control is tied intimately to individual typefaces, based on the assumption that it is the type designer who best understands what ligatures should be available to end users. Using either technology, elaborate ligatures involving multiple characters can easily be provided and included in documents.

Summary

Ligation control is not needed for plain-text Latin, Greek, or Cyrillic. Adding ZWL (and ZWNL) would disrupt existing models for higher-level protocols which provide ligature control and present users with two different models for achieving the same end. Although existing applications that control ligatures are rare in the current market, this is changing and will continue to do so over the next few years. Adoption of ZWL by Unicode will not speed this process up.

Apple requests that the UTC reject the addition of ZWL to the standard and include it in its list of rejected proposals.