Using Unicode's Historic Scripts: Glyph Variants and Other Issues

David J. Perry

Version 1.0, updated September 14, 2017

Contents

Note: this is a work in progress. Everything included is up to date and correct, as far as I know, as of September 2017. I am still researching the question of locales and in future versions may have additional information in the Appendix, particularly in regards to Mac OS.

Please send any corrections or suggestions for improvement to hospes02@scholarsfonts.net.

I. Introduction

A. Some Background

At the start of the personal computer era (roughly 1985–1995) support for non-Latin scripts was very limited. For instance, since there was no standard for polytonic Greek, font makers replaced the usual 256 characters in a font with Greek letters and the various combinations of vowels with accents and breathings. The arrangement of characters differed from one font to another, making interchange of text difficult.

By the late 1990s, the introduction of TrueType fonts had broken the 256-character limit. Software companies had also begun to base their products on The Unicode Standard, a character encoding that made possible real improvements in script and language support. Unicode originally had space for about 65,000 characters but it soon became clear that this was not enough. The Standard was expanded by adding fifteen additional groups of 65,000 characters, called planes. The original Plane 0[1] is called the Basic Multilingual Plane and the later additions the Supplementary Planes, numbered 1–15. Plane 1, the Supplementary Multilingual Plane, is used for many historic scripts.[2]

A fundamental principle of Unicode is to *encode characters not glyphs*. The Latin letter 'g,' for example, may appear in various shapes such as g *g* g or 𝔤. These varying shapes, referred to as glyphs, are all representations of a single underlying entity, which has been encoded as LATIN SMALL LETTER G.[3] The practice of encoding characters not glyphs works reasonably well for modern languages but causes some issues for scholars. Epigraphers and palaeographers often want to display characters using the specific forms in which they appear in an inscription or a manuscript. The various shapes which a given character can assume are often referred to as glyph variants, particularly in the context of historic languages. It is highly unlikely that glyph variants will ever be officially added to Unicode, for several reasons; the issue has been discussed more than once by the Unicode Technical Committee. More will be said below about how to deal with this issue, but we should not expect hundreds of variants to be encoded.

Microsoft Word 95 was the first widely available word processor that supported Unicode, and Windows 2000 implemented true right to left, Unicode-based Hebrew. Since that time language support has expanded and become more sophisticated in many ways, including the ability to use characters in the Supplementary Planes of Unicode. Font makers learned about the Private Use Area (PUA), a Unicode block reserved for custom uses, and began to employ it for characters that were not yet encoded officially or were not appropriate for inclusion in Unicode.

[1] Programmers tend to think of numbers as going from zero to nine rather than one to ten.

[2] Unicode distinguishes ancient scripts, that have not been used for a millenium or more, from historic scripts, whose usage has ended more recently. Rather than repeat the phrase "ancient and historic scripts," I have used the term historic throughout this paper.

[3] Every Unicode character has an official name in all capitals, or shown in small capitals in running text. Every character also has a codepoint assignment of four hexadecimal digits for characters in the Basic Multilingual Plane (e.g., 106F) or five digits for those in the supplementary planes (e.g., 1030A).

A final development has been the adoption of 'smart font' technologies, of which the best known is OpenType (OT).[4] Smart fonts can contain features that change the appearance of text in many ways *without changing the underlying Unicode values*. One OT feature allows the use of oldstyle figures such as 256 rather than lining figures such as 256. If a user copied this number, regardless of its appearance in this document, and pasted it into another application, the standard Unicode digits would be placed into the file. (This would not be true if the font maker placed the oldstyle figures in the PUA rather than accessing them via an OT feature.)

B. Historic Scripts without Glyph Variants

If you work with one of the historic scripts in which glyph variants are not an issue, such as Gothic, things are reasonably straightforward. You need a word processor that supports characters in the supplementary planes, which many nowadays do, and a keyboard. Keyboards are discussed in Chapter VII; if glyph variants are not an issue for you, you can skip most of the intervening material, although you should read Chapter V if your work involves right to left processing (Lydian, for example). You might wish to read the rest of the discussion if you would like to understand some general font and character issues more deeply.

C. Old Italic: A Complex Situation

Most of this paper will deal with Old Italic, since it is the historic script with which I am most familiar. It is also one of the most complex cases, perhaps the most complex. What is said about Old Italic will apply, *mutatis mutandis*, to other scripts that require glyph variants.

As with many other historic scripts, the Old Italic characters have been encoded in Plane 1. However, scholars who wish to use the Old Italic block in their work on languages of ancient Italy face several difficulties caused by the fact that characters are heavily unified[5] and appear with a wide variety of glyph shapes.

Some people are still using fonts of the type described at the beginning of Part IA, in which the regular letters, punctuation marks, and symbols found on a keyboard have been replaced with various ancient characters; such fonts do not follow the Unicode Standard in any way. More recent offerings, such as Juan-José Marcos's Alphabetum, include the encoded Old Italic characters along with glyph variants in the PUA. The PUA offers a lot of room to store variants in a way that can be accessed by Unicode-aware applications but has some serious disadvantages, including complete lack of standardization. Some font makers are now providing variants both in the PUA and via OpenType features; see, for instance, the most recent version of George Douros's Aegean font, freely available from many download sites including FontSpace, or Athena Ruby, designed for Byzantine inscriptions.

[4] The other smart font technologies are Graphite, developed by SIL International, (Windows, Linux, and Mac OS) and AAT (Mac OS only).

[5] 'Unified' means that a character may be used in multiple languages, sometimes with a different appearance. For example, some Chinese characters are also used, differently shaped, in Japanese.

D. Limitations of the Private Use Area

Let's explore the limitations of the PUA in greater detail. In the case of Old Italic, one difficulty arises from the large number of variants. Look at the PUA of fonts such as Alphabetum or Aegean. Finding the glyph you need is not an easy task. Sorin Paliga has created a keyboard driver for Mac OS to work with the Old Italic characters in Alphabetum; some of the letters on the keyboard have as many as six levels (plain, shift, option, option-shift, etc.) associated with them. This keyboard is very clever and does provide access to all the Old Italic variants in Alphabetum, at the same time as it illustrates the complexity involved.

More importantly, a text that relies on glyphs in the PUA is meaningless without the original font. This is not so much an issue with PDF documents since fonts can be embedded in the PDF. It is now possible to download fonts automatically so that web documents can be viewed as intended by the author. But there is no guarantee that the font will be available in the future; without the font all the work would be lost. Furthermore, text prepared using variants in the PUA cannot be reused, at least without considerable correction, if the original font is not available or if one needs to use a different font (e.g., a publisher wants to use a font different from the one used by an author in preparing a manuscript).

Another important consideration is that text using PUA characters cannot be searched or sorted accurately. If one created a database of inscriptions, for example, one would certainly want users to be able to search for all instances of a given text. This is not possible with the PUA since a user cannot know in advance which variants might be used in the text for which she or he is searching. Asking users to do multiple searches with many different PUA values is not appropriate or realistic.

Finally, using variants in the PUA vitiates one of the important advantages of Unicode's principle of encoding characters not glyphs. The glyph R can represent not only OLD ITALIC LETTER R but also OLD ITALIC LETTER A or OLD ITALIC LETTER DE. A user who saw the R shape might not be sure which was meant. If the text was prepared using a PUA value for this glyph, there would be no way to tell. If the glyph was displayed using an OpenType feature, the user could check the original source (e.g., a Word or XML document) and determine the encoded value underlying the R. He or she could also copy the text from a PDF and paste it into another application, and the underlying Unicode character would appear.

For all these reasons I strongly believe that we should look for better solutions, although the existing PUA fonts may continue in use for some time.

Recommendation for the Old Italic Characters

While idiosyncratic 256-character fonts and fonts with variants in the PUA continue to function, they do not provide the true benefits of standardization and long-term stability that are offered by texts that make use of the official Old Italic characters.

The best option at this time for accessing glyph variants seems to be a combination of certain OT features and fonts tuned for one or more languages.[6] Note that this situation may (and almost certainly will) change as software evolves.

Organization of This Paper

While the topics discussed here are not the proverbial rocket science, they may seem complex to those who are not familiar with font and character issues. If you've read this far, you should have grasped some basic points and understood the advantages that moving away from the PUA will bring to scholars. I have organized the rest of this paper as follows.

Chapters II and III present what I see as the most effective and easiest ways to work with glyph variants at the present time. I have kept the discussion here as simple as possible without omitting anything important. These sections are intended to give you an overview of what it will be like to prepare texts using OT features. After reading these two sections, you might decide that you would like to explore this topic further, or you might decide it's not for you.

If you are interested in continuing, Chapter IV gives some additional information about OT features and Chapter V discusses how to use right to left text. Chapter VII deals with keyboards; it is the most technical section of the paper.

II. The Basics: A Keyboard and A Font

A. The Toolbox

In order to use the Old Italic range of Unicode, you need three things:

- A keyboard to enter the Old Italic characters. One is available (both Windows and Mac versions) from my website.
- A font that contains the official Old Italic characters plus glyph variants. In my Italica Vetus font[7] the glyph variants are available in the PUA and also through the OT Character Variants feature. Other fonts might use different OT features to access the variants.
- Software that supports the OT features used by your font. LibreOffice, version 5.3 or later, supports all OT features and is what I recommend for most users; to learn how to use OT features in LO, see the article "Using OpenType Features in LibreOffice" here. XeTeX or XeLaTeX also has complete OT and language support but the learning curve is steeper than with most other software. Microsoft Word and other office programs, both on Windows and Mac OS, support a more limited set of OT features (but which might be sufficient, depending on the design of the font you use).

[6] For convenience I will use OT features in this discussion, since they are the most widely supported, but the same features are available in Graphite fonts. AAT fonts are less often used now because Mac OS supports many OT features and AAT fonts are more difficult to develop than the other two types.

[7] Development of Italica Vetus was underwritten in part by the Script Encoding Initiative at the University of California, Berkeley, supported by a grant from the National Endowment for the Humanities.

B. Text Entry

When using OpenType features to select glyph variants, text entry is a two-step process. You enter the Unicode characters, preferably using a keyboard specific to the script with which you work, and then apply OT features to get the specific shapes you need.

You could, in fact, do without a keyboard and enter the characters via a dialog box such as LibreOffice's Insert/Special Character. That is very slow and cumbersome for all but the shortest texts; a keyboard is much better. Keyboards can also provide easy access to some characters needed by scholars such as U+0323 COMBINING DOT BELOW (for doubtful readings). They are easy to make or modify using free tools; see "Customizing Keyboard Layouts" on page 14 below. For help with adding a keyboard to your system, see "Installing Keyboards" on page 16.

How you apply OT features will vary depending on the software in use. Many programs provide a graphical interface with checkboxes, pulldown menus, or dialog boxes. LibreOffice (5.3 or more recent) takes a different approach, requiring you to type codes in the box where you select a font. (Each OT feature has a four-character code associated with it.) This process is explained in detail the article *Using OpenType Features in LibreOffice* (download here; also included in the Italica Vetus font package), so I will not repeat the information at this point.

III. Customized Fonts for Languages with Many Glyph Variants

The Unicode Standard says in §8.5 (page 347 in Unicode 9.0, page 349 in 10.0):

> The unification of these alphabets [Etruscan, Oscan, etc. - DJP] into a single Old Italic script requires language-specific fonts because the glyphs most commonly used may differ somewhat depending on the language being represented.

This innocuous-sounding statement is the tip of a large iceberg. Before exploring all the complexities that come from the unification of the Old Italic languages, we need to be clear about one point. We have already mentioned the Unicode principle of encoding characters not glyphs. The corollary to this is that Unicode does not care (much, anyway) what the glyphs in a font look like as long as they are suitable for representing the intended characters. *There is no requirement that glyphs in a given font look the same as the reference ones printed in* The Unicode Standard.

So, if you work with inscriptions in Faliscan, you might construct a font in which a few characters are drawn in shapes typical of Faliscan, where these differ from the generic OI shapes in Unicode. This is probably the kind of thing that the authors of *The Unicode Standard* had in mind when they wrote the passage cited above. Sounds good — no need to worry about selecting glyph variants from a general-purpose OI font. This kind of thing can work with some languages.

But not with Old Italic, unfortunately. Even in Faliscan there are a few letters with two different shapes. Things are much more complicated with Etruscan or Raetic that employ large numbers of variants. The moral is that you can make (or have made for you) custom fonts where the Unicode reference shapes are replaced by those more common in your work. This can save time, perhaps considerable time, but will never eliminate the need to choose variants via OT features.

IV. More about OpenType Features

This section provides additional information about OT features that might be used to facilitate the input of glyph variants.

A. Character Variants

Character Variants is the most suitable OT feature for languages that use large numbers of glyph variants. It is specifically intended to apply one of many variants to a single character at a time.[8] A font can include up to 99 CV features, and each feature can hold a large number of variants (there may be a limit, but I do not know what it is; I have personally worked with CV features holding more than 30 variants). Variants within a feature are typically accessed by number, although an application could provide a glyph palette for users to choose the shapes visually. Scholarly fonts that use Character Variants include Italica Vetus and Athena Ruby. The screen shot below illustrates the use of Character Variants in Serif PagePlus, a Windows desktop publishing application that has excellent support for OT features. (Unfortunately, it does not like characters in the supplementary planes.)

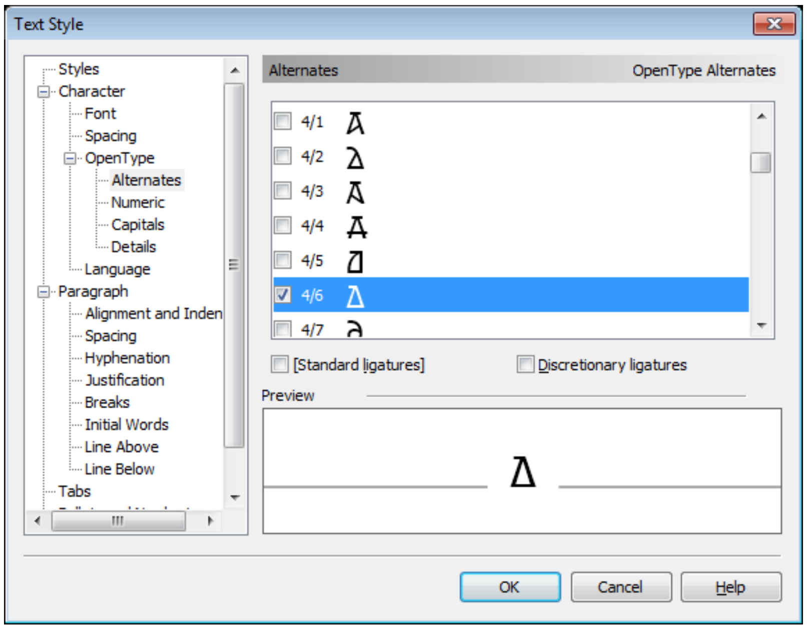


Figure 1. In a PagePlus document we have entered and highlighted a Greek capital Delta using the Athena Ruby font. We then choose Format / Character and this dialog box opens. Clicking the plus sign next to **OpenType** and selecting **Alternates** gives access to the variants. The Delta variants in this font are located in the Character Variants 04 feature and we are going to change the generic Δ to the sixth variant; hence the notation **4/6**. Some programs would only let users select the variants by number, but PagePlus helpfully displays the variant shapes.

As of August 2017, Character Variants are supported by relatively few applications: Serif PagePlus, LibreOffice 5.3 or higher, and XeTeX / XeLaTeX. We hope that this situation will improve in the future.

[8] Statements about the intended use of OT features in this and the two following sections are based on the descriptions in the OpenType Specification; see https://www.microsoft.com/typography/otspec/featuretags.htm.

B. Stylistic Alternates

This feature was designed to provide access to alternate glyphs for aesthetic effect; in other words, it was intended for graphic designers rather than scholars. It is similar to Character Variants in that it can include more than one alternate shape for a character, although some applications may apply only the first variant (if there is more than one). Support for Stylistic Alternates in applications is somewhat more extensive than for Character Variants, but it is still not as widely available as Stylistic Sets. Despite its original purpose this feature certainly could be used for variants needed by scholars. With Character Variants, it is easy to organize the glyphs; all the Alpha variants could go in <cv01>, all the Beta variants in <cv02>, and so forth. With Stylistic Alternates, everything has to go in one big list. This arrangement is possible but is more difficult for font makers to construct and for users to apply when there are large numbers of variants. Given this fact and the inconsistent implementations I suggest avoiding it; use either Character Variants or Stylistic Sets.

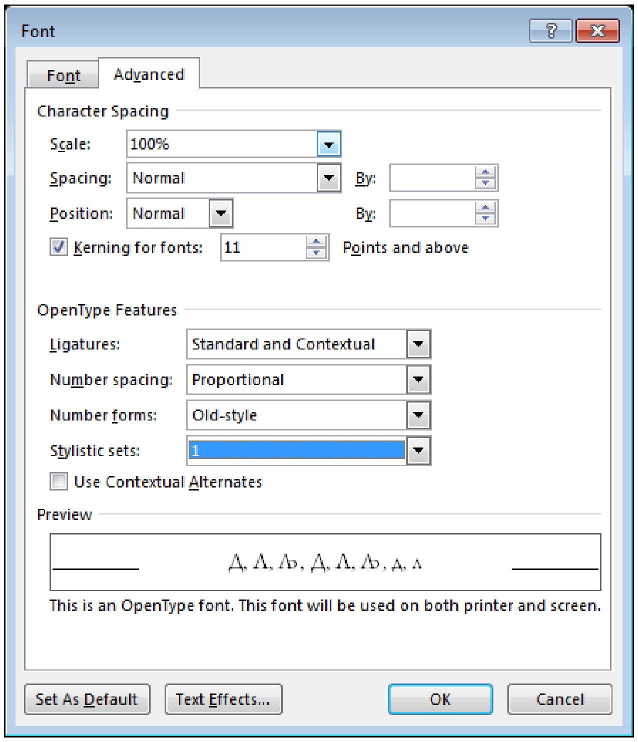
C. Stylistic Sets

A stylistic set is a group of shapes that are often used together. A font may contain up to twenty stylistic sets. As an example, the EB Garamond font contains triangular variants of some Cyrillic letters, accessed through Stylistic Set 1.

Д, Л, Љ, Д, Л, Љ, д, л and љ (default)
Д, Л, Љ, Д, Л, Љ, д, л and љ (Stylistic Set 1)

A designer who chose to set a Cyrillic text using the triangular forms would presumably use them consistently throughout. So she or he could select the entire text and apply Stylistic Set 1; of course, only certain characters would change their appearance.

Stylistic sets have better support than many other OT features in mainstream software (you can see in Figure 2 that they are one of only a few OT features available in Word). Stylistic sets are not a good solution where large numbers of variants are involved, but can work satisfactorily in languages with less complex requirements. Faliscan, for instance, has eleven letters that can appear in forms different from the Unicode reference glyphs, and six of these have a second variant. A font could contain two stylistic sets, one with the eleven first alternates and one with the second six; this would work well. Of course the user would apply the sets selectively, rather than to an entire text as in the Cyrillic example.


Font
Font | Advanced

Character Spacing
Scale: 100%
Spacing: Normal By:
Position: Normal By:
[✓] Kerning for fonts: 11 Points and above

OpenType Features
Ligatures: Standard and Contextual
Number spacing: Proportional
Number forms: Old-style
Stylistic sets: 1
[ ] Use Contextual Alternates

Preview
Д, Л, Љ, Д, Л, Љ, д, л
This is an OpenType font. This font will be used on both printer and screen.

Set As Default | Text Effects... | OK | Cancel

Figure 2. Stylistic sets in Microsoft Word are accessed through **Font/Advanced**. Notice that after selecting Stylistic Set 1, the Preview window shows how the appearance of the selected text will change after clicking on **OK**. The version of Word on my machine does not display the preview correctly for Old Italic characters, but the stylistic sets do work in the body of the document.

D. Other Options: Localized Forms and Variation Selectors

There are two other OpenType features, Localized Forms and Variation Selectors, that are sometimes suggested as a solution to the problems presented by scripts that are heavily unified. I do not believe that either offers a good solution for Old Italic, at least at the present time. I am discussing them here for those who are interested; you can skip this section unless you are looking for more in-depth information about character and font issues.

Localized Forms

If the user tags a run of text as being in a particular language, this feature can substitute varying shapes as appropriate. For example, the Junicode font for medievalists uses by default the Old English shape of the letter Thorn/thorn (Ꝥ/þ). But if a user wants the modern form (Þ/þ), she or he can tag the text as Icelandic and the shape will change since Junicode includes this substitution as a localized feature for Icelandic.

However, localized forms are not a good solution for languages with many glyph variants. They work well when there are two shapes involved, a default shape and one alternate, as with the Thorn example given above; the user marks text as a particular language and the appropriate shape automatically appears. Localized forms are not designed to handle situations where a user needs to choose among several variants. After tagging text as belonging to (let's say) Faliscan, the user will still need to do something manually to choose appropriate variants – apply an OT feature such as a stylistic set or Character Variants. Localized forms might be used to reduce the number of variants that had to be adjusted manually but would not automate the process completely.

There are also some technical challenges involved in using localized forms with historic scripts. I discuss these in the Appendix for those who are interested. They are difficult but not insurmountable. However, since localized forms are not the best solution when multiple variants are involved, there is no need to jump through these computer hoops.

Variation Selectors

In rare situations, certain languages require differing shapes to be used that cannot be determined from context or via any other normal mechanism. To deal with this situation, Unicode introduced Variation Selectors. Users enter a special variation selector (a character with no visible form; sixteen are available in the Variation Selectors block FE00–FE0F) followed by the character; the alternate shape is then displayed. Each combination of variation selector plus character must be approved by Unicode.

This mechanism certainly could be used for the glyph variants needed by scholars. However, the Unicode Technical Committee has made it clear that they see this as a last resort with limited use and are not willing to extend it to large numbers of variants in historic languages. (My sense is that the committee feels this would be opening the door to a major exception to the principle of encoding characters not glyphs.) So this is not a solution that will meet the needs of scholars, attractive as it might appear. I have seen one font in which the font maker used variation selectors to access variants in ancient languages. I cannot recommend this practice. Furthermore, it will not work in many applications that check whether the script in use has variation selectors approved for that script and refuse to apply them if not.

V. Issues of Directionality (Right to Left vs Left to Right)

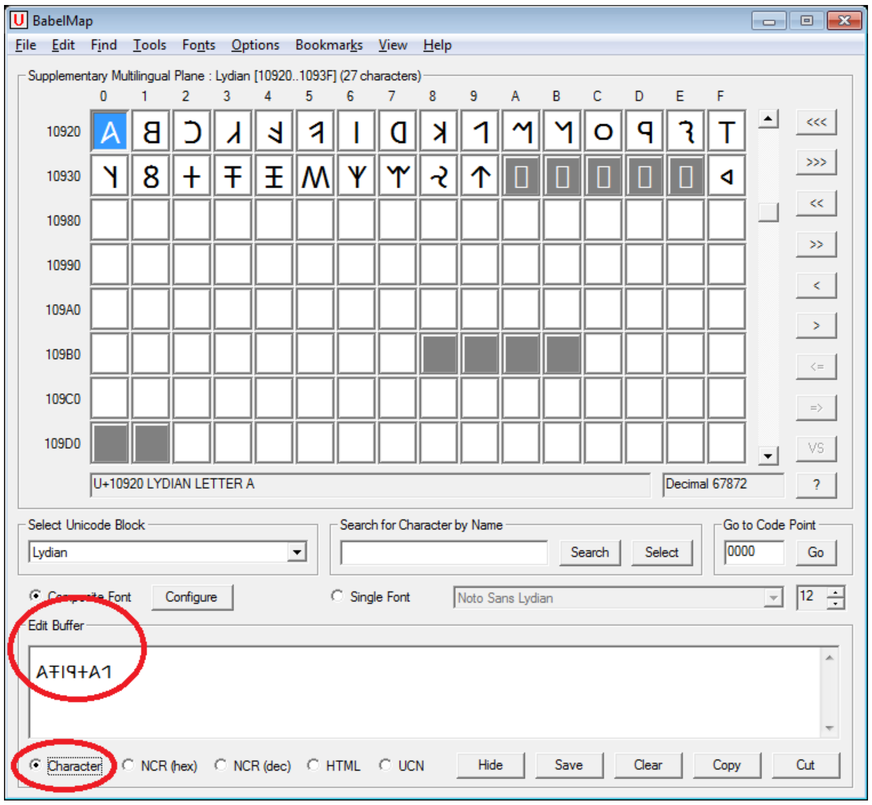
A. Introduction

Most characters in Unicode have a strongly defined direction, either right to left or left to right. The exceptions are mainly characters such as digits and punctuation marks that are used in multiple scripts.

Unicode includes a Bidirectional Algorithm to facilitate using both directions of writing in a single document. If you enter a series of Hebrew Unicode characters in a document that is mainly left to right, the Hebrew letters will automatically be placed right to left on the screen. This happens because Hebrew is encoded with a strongly right to left directionality; an application that supports the bidi algorithm properly knows what to do with them. Furthermore, if you type an opening bracket [when using RTL text, it will appear as] since that is the correct shape in a RTL context. (Word processors may provide additional tools for working with RTL text. The behavior described here is the minimum that bidi-compliant software should do.) The same things happen with ancient scripts such as Lydian that are encoded as RTL, even though word processors don't have any special tools for working with them. You can try this for yourself — just get a font that supports Lydian, such as Noto Sans Lydian or Everson Mono. You can copy and paste the characters from a utility such as BabelMap or use Word's Alt-x method (see this page for the Unicode values of Lydian). If things don't work correctly, the application's bidi support is buggy. See Figure 3 on page 11 below.

Some ancient scripts such as Carian and Old Italic were used to write texts in both directions but were encoded with a strongly left to right directionality in Unicode. This means that operating systems will treat them like the Latin and Cyrillic scripts that are always written LTR. In addition to the fact that texts were written in both directions using these scripts, modern scholarship often presents texts that were originally RTL in a LTR format. Unicode chose to make Old Italic and Carian LTR scripts because doing so meets the needs of many (perhaps a majority) of users and avoids some of the complexities involved in supporting true RTL text.

Sometimes scholars do want to present RTL texts in their original form. One can use custom 256-character fonts or Unicode fonts with RTL shapes in the PUA for this purpose. As explained above, these options do not provide the benefits of standardization that Unicode offers. You can also use a Unicode font where the glyphs are reversed (e.g., U+10304 𐌄 appears as Ǝ.) Remember that Unicode does not prescribe specific shapes; as long as the character is meant to encode OLD ITALIC LETTER E the font police will not come after you just because the glyph is flipped. Even so, this should be regarded as a stopgap — albeit perhaps the best solution available at the moment. Unicode does provide a mechanism to deal with this situation through the use of special directional formatting characters, as we will see in the next section.


BabelMap
File Edit Find Tools Fonts Options Bookmarks View Help
0 1 2 3 4 5 6 7 8 9 A B C D E F
10920
10930
10980
10990
109A0
109B0
109C0
109D0
<<< >>> << >> < > <= => VS ?
Search Select Go Configure Hide Save Clear Copy Cut

Supplementary Multilingual Plane : Lydian [10920..1093F] (27 characters)

U+10920 LYDIAN LETTER A

Decimal 67872

Select Unicode Block: Lydian

Search for Character by Name:

Go to Code Point: 0000

[✓ Composite Font] Single Font: Noto Sans Lydian 12

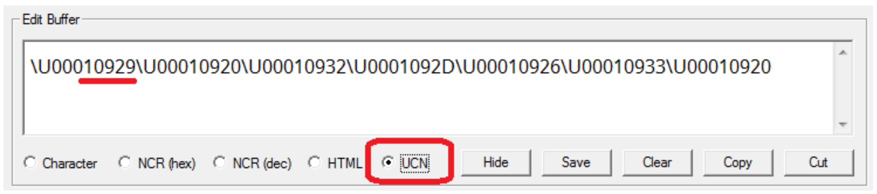
Edit Buffer

𐤩𐤠𐤲𐤭𐤦𐤳𐤠

[✓ Character] NCR (hex) NCR (dec) HTML UCN

Figure 3. In the BabelMap utility I selected characters for the Lydian word *laqrisa*; they appear in the Edit Buffer at the bottom (red circle). Even though I clicked on the letters in the order 𐤩 - 𐤠 - 𐤲 - 𐤭 - 𐤦 - 𐤳 - 𐤠, they automatically appeared in right to left order in the Edit Buffer. This happened because BabelMap is a Unicode-aware program that handles right to left scripts properly.

If I change the Edit Buffer to display the Unicode values of the characters, as shown in Figure 4, you can see the order in which the characters were entered and stored, sometimes called ‘logical’ as opposed to ‘visual’ order. U+10929 is LYDIAN LETTER L; it appears at the right in the Lydian script but at the left in the sequence of Unicode values.



Edit Buffer

\U00010929\U00010920\U00010932\U0001092D\U00010926\U00010933\U00010920

Character NCR (hex) NCR (dec) HTML [✓ UCN]

Figure 4. BabelMap’s Edit Buffer showing Unicode values not character shapes.

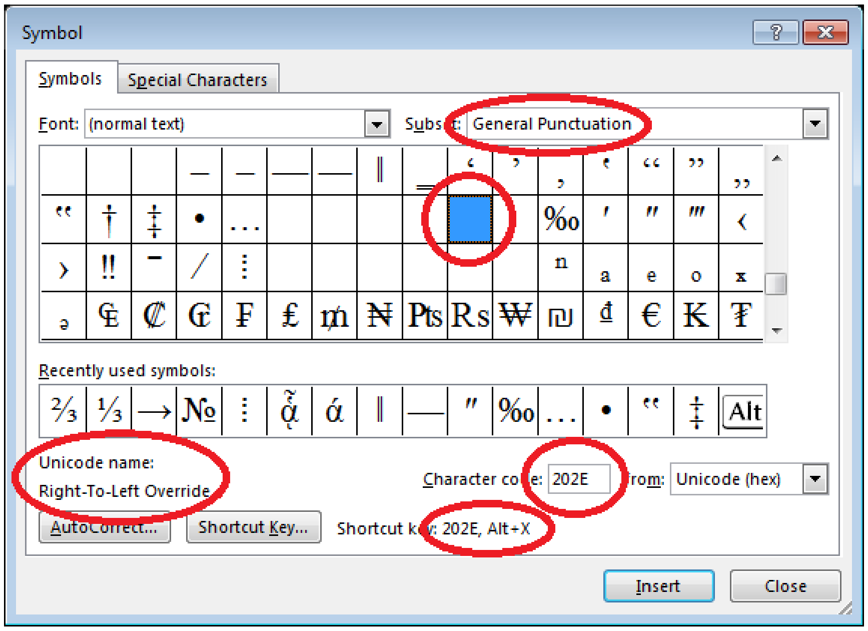
B. How to override directionality

In order to display text written in a strongly LTR script in RLT order, you must override the inherent directionality of the characters. Unicode provides special formatting controls for this. Here are the steps:

- Start your document using a modern language written LTR.
- When you reach the point where you want to place a chunk of RTL text, insert the control character U+202E RIGHT TO LEFT OVERRIDE (RLO) into your text. You can use the Alt-x method if this is available in your software, copy the RLO from a utility such as BabelMap, use the Unicode hex entry method on Mac OS, or insert it via a dialog (Word’s Insert/Symbol, e.g.) *provided that* the font you are using contains this character. The RLO belongs to the General Punctuation range, as shown in the Word screen shot in Figure 5 on page 12. (Many fonts do not contain this character. This does not matter; the operating system will still insert it if you use one of the other methods.)
    - o Note that the RLO has no visible representation. You can still copy, paste or delete it; you just have to watch carefully where you are on the screen and how the cursor

moves. For instance, if it appears that you have one space between words but it takes two taps of an arrow key to move the cursor over that space, this indicates the presence of an invisible formatting character.

- If necessary, change to a different font (if the font you were using for LTR text does not have the Old Italic characters in it).
- Change to a keyboard that will insert Old Italic characters, using the keystrokes provided by Windows or Mac OS for this purpose. (See page 14 for keyboard switching.)
- Type your text; you will notice that the behavior of the cursor keys changes if you make edits in the text you have typed.
- Switch back to your usual keyboard and insert another control character, U+202C POP DIRECTIONAL FORMATTING (PDF[9]), to return to LTR text entry. (If using the Alt-x method, be sure to switch away from your Old Italic keyboard *before* typing 202C!) If necessary, change the font. Ending a paragraph with the ↵ key also terminates the effect of the RLO.



Font: (normal text)
Subset: General Punctuation
Unicode name:
Right-To-Left Override
Character code: 202E from: Unicode (hex)
Shortcut key: 202E, Alt+X

Figure 5. Using Word's Insert/Symbol dialog to enter the RLO control character.

A keyboard customized for Old Italic or other historic scripts can include keystrokes to enter RLO and PDF controls easily. If you use software that supports macros, such as LibreOffice Writer or Word, you can record a macro to enter the RLO, switch to a different font (if necessary) and apply an OT feature. A second macro could enter a PDF and switch back to the original font.

Unfortunately, not every program implements the RLO/PDF controls as it should. On Windows, the simple Notepad editor has worked correctly for many years and is an easy way to test. Word has has been cooperative since version 2010 (Windows); as I recall, earlier versions

[9] Not to be confused with the Portable Document Format, originally created by Adobe Systems, that makes it easy to display documents correctly on any operating system.

were not but I can no longer test them. LibreOffice 5.4 also works, but not 5.3 or any earlier version (exception: 5.3 is fine on Linux). On Mac OS, TextEdit, Nisus Writer, LibreOffice, and Word 2016 work properly. Scribus, the open source layout program, works correctly on all platforms, as do the Firefox and Chrome browsers.

If your software supports the use of RLO, feel free to use it if you like. Some people find the changed behavior of the cursor keys difficult to get used to when editing. But many inscriptions from ancient Italy are very short and typing text backwards, while klutzy, does work. Also read the following section, which may affect your decision about whether to use the RLO control.

For longer inscriptions, note that if you have at least one RTL language enabled in Word or LibreOffice, you can change the direction of an entire paragraph.

C. Mirrored shapes

Many characters require mirrored shapes when used in RTL settings; the diagonals of 𐌊 OLD ITALIC LETTER KA, for instance, should point to the left rather than to the right. Fonts designed to support RTL text must include mirrored forms of all characters that need them. Such mirrored glyphs can be given assignments in the PUA, but that is not a best practice.

Users might expect that mirrored shapes would be applied automatically after a RLO was inserted. There is an OT feature called Right to Left Alternates (rtla) that is designed to do just this.[10] And, in fact, the OpenType Specification 1.8.2 says that when a layout engine designed to implement the Unicode Bidirectional Algorithm encounters a run of text that needs mirrored shapes (as after an RLO has been inserted) it should apply Right to Left Alternates (reference here).

As of September 2017, the number of applications that behave properly is small: TextEdit and Nisus Writer[11] on Mac OS[12], Scribus 1.5.3 and XeTeX on all platforms, LibreOffice 5.3 on Linux / 5.4 on Mac OS and Windows, and several web browsers. If you have a Mac and want to see how this should work (even if the word processor you normally use doesn't support automatic mirroring), try TextEdit with a font that contains the <rtla> feature. It really does work!

Other methods will have to be employed until more applications implement the Bidirectional Algorithm properly. One option is to change to a separate font. A potential advantage with this method is consistency. If one is in the habit of accessing a specific variant via the OT

[10] In fact, there are two such features: <rtla> and another, <rtlm>, Right to Left Mirrored Forms. There is some debate among font makers as to which is preferable, but they both work if the application implements automatic mirroring; users need not worry about this.

[11] Any other word processor that relies on the text services provided by Mac OS, rather than doing its own thing, will probably work also.

[12] Testing done using 10.11.6 (El Capitan).

Character Variants feature, for example, the feature can work the same in the sinistroverse and dextroverse versions of the font, and one can switch back and forth with a simple font change. This what I did the when creating version 1 of the Italica Vetus font, which comes in a separate version, Italica Vetus SN, with sinistroverse glyphs.

Another option is to provide OT features such as stylistic sets to access the mirrored glyphs. For instance, a font designed to support Faliscan needs to support only a small number of glyph variants. In this situation a stylistic set would be appropriate and easy to construct.

VI. Conclusions

Before turning to the more technical issues involving keyboards, it might be useful to review the main points made in this paper.

- There is no single, easy solution to the issue of glyph variants in Old Italic and other historic scripts. This is partly because the issue is complex, due to the large number of variants and languages involved, and partly because software makers do not pay as much attention to historic scripts as they might. But solutions are available.
- It is time to move away from the PUA in order to gain the benefits of standardization, easy searching and sorting, and easy reuse of text that Unicode offers.
- One large pan-OI font, such as Italica Vetus, can be used with software that supports the OT Character Variants feature (or, as a fallback, with the PUA assignments).
- Fonts customized to support particular language(s) can simplify text entry by requiring less frequent selection of variant shapes and by using OT features such as stylistic sets.
- The mechanisms designed to override strongly LTR scripts such as Old Italic are not widely supported, particularly when it comes to automatically applying mirrored glyph shapes. Using a font that contains reversed shapes (preferably accessed through variants of the encoded Unicode values rather than through the PUA) may be the best solution when dealing with large numbers of variants; in simpler situations, Stylistic Sets may work well.

VII. Keyboards in Detail

A. Keyboard Basics

An appropriate keyboard driver makes it possible to type characters not found on a keyboard designed for use with the script that one generally uses (Latin, Cyrillic, or many others). Otherwise, one must resort to things like Word's Insert/Symbol, which are very cumbersome for inserting more than one or two characters.

Some users do not realize how easy it is to switch among languages and keyboards. Operating systems provide keystrokes to facilitate this.

- On Windows:
  - With Windows 7 and earlier, Alt-⇧ moves from one language to another (English to Greek, for instance); if more than one keyboard is installed for a language,

    Ctrl-⇧ cycles among them. (Use the keys on the left side of the keyboard for this.) On my computer I have English with four keyboards, Greek, and Hebrew.
    - With Windows 8.1 and more recent, use ⊞-Spacebar. The earlier methods also work if you are accustomed to them.
- On Mac OS, two shortcuts make it easy to switch layouts:
    - To move through the available keyboards in sequence, use ⌥-⌘-Spacebar.
    - To toggle back and forth between the current layout and the one previously in effect, use ⌘-Spacebar.
        - Note: these shortcuts are assigned by default to Searchlight in recent versions of Mac OS. You can use them to switch keyboards by opening Keyboard in System Preferences, clicking the Shortcuts tab then Input Sources at the left, and checking the two options.

When you enable an additional language that the operating system knows about, you choose which keyboard(s) you want; for instance, when you add Greek you can choose to use a monotonic keyboard, a polytonic one, or both. There are many internet resources that provide directions for how to do this, so I will not repeat the information here.

Historical languages, however, do not appear in the list of languages that you can easily add to your system. What if you need Gothic or Runic or Etruscan? The answer is: "Cheat!" It may seem odd at first to be typing an Oscan inscription while the bar at the bottom of your word processor's screen says 'English (United States)' or whatever. There are three important facts:

- You can make a customized keyboard for Old Italic or any historic script and associate it with whatever language you typically use, and the operating system does not care.
- To do it "right" — that is, get the word processor to display 'Oscan' rather than 'English (United States)' — is a lot of work. You have to create a custom locale, which can be done on Windows 8.1 or later. It is not easy. And you probably won't get 'Oscan' but rather a symbol for an unknown language. If you really want to know more about locales, see the Appendix.
- Most importantly, there is no reason to do all this work because *there is no benefit to doing so*. When you change from one common language to another, applications automatically switch to the appropriate spelling dictionary, hyphenation rules, and grammar checkers. These things do not exist for historic languages. Furthermore, if you type some text in Latin, the application may put red squiggles under each word because it think they are all misspelled. Since Old Italic, Gothic, Runic, etc. are separate scripts, this does not happen. So there's no reason to bother making custom locales.

B. Customizing Keyboard Layouts

Creating your own keyboard is not difficult. The programs described below are all reasonably easy to learn and they all let you modify an existing keyboard rather than starting from scratch, if that is appropriate for your needs. Before starting such a project, it is helpful to have the Unicode values of the characters you want to place on your new keyboard handy.

On Mac OS, use Ukelele, a keyboard editor developed by SIL International and freely available from this page. It comes with an excellent user manual that gives you all the information you need. See also the discussion of Keyman below.

There are several options available for Windows users.

The Microsoft Keyboard Layout Creator (MSKLC) is a free utility from Microsoft that you can download from this page. It is no longer actively maintained but still works and is easy to use. It has one limitation: it supports only one level of dead keys. So, for instance, you cannot use it to create a Greek polytonic keyboard where you type a deadkey for a breathing mark followed by another for an accent. (This is actually a limitation of Windows keyboards rather than MSKLC itself.) Also, some users have reported a problem with MSKLC on Windows 10; the keyboard grid where you enter the characters you want is not visible on screen (although it is actually running). Because MSKLC is no longer supported, it is unlikely that this problem will be fixed.

Kbedit is perhaps the best tool at the moment for creating native Windows keyboards, particularly for those who need more power than than MSKLC provides or who can no longer run it. It is available in a free personal edition as well as a premium version. It works well with Unicode, including characters in the supplementary planes, and provides many excellent features such as more sophisticated handling of deadkeys than MSKLC offers.

A web-based keyboard editor for Linux is here (I have not tried it).

Finally, there is Keyman, originally developed by Tavultesoft and now made freely available by SIL International. It is available for Mac and Windows. It is different than the other products mentioned here in that it does not create a native keyboard driver for the operating system, but one that utilizes Keyman's own format. One must therefore run a small program, Keyman Desktop, on one's computer that enables Keyman keyboards to work. The advantage to this system is that the Keyman format is very powerful; it supports (for instance) multiple deadkeys and many other advanced features and can work across platforms. To develop your own keyboard with Keyman, see the Developer page.

C. Installing Keyboards

Detailed instructions for installing keyboards and related issues, such as the Language Bar in Windows or the Input Menu in Mac OS, are given in the documentation for my Old Italic keyboards. Go here, scroll down to the keyboard section, select either the Windows or Mac version, download the ZIP file, and read the instructions (you don't need to install the keyboard unless you wish to).

Kbdedit provides built-in facilities for working with the Windows Language Bar; see this page of the kbdedit manual.

Note that keyboards on Mac OS may come as two separate files, a keyboard layout and an icon, or as a single bundle. Installation is the same either way. For those on Mac OS, the Ukelele manual also discusses this topic in detail.

Appendix: More about Locales

A. What is a Locale?

A locale is a collection of cultural norms typically used by people living in a certain area. Language (including spelling and grammatical rules) is chief among them, but a locale can also include things such as currency symbols, time and date formats, and whether to use a period or a comma as a decimal separator. Locales begin with a language identifier and often have an additional geographic location shown in parentheses, such as "French (Canada)" or "English (United Kingdom)." They may also be shown with two-letter abbreviations when space is at a premium, such as "EN-US."

{For next version: screenshot here?}

B. Why Locales Matter

Locales are useful in several ways with modern languages, but here I will focus on one aspect that is relevant to readers of this paper. I created a test font in which the OpenType Localized Forms feature (see page 9 above) was implemented for Faliscan under the Old Italic script. Despite considerable efforts I could not get the feature to work with any software I had available. Finally I constructed a custom Windows 10 locale for Faliscan; when it was in effect, the Localized Forms worked as intended.

As explained in Chapter IV, Localized Forms is not particularly suitable for languages that make use of multiple glyph shapes for a single character. But even if Localized Forms is appropriate to the needs of a particular ancient language, it will not work properly — assuming that my test Faliscan font reflects the general situation — unless Windows associates the font with a locale. It is not reasonable to expect users to go around constructing custom locales. For those who are interested, though, I will give some information in the next section.

[For next version: add info about Mac OS here; it's not quite the same as Windows]

C. Making Custom Locales

It is possible to create a custom locale if the language and script in question have been officially registered. Script codes are contained in the OpenType specification and language codes are defined an ISO 639-3, an international standard. See section D for a listing of some of these codes for historic languages.

[For next version: add more details about how to construct locales]

D. Some Codes for Historic Languages

scrp	four-letter script codes	
	lng	three-letter language code
cari	Carian script	
	xcr	Carian
goth	Gothic script	
	got	Gothic
ital	Old Italic Script	
	ett	Etruscan
	osc	Oscan
	npr	North Picene
	spx	South Picene
	xcc	Camunic
	xcg	Cisalpine Gaulish
	xfa	Faliscan
	xlp	Lepontic
	xrr	Raetic
	xum	Umbrian
	xve	Venetic
latn	Latin script	
	ang	Old English (Anglo-Saxon)
	lat	Latin (la in earlier versions)
lyci	Lycian script	
	xlc	Lycian
lydi	Lydian script	
	xld	Lydian
runr	Runic script	
	what languages?	
zzzz	unencoded script	
	zz private use	

Change Log

Sept. 14, 2017	Version 1.0	Initial posting on web site.