Εὔτυπον, τεῦχος № 20 — Ἀπρίλιος/April 2008

Unicode support for the Greek LGR encoding

Werner Lemberg

Municipal Theatre of Koblenz
Germany
E-mail: `wl@gnu.org`

Up to now, only the ucs package provides Unicode support for Greek. This article describes new support files for the LGR encoding which does the same (and even more) for LaTeX's default inputenc mechanism. The files described in this article can be found at http://www.latex-project.org/cgi-bin/ltxbugs2html?pr=babel/4015.

1 Introduction

While writing an article for the *Asian Journal of TEX* I tried to typeset the name Γιάννης Χαραλάμπους. LaTeX's inputenc package returned an error, reporting that no proper Unicode support for the Greek script was available. I was quite surprised. Doing a search in the internet I found out that indeed nobody had written a file `lgrenc.dfu` (see below). Being a perfectionist, I decided to implement complete Unicode support instead of using some hacks to enter this very name – since I had already defined the T5 encoding for Vietnamese the whole topic was not new for me, and things went rather straightforward. The stress lies on 'rather' since the LGR encoding has some tricky details which are described below.

2 The Unicode model of polytonic Greek

In Unicode [2] there are three blocks which are relevant for polytonic Greek: U+0370–U+03FF (Greek and Coptic), U+0300–U+036F (Combining Diacritical Marks), and U+1F00–U+1FFF (Greek Extended).[1] Two encoding models are provided: combining character sequences (using elements of the first two blocks) and precomposed base plus diacritic combinations (using elements of the second and third block).

[1] Recent Unicode versions added ancient Greek numbers at U+10140–U+1018F and characters for ancient Greek musical notation at U+1D200–U+1D24F. Currently, the inputenc package doesn't support Unicode values larger than U+FFFF.

ἄ = U+03B1 α + U+0313 ᾿ + U+0301 ´

ᾧ = U+03C9 ω + U+0314 ῾ + U+0342 ῀ + U+0345 ͺ

ᾋ = U+0391 A + U+0314 ῾ + U+0300 ` + U+0345 ͺ

Table 1: Examples of combining character sequences in Unicode for Greek. Note how the visual order in the last line differs from the logical order.

A combining sequence consists of a (spacing) base character followed by one or more (non-spacing) diacritics. In the case of Greek, an inside-out and left-right model is used for multiple diacritics: If the diacritics are stacked vertically, the first one in a character sequence is positioned next to the base glyph, and the second one is placed above the first one. If the diacritics are positioned horizontally, the left one is encoded before the right one. Finally, diacritics below the base character follow the diacritics above the base character, again stacked inside-out.[2]

3 The `LGR` encoding

The very reason for defining the `T1` encoding in 1990 during a TUG meeting in Cork, Ireland, is described in one of the documentation files for the EC fonts as follows:

> The design goals of the Cork encoding are to allow as many languages as possible to be hyphenated correctly and to guarantee correct kerning for those languages. Therefore it includes many ready-made accented letters.

Transferring this to the `LGR` encoding means that (at least) all of the precomposed Unicode characters should be included in the encoding so that proper hyphenation and kerning for polytonic Greek is available.

Unfortunately, there is an obstacle: The number of such characters together with the number of Greek base characters exceeds 256, no longer fitting into a single TEX font. The compromise used in the `LGR` encoding is to have an almost complete precomposed set for lowercase characters only; vowels with prosgegrammeni and dialytika are the only precomposed uppercase combinations.

3.1 Input and output ligatures

There is a difference between *input* and *output* ligatures;[3] The former is just a convenience to have easier access to some glyphs within an encoding; an example is the ASCII sequence '`---`' to create '—', the em-dash glyph, which

[2] That diacritics below the base character follow the ones above the base character in the combining sequence is not mandatory but a convenience.

[3] These are the terms I use to distinguish them; maybe other authors call them differently.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	—	⁀	[Δ]	[H]	[X]	[M]	ϛ	ϛ	ι	$A_ι$	$H_ι$	$Ω_ι$	Α	Ϋ	α	ϋ
1	ˏ	ˎ	ϟ	ϙ	‿	Ϙ	Ϡ	ƛ	€	‰	ə	ϡ	ʽ	ʼ	˘	¯
2	῁	!	¨	΅	῭	%	·	′	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	·	῾	=	᾿	;
4	῟	Α	Β	῝	Δ	Ε	Φ	Γ	Η	Ι	Θ	Κ	Λ	Μ	Ν	Ο
5	Π	Χ	Ρ	Σ	Τ	Υ	῞	Ω	Ξ	Ψ	Ζ	[	῏	]	῎	῍
6	`	α	β	ς	δ	ε	φ	γ	η	ι	ϑ	κ	λ	μ	ν	ο
7	π	χ	ρ	σ	τ	υ		ω	ξ	ψ	ζ	«	͵	»	˜	—

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
8	ὰ	ἁ	ἀ	ἃ	ᾲ	ᾁ	ᾀ	ᾃ	ά	ἅ	ἄ	ἂ	ᾴ	ᾅ	ᾄ	ᾂ
9	ᾶ	ἇ	ἆ	ϝ	ᾷ	ᾇ	ᾆ	‿	ὴ	ἡ	ἠ		ῂ	ᾑ	ᾐ	
A	ή	ἥ	ἤ	ἣ	ῄ	ᾕ	ᾔ	ᾓ	ῆ	ἧ	ἦ	ἢ	ῇ	ᾗ	ᾖ	ᾒ
B	ὼ	ὡ	ὠ	ὣ	ῲ	ᾡ	ᾠ	ᾣ	ώ	ὥ	ὤ	ὢ	ῴ	ᾥ	ᾤ	ᾢ
C	ῶ	ὧ	ὦ	Ϝ	ῷ	ᾧ	ᾦ		ὶ	ἱ	ἰ	ἳ	ὺ	ὑ	ὐ	ὓ
D	ί	ἵ	ἴ	ἲ	ύ	ὕ	ὔ	ὒ	ῖ	ἷ	ἶ	Ϊ	ῦ	ὗ	ὖ	Ϋ
E	ὲ	ἑ	ἐ	ἓ	ὸ	ὁ	ὀ	ὃ	έ	ἕ	ἔ	ἒ	ό	ὅ	ὄ	ὂ
F	ϊ	ῒ	ΐ	ῗ	ϋ	ῢ	ΰ	ῧ	ᾳ	ῃ	ῳ	ῥ	ῤ		ʹ	͵

Table 2: The LGR encoding of grmn1000.

has a ligature-independent representation as `\textemdash`. Output ligatures, however, are typographical features to 'improve' (in the broadest sense) the display of certain glyph combinations. The standard example is the 'fi' ligature, used to avoid the clash of the bulb of letter f with the dot of letter i.[4] For Unicode input, no input ligatures should be necessary at all – it is basically the job of the editor program to convert input key sequences to proper characters.

In TEX, input and output ligatures can't be distinguished; both are implemented on the font level (which is generally considered as a design error, but see section 7 for unexpected benefits).

The `LGR` encoding uses a rich set of input ligatures to map ASCII sequences onto the various glyphs;[5] it has no output ligatures. Being able to input Greek entirely with ASCII characters guarantees compatibility with any TEX macro format and document encoding, and the used representations can be easily memorized.

However, there is a serious drawback of the `LGR` ligatures: Some diacritics must be input before the base character (psili, dasia, oxia, varia, dialytika). In combination with the fact that TEX's input convention is exactly the opposite of the one Unicode expects (TEX always needs the `\accent` primitive before the base glyph), there is no possibility to handle Unicode combining sequences for Greek, and we can support precomposed Unicode characters only.[6]

4 The ucs package

Usually people are using the ucs package developed by Dominique Unruh [1] to write Greek in Unicode encoding. Unfortunately, ucs is incompatible with inputenc, and not all packages cooperate properly with ucs. Additionally, ucs is no longer developed and maintained. On the technical level, ucs partially uses double accents like ῎ directly, partially it relies on the input ligature mechanism of `LGR`. In file `ucsencs.def` it extends the very rudimentary declarations in Babel's `lgrenc.def` to make it a full-featured encoding definition file; the Unicode mappings are in files `uni-3.def` and `uni-31.def`. The support files I present in this paper have almost the same structure.

Similar to all other entities in the ucs package, the prefix 'text' is used to address Greek characters, for example `\textalpha` or `\textKappa`. Multiple diacritics are handled together so that only one 'accent cluster' is applied the base character. Example: `\textpsiliperispomeniiota\textAlpha`.

---

[4] In Turkish, for example, the 'fi' ligature is not used normally since this language uses a dotless i also. Compare 'fi', 'fi', and 'fi' (this is the Times Roman font). Other languages like Portuguese also avoid the 'fi' ligature.

[5] It would be silly to explain Greek input ligatures in a Greek magazine.

[6] Using Ω's OTPs (Omega Translation Processes), or something similar in the forthcoming LuaTEX this problem can be solved: The characters in the data stream get reordered before the TEX engine is handling them. A different approach provides XeTEX (and LuaTEX): Since it supports direct access to OpenType features, a sequence 'base character' + 'diacritic(s)' can be handled by the font itself without using TEX ligatures, automatically providing the correct glyph shape.

5 The inputenc package

Documentation for the inputenc package and the UTF-8 support in particular is part of the base LaTeX bundle [4, 5].

The main missing file in the current Babel distribution is `lgrenc.dfu`, providing Unicode mappings for (almost) all glyphs of the LGR encoding. However, before creating such a file it is necessary to define encoding-independent entities which can be used for such a mapping; as mentioned in the previous section, this is done in file `lgrenc.def`. The next subsections describe some details of the new version I provide.

5.1 Implementation details

Since the LGR encoding is heavily based on input ligatures I first tried to use them for the definitions of the entities in `lgrenc.def` also. However, I soon found out that TeX can only handle either an input ligature or a kerning operation but not both at the same time; using input ligatures would thus mean that kerning between the previous character and the ligature itself fails. Note that this affects only input ligatures which don't start with a base character – ligatures with a trailing '`|`' character for the ypogegrammeni has no negative effect.

All definitions in the original file have been retained; I won't go into the details of trivial additions like `\textemdash` or `\guillemotleft` for non-composite glyphs. For macros specific to Greek I decided to use the prefix 'grk'.

Unaccented Greek characters have names like `\grkl` (λ) or `\grkOM` (Ω). The only interesting cases are the definitions of sigma (σ) and the final sigma (ς):

```
% this sigma glyph changes shape -- \grksigma doesn't
\DeclareTextSymbol{\grks}{LGR}{115}
\DeclareTextCommand{\grksigma}{LGR}{\grks\noboundary}
\DeclareTextSymbol{\grksfinal}{LGR}{99}
```

Slot 0xFD in LGR is an empty glyph, used internally as the bounding character to indicate word endings. For ASCII input it is very convenient that LGR contains ligature rules to automatically convert a sigma at the end of a word to a final sigma character – you always type character `s` and you get the right shape. However, as stated earlier, this should be done on the editor level for Unicode input, thus we need a macro which always expands to a non-final sigma. The rarely used TeX primitive `\noboundary` does exactly what we want, suppressing a ligature with the boundary character.[7]

The dialytika (¨), oxia (´), and varia (`) accents are mapped onto `\"`, `\grkoxia`, and `\grkvaria`, respectively.[8] Because `\~` is special in the Greek

[7] LGR provides a set of ligatures to another empty glyph slot at position 'v' (0x76) so that you can say '`sv`', for example, to get a sigma which never changes to the final form. See the discussion in the next subsection why this approach is not suitable for proper cut and paste support.

[8] Since the tonos accent has the same shape as the oxia, `\grkoxia` is used for it also.

language definition of Babel (in file `greek.ldf`), I selected `\grkperis` for the perispomeni (῀). Dasia (῾), psili (᾿), and the subscript iota (that is, both prosgegrammeni 'ι' and ypogegrammeni 'ͺ') are mapped onto `\grkdasia`, `\grkpsili`, and `\grksubiota`, respectively. All accents are defined with the standard `\DeclareTextAccent` command; the exception is `\grksubiota` which uses `\DeclareTextCommand` to map to an input ligature:

```
\DeclareTextCommand{\grksubiota}{LGR}[1]{#1|}
```

`lgrenc.def` also contains commands for (almost) all possible combinations of diacritics like `\grkdialtonos` for ΅. Again, they are defined with `\DeclareTextAccent` except combinations with the subscript iota.

Non-accent entities which are composite can now be easily defined with `\DeclareTextComposite`, for example

```
\DeclareTextComposite{\grkpsilivaria}{LGR}{\grka}{"8B}
```

The set of definitions is rounded up with entries for uppercase composite characters; since they don't map to glyph slots, we have to use `\DeclareTextCompositeCommand` like this:

```
\DeclareTextCompositeCommand{\grkpsili}{LGR}{\grkE}{>E}
```

Please, keep in mind that most of the macros defined in `lgrenc.def` are not intended for manual input but for proper definitions in `lgrenc.dfu`. However, the following macros are probably of greater interest since `greek.ldf` doesn't provide equivalents:

`\grkFifty` 𐅄
U+10144 GREEK ACROPHONIC ATTIC FIFTY
`\grkFiveHundred` 𐅅
U+10145 GREEK ACROPHONIC ATTIC FIVE HUNDRED
`\grkFiveThousand` 𐅆
U+10146 GREEK ACROPHONIC ATTIC FIVE THOUSAND
`\grkFiftyThousand` 𐅇
U+10147 GREEK ACROPHONIC ATTIC FIFTY THOUSAND

`\grkStigma` Ϛ
U+03DA GREEK LETTER STIGMA (variant)
`\grkKoppaold` Ϙ
U+03D8 GREEK LETTER ARCHAIC KOPPA
`\grkkoppaold` ϙ
U+03D9 GREEK SMALL LETTER ARCHAIC KOPPA
`\grkSampi` Ϡ
U+03E0 GREEK LETTER SAMPI

5.2 Issues with `\MakeUppercase`

A specialty of Greek is the handling of phrases completely typeset with uppercase letters: All diacratics except the subscript iota and dialytika are discarded. To do this, the `LGR` encoding has a third empty glyph slot at position 0x9F so that the `\uccode` value of the accents can be set to it. For example, `\uppercase{<~a|}` (ᾇ) is converted to `^^9f^^9fA|`, and which looks like `A|` (Aͅ).

While this approach works well with `\uppercase`, it has two drawbacks:

- It is necessary to modify the uccode values of some punctuation characters, making it a fragile operation if used outside of a Greek language environment – remember that the original TEX version applies only one set of uccode and lccode values to a whole paragraph. ε-TEX and all other TEX variants which implement the ε-TEX extensions have this fixed, however.
- Invisible glyphs are inserted into the output which can't be 'mapped away' for cut and paste support in PDF files (see next section).

Fortunately, LATEX provides a more elegant solution: `\MakeUppercase`, a wrapper macro around the `\uppercase` primitive (which is never called directly in the LATEX base files). This macro checks another macro called `\@uclclist` whether replacement macros for entities in its argument are available. By default, `\@uclclist` is quite short:

```
\def\@uclclist{%
  \oe\OE \o\O \ae\AE \dh\DH
  \dj\DJ \l\L \ng\NG \ss\SS \th\TH}
```

`\MakeUppercase` replaces entity `\oe` with `\OE`, `\ng` with `\NG`, and so on.
`lgrenc.def` extends this list with entries for Greek:

```
\expandafter\def\expandafter\@uclclist\expandafter{%
  \@uclclist
  \grka\grkA
  \grkb\grkB
  \grkd\grkD
  ...}
```

The main trick is to map the Greek accent commands to an empty macro or to other accent macros. With

```
\def\grk@none{}
```

we can now add the following pairs to `\@uclclist`:

```
\grkoxia\grk@none
\grkvaria\grk@none
...
\grkdialperis\"
\grkdialtonos\"
...
\grkpsilivariaiota\grksubiota
\grkdasiavariaiota\grksubiota
...
```

Contrary to ASCII input of Greek, some accents vanish completely during conversion to uppercase characters. People might argue that this causes a loss of information; this is true to a certain extent but unavoidable, because `LGR` neither contains all precomposed glyphs for accented uppercase characters, as they occur at the beginning of a paragraph or proper name, nor special representation forms without diacritics for text to be typeset in all-uppercase.

One problem, however, can't be solved. As discussed extensively in [10] and [11], correct uppercasing of words like ἄυλος (which should become ΑΫΛΟΣ – note the dieresis)[9] can't be handled with TEX; a more sophisticated solution is needed for that. See section 8 below for more.

6 Cut and paste support for pdfTEX

In 2005, Απόστολος Συρόπουλος (Apostolos SYROPOULOS) updated the Type 1 PostScript versions of the Greek CB fonts to contain glyph names which follow the rules of the *Adobe Glyph List* (AGL) [6]; you can find a PostScript encoding vector for the `LGR` encoding in file `CB.enc`. Theoretically, this should be sufficient to get correct mappings from glyph names back to Unicode so that cut and paste from a PDF file yields the right Unicode values. However, some Acroread versions to view PDFs appear to be buggy, ignoring the glyph names for an unknown reason.

To overcome this, I decided to write a file `lgr.cmap`, providing an explicit ToUnicode CMap (see section 5.9 in [7] for more details on the CMap format) which can be loaded by the `cmap` package written by Владимир Волович (Vladimir Volovich). For comparison purposes, here the first few data lines of `CB.enc`

```
/CBencoding [
  /endash      % EN DASH
  /uni0020032F % SPACE + COMBINING INVERTED BREVE BELOW
  /u10144      % GREEK ACROPHONIC ATTIC FIFTY
  /u10145      % GREEK ACROPHONIC ATTIC FIVE HUNDRED
  /u10146      % GREEK ACROPHONIC ATTIC FIVE THOUSAND
```

[9] Contrary to αὐλός which has ΑΥΛΟΣ as its uppercased form.

```
/u10147      % GREEK ACROPHONIC ATTIC FIFTY THOUSAND
/uni03DB     % GREEK SMALL LETTER STIGMA
/uni03DB.var % GREEK SMALL LETTER STIGMA (variant)
...
```

and `lgr.cmap`:

```
...
/CIDSystemInfo
<< /Registry (TeX)
   /Ordering (LGR)
   /Supplement 0
>> def
/CMapName /TeX-LGR-0 def
/CMapType 2 def
1 begincodespacerange
  <00> <FF>
endcodespacerange
64 beginbfchar
  <00> <2013>
  <01> <0020032F>
  <02> <D800DD44>
  <03> <D800DD45>
  <04> <D800DD46>
  <05> <D800DD47>
  <06> <03DB>
  <07> <03DB>
  ...
```

Some explanations.

- It is possible to map a glyph to more than a single character, as the entries for the second glyph demonstrate. TeX accents are always spacing glyphs, but most accents in Unicode are non-spacing, so we have to use the space character U+0020 as the base character and apply the non-spacing accent U+032F to it.
- A Unicode value larger than U+FFFF must be represented as a *surrogate pair* (which is a pair of two 16-bit integers) in a CMap since PDF uses the UTF16-BE encoding for such data. The conversion rules from Unicode scalar values to surrogate pairs (and vice versa) are given in [2].
- The glyph in slot 0x07 is a variant glyph of slot 0x06; both values get the same Unicode value.
- It is not possible to 'omit' a value in a CMap (this is, to not assign a Unicode value to a glyph index). If you do so, Acroread uses the particular

glyph index as the Unicode value instead of omitting the glyph, as some experiments have shown.

One big problem, unfortunately, can't be solved with CMaps: correct mappings for uppercase Greek letters with diacritics. As described earlier, `LGR` doesn't contain precomposed glyphs for them. Instead, the ligature mechanism represents them as two or three glyphs: the accent glyph (which consists of one or two diacritics) and the base glyph, possibly followed by a subscript iota, in that order – exactly the opposite of what Unicode needs.

6.1 The *nexus* problem

There are two glyphs in `LGR` which have a very special purpose: The entities at positions 0x10 and 0x11, together with a horizontal rule, form a kind of horizontal bracket, a nexus, to be used in philology. It looks like this, and it is intended as a substitute for an extensible, hat-like bracket (like $\widehat{\mathrm{rrr}}$). After some discussion with Claudio Beccari and Apostolos Syropoulos it has become clear that the Type 1 glyph names currently used[10] are misnomers. Instead, they represent the left and right part of U+23E0 TOP TORTOISE SHELL BRACKET – the middle part is the horizontal rule. However, a follow-up discussion on the Unicode mailing list [3] (in March 2008) showed that horizontal brackets can't be encoded directly with plain Unicode; a higher-level markup is needed for that. This means that it is not possible to provide a mapping from Unicode input to `LGR`; consequently, there are no entries for the two glyphs in `lgrenc.dfu`. For a future version of the Type 1 fonts we have agreed on the glyph names `uni23E0.left` and `uni23E0.right`, respectively; `lgr.cmap` uses those glyphs names too. In `lgrenc.def`, I have assigned the macros `\grknexusl` and `\grknexusr` for the sake of completeness.

7 The `LGI` encoding of the Ibycus fonts

Similar to `LGR`, Greek input for the Ibycus fonts [8] is realized with input ligatures. However, all diacritics are entered *after* the base character (see table 4 for a comparison of the used ligature characters).

Besides following the (lowercase) input conventions for the *Thesaurus Linguæ Græcæ* [12], the largest corpus of ancient Greek texts, its ligatures are directly suitable for Unicode combining sequences. The many dots in the encoding, which are input as exclamation marks and graphically put below the glyphs,[11] represent partially preserved characters in manuscript or epigraphical texts. However, this doesn't completely explain why there are so many dots.

[10] `uni02CF`, MODIFIER LETTER LOW ACUTE ACCENT and `uni02CE`, MODIFIER LETTER LOW GRAVE ACCENT.

[11] The large number of such dots is needed to center them vertically for various glyph widths.

Table 3: The `LGI` encoding of `fibr84`. Note the many zero-width glyphs.

diacritic	LGR	LGI
dasia	`<`	`(`
psili	`>`	`)`
tonos	`'`	`'`
varia	`` ` ``	`` ` ``
perispomeni	`~`	`=`
dialytika	`"`	`+`
iota subscript	`|`	`|`

Table 4: A comparison between the `LGR` and `LGI` input ligatures. For example, character 'ᾧ' is input as '`<~w|`' in `LGR` and as '`w(=|`' in `LGI`.

You probably might have noticed that there are no precomposed glyphs which contains subscript iota, and you might have wondered how proper kerning with the next glyph works if the zero-width subscript iota has to be inserted between two base glyphs – the same problem exists for the under-dot too. The solution is to use glyph clones which look exactly the same, but which are used to indicate the previous base character.

As an example, let us assume the composition of 'ᾔϊ'. Character 'ἤ' is input as '`h)'`'; file `fibr84.tfm` now contains the following ligature rule (using real glyphs instead of glyph slot values for better readability):

```
(LABEL ἤ)
(/LIG ͅ O 371)
```

In words: If 'ἤ' is followed by a subscript iota, retain 'ἤ' and insert the glyph at octal position 371,[12] which is a zero-width subscript iota glyph properly shifted to the left. Those two glyphs together appear as 'ᾔ' as expected.

We now have another ligature:

```
(LABEL O 371)
(KRN C i R 0.027779)
```

Again in words: If the glyph at octal position 371 is followed by (input) character 'ι', insert some kern. Now the next ligature building cycle starts: The base iota glyph forms a ligature together with the diaeresis, and so on, and so on.

The corollary of the above ligature mechanism is that we need just a single subscript iota glyph to transport the kerning information for a whole class of precomposed characters: In our particular example, the twelve glyphs 'ηὴἡἠἣ ήἥἤἢῆἧἦ' need a single subscript iota in the `LGI` encoding to form 'ῃῂᾑᾐᾓῄᾕ ᾔᾒῇᾗᾖ'; this saves eleven glyph slots if compared to `LGR`.

[12]This glyph is called 'iotasubeta' in the Type 1 version of the Ibycus fonts, which is unfortunately not a valid AGL glyph name. A proper replacement would be 'uni0345.eta', for example.

8 The xgreek package for XeTeX

Given that Apostolos Syropoulos has written most of Babel's Greek support it is not suprising that he is also the author of the xgreek package for XeTeX, which does approximately the same. There is only one noteworthy difference between xgreek and the Unicode support I present in this article, namely better support for the Greek specific `\MakeUppercase` within Babel, which is currently missing in xgreek.[13]

Ideally, such a case changing should be provided by OpenType features,[14] however, not all fonts usable by XeTeX actually have those tables. Even better, the uppercasing should be done using Unicode mapping tables (as defined by the Unicode Consortium), then applying OpenType features to convert the uppercase letter sequences (this is, with diacritics) to the correct presentation forms as used in Greek typography – there is basically no reason to add uppercasing support to fonts since this can be done on the input character level.

9 A forthcoming X7 encoding?

Today, `LGR` is the de-facto standard for typesetting (modern) Greek with plain TeX or LaTeX; together with the `LGI` encoding, which has its niche for philogical purposes, most needs can be covered.

Inofficially, the encoding name `X7` is reserved for a standard LaTeX 8bit Greek encoding. However, given that some typographical problems can't be solved with the plain TeX engine, as discussed above, it is probably a waste of time to invest energy in defining a new, 'standard' encoding. Anyway, here are some `X7` items which came to my mind while writing this article.

- Many glyphs contained in `LGR` do belong into a companion `XS7` encoding. Candidates are, for example, the acrophonic numerals or the glyphs forming the nexus bracket. A rough count gives at least 10 such characters which don't interact typographically with Greek letters.

- The available slots should be then filled with more useful glyphs, for example, precomposed accented uppercase characters for monotonic Greek.

- For better compatibility with Unicode, and for better ToUnicode support in PDF documents, only postfix notation should be used for ligatures (as is done in the `LGI` encoding).

[13] **Editor's note:** The uppercase–lowercase conversion scheme for Greek characters is already included in the `xetex` package. For the curious, that scheme can be found in the file `xgrcodes.tex`, which is part of the latest version of `xetex-greek`. That version can be found in CTAN, but was not included in the 2007 release of TeXLive.

[14] According to [9] it is best to use the 'calt' (contextual alternates) feature together with 'titl' (titling). The former removes the accents, and the latter does the uppercasing.

- Postfix notation also saves a great deal of glyph slots: All combinations with subscript iota don't need to be precomposed, saving 33 glyph slots (for base characters 'α', 'η', and 'ω').

References

[1] The ucs package. http://www.ctan.org/tex-archive/macros/latex/contrib/unicode.

[2] The Unicode Standard. http://unicode.org.

[3] The Unicode Public Email List. http://www.unicode.org/consortium/distlist.html.

[4] Alan Jeffrey, Frank Mittelbach, *inputenc.sty*. File inputenc.dtx of the base LaTeX bundle.

[5] Frank Mittelbach, Chris Rowley, *Providing some UTF-8 support via inputenc*. File utf8ienc.dtx of the base LaTeX bundle.

[6] *Unicode and Glyph Names*, version 2.4, Adobe Systems, 2003. http://www.adobe.com/devnet/opentype/archives/glyph.html.

[7] *PDF reference*: Adobe portable document format version 1.6. Adobe Systems, 3rd ed., 2005. http://partners.adobe.com/public/developer/en/pdf/PDFReference16.pdf.

[8] Pierre MacKay, Walter Schmidt, *The Ibycus font for Greek*. http://www.ctan.org/tex-archive/fonts/greek/ibygrk.

[9] *Greek All Caps*. A discussion how to implement Greek uppercasing with OpenType features. http://www.typophile.com/node/15982.

[10] Γιάννης Χαραλάμπους (Yannis Haralambous), *Τὸ Ω καὶ τὰ ἑλληνικά, ἢ «ὁ ἄυλος αὐλός»*, Eutypon, Issue 2 (April 1999), pp. 1–15. http://www.eutypon.gr/eutypon/pdf/e1999-02/e02-a01.pdf.

[11] Nick Nicholas, *Greek Unicode Issues: Titlecase and Adscripts*. http://ptolemy.tlg.uci.edu/~opoudjis/unicode/unicode_adscript.html.

[12] The *Thesaurus linguæ Græcæ*. http://www.tlg.uci.edu/.