

# Greek support for the ConT<sub>E</sub>Xt macro package

---

Thomas A. Schmitz

*Dept. of Classics  
Bonn University  
Am Hof 1 e  
53113 Bonn  
Germany  
E-mail: [thomas.schmitz@uni-bonn.de](mailto:thomas.schmitz@uni-bonn.de)*

This paper describes the implementation of support for typesetting ancient (polytonic) Greek in ConT<sub>E</sub>Xt. ConT<sub>E</sub>Xt is a macro package for T<sub>E</sub>X. It allows for a great deal of flexibility and customizability. Support for typesetting polytonic Greek was lacking. The article describes in detail what was needed to typeset Greek with ConT<sub>E</sub>Xt. It discusses the most frequently used input methods (Unicode and transliterated ASCII babel input), fonts and encodings and some of the problems that had to be solved. It also describes some of the challenges and new possibilities which luaT<sub>E</sub>X, the designated successor to pdfT<sub>E</sub>X, is bringing.

## 1 Introduction

Though it seems like a long time in retrospect, support for writing polytonic Greek in ConT<sub>E</sub>Xt has been around for somewhat more than two years only. When I started writing the package (or “module,” as they are called in ConT<sub>E</sub>Xt), it was initially only for my own needs, so I must apologize for beginning with a few sentences about myself, explaining what these needs are.

I am a classicist, teaching ancient Greek at a German university. For a long time, I had been using the same WYSIWYG word processor that everyone else seemed to be using. But after finishing a book project with it in early 2002, I became so dissatisfied that I was ready to try something new. It was a coincidence that brought me to T<sub>E</sub>X. A relatively easy and painless way of installing a complete T<sub>E</sub>X system on Mac OS X (which I use) became available then, and I just downloaded it and played around. I became immediately fascinated with the ease and customizability that ConT<sub>E</sub>Xt offers. But there was no support for (polytonic) Greek in ConT<sub>E</sub>Xt, so I went, somewhat reluctantly, to L<sup>A</sup>T<sub>E</sub>X, where I could use packages such as `teubner` or `psgreek` to type Greek. I was

too inexperienced to even think of writing Greek support for ConT<sub>E</sub>Xt myself, but I came back to it a number of times. By sheer perseverance, by nagging and pestering people on the relevant mailing lists, I managed to get a lot of help, and I began to understand what was required. Slowly, support for writing polytonic Greek developed, became usable, and even stable. ConT<sub>E</sub>Xt is developing at a really brisk pace, and its developpers are extremely helpful and friendly. My heartfelt thanks go to a number of people, without whom none of this would have been possible: Hans Hagen and Taco Hoekwater, who develop ConT<sub>E</sub>Xt, Giuseppe Bilotta, and Adam Lindsay.

Of course, it took me a rather long time to really get things the way I wanted them. The module now seems rather stable, and I have not had to change any important parts in it for quite a while. I now write almost everything I need in ConT<sub>E</sub>Xt: all my teaching materials, presentations, book projects, bibliographies, etc. Only when I have to share work with my less enlightened colleagues I (grudgingly) go back to using a word processor.

This article proposes to explain the basic architecture behind the module `ancientgreek`. Since I suspect that most readers are not familiar with ConT<sub>E</sub>Xt and its syntax and since I don’t want to proselytize<sup>1</sup>, there will not be much actual code in this article. Instead, I will try to explain the fundamental decisions and problems of the implementation. The module itself can be downloaded at <http://modules.contextgarden.net/t-greek>; the download contains full documentation. I provide more technical details about the way it is implemented in a short article “Greek in Proper ConT<sub>E</sub>Xt,” which can be found at <http://articles.contextgarden.net/article/67>.

## 2 The requirements

My first aim was to reproduce as exactly as possible what L<sup>A</sup>T<sub>E</sub>X packages such as `teubner` or `psgreek` offered. ASCII-input according to the `babel` transliteration scheme seemed a very good choice because such a transliteration scheme allows users to type Greek passages in any text editor. Since the main target audience for the module is scholars studying ancient Greek, users will generally have text in a Western language interspersed with (shorter or longer) Greek passages; so a simple interface for switching between the main language and Greek was needed. The L<sup>A</sup>T<sub>E</sub>X packages offer two commands for this: the command `\localgreek{...}` for shorter passages of Greek, and the environment `\begin{greek} ... \end{greek}` for extended passages (in order to conform to general ConT<sub>E</sub>Xt usage, the latter would be translated into an environment of the type `\startgreek ... \stopgreek`).

---

<sup>1</sup>Actually, I do. Try ConT<sub>E</sub>Xt. You’ll love it.

### 3 The implementation

The equivalent of a L<sup>A</sup>T<sub>E</sub>X package is called a “module” in ConT<sub>E</sub>Xt. Like a package, it is loaded in the preamble of the document. It is only fairly recently (spring 2006) that a syntax corresponding to L<sup>A</sup>T<sub>E</sub>X package options has been implemented in ConT<sub>E</sub>Xt. The module is now loaded with these parameters:

```
\usemodule[ancientgreek]
  [font=<name>,scale=<factor>,altfont=<name>,altscale=<factor>]
```

The user can specify a Greek font and an alternative font `altfont` (if (s)he wants to mix two Greek typefaces); both can have their own scaling factors (below, section 3.4).

I was at first reluctant to implement fonts as full font “families” with “italics,” bold, and bold italics since these have no real historical justification for the Greek script (typographically, Greek “italics” is a very recent invention), but since so many Greek fonts now come with these variants, I gave up my resistance. Within the Greek text, the usual font switches (in ConT<sub>E</sub>Xt, these are `{\em }` or `{\it }`, `{\bf }`, and `{\bi }`) work for fonts that offer these families.<sup>2</sup>

#### 3.1 Babel input

Implementing ASCII input the same way as `babel` handles it was relatively straightforward once you understand how it works. As many readers know, `babel` lets you enter breathings, accents, and the iota subscript via special ASCII symbols: “<” and “>” for the rough and smooth breathings; “’,” “‘,” and “~” for the acute, grave, and circumflex accents, and “|” for the iota subscript. A typical line of polytonic Greek thus looks like this:

```
M~hnhin >'aeide, je'a, Phlh|i'adew >Aqil~hoc
Μῆνην ἄειδε, θεά, Πηληϊάδεω Ἀχιλῆος
```

The correspondence between the “naked” consonants and vowels and their Greek counterparts is a matter of convention. In order to combine the breathings/accents and vowels into precomposed characters, `babel` (ab)uses T<sub>E</sub>X’s ligature mechanism, so this is implemented at the level of the font, specifically of the `tfm` file which contains the ligature information that combines, e.g., a ~ followed by a `h` into the character “eta with circumflex accent” ῆ.

This works quite well, but it needs some special considerations: some of the characters that this input method uses are active in T<sub>E</sub>X (catcode 13). If we want to use them as normal letters for our Greek input, we need to reverse this and make them inactive, assigning them to the group “other” (catcode 12). This has to be wrapped into the definitions for the macro `\localgreek` and

<sup>2</sup>As far as I can see, Greek support in L<sup>A</sup>T<sub>E</sub>X does not offer this possibility.

the environment `\startgreek` since we want this just for the Greek passages, not for the rest of the document.

Once this principle is clear, ASCII input is quite easy to define. All we have to do is change these catcodes and switch to a Greek font. The proper conversion of the ASCII characters to their Greek equivalents is produced by the font encoding; the production of the precomposed combinations by the ligature mechanism; so once we have set up our fonts (below, section 3.3), we’re almost home free.

### 3.2 Unicode input

Unicode input was more tricky, but since the framework is already there in ConTEXt, it was feasible (with some help). ConTEXt has native support to load Unicode vectors. These are defined in files with names such as `unic-XXX.tex`; for polytonic Greek, this is `unic-031.tex`. The beginning of this file looks like this:

```
\startunicodevector 31
  \expandafter\strippedcsname
  \ifcase\numexpr#1\relax
    \greekalphapsili \or %1f00
    \greekalphadasia \or
    \greekalphapsilivaria \or
    \greekalphadasiaivaria \or
    \greekalphapsilitonos \or
    \greekalphadasiatonos \or
    \greekalphapsiliperispomeni \or
```

I must admit that this wizardry is beyond my level of knowledge; if you’re curious, you’ll find the definition of the relevant macros and a very enlightening commentary in the file `unic-ini.tex`, which is part of the ConTEXt distribution. Here, it is enough to know what these macros do: they enable ConTEXt to look up Unicode characters from vector 31 (Greek Extended) in this table and transform them into “named characters,” which behave just like any ordinary TEX command. So the ConTEXt engine “knows” that the character with the hexadecimal value 1F00 which it finds in the input file has to be converted to the named character `\greekalphapsili` and so on.

So our input stream of Unicode characters is translated into a sequence of TEX commands. Now we need just one last piece of the jigsaw puzzle: ConTEXt now needs to be told to which characters in an actual font these names refer. This is done by files with names such as `enco-xxx.tex`; for our Greek module, it’s `enco-agr.tex`. Here are a few lines of this file:

```
\definecharacter greekalphadasia 161
\definecharacter greekalphapsili 162
\definecharacter greekalphaoxia 163
```

|   |     |
|---|-----|
| <code>\definecharacter greekalphadasiatonos</code>  | 164 |
| <code>\definecharacter greekalphapsilitonos</code>  | 165 |
| <code>\definecharacter greekalphavaria</code>       | 166 |
| <code>\definecharacter greekalphadasiavaria</code>  | 167 |
| <code>\definecharacter greekalphapsilivaria</code>  | 168 |
| <code>\definecharacter greekalphaperispomeni</code> | 169 |

We now understand what’s happening: in the input stream, ConT<sub>E</sub>Xt reads the Unicode character with the value 1F00 “Greek small letter alpha with psili” (ἄ̣). It looks this character up in the corresponding vector `unic-031.tex` and translates it to the name `\greekalphapsili`. The command `\localgreek` has switched to a Greek font, and all Greek fonts are connected (by their typescript file) to the encoding `enco-agr.tex`, so ConT<sub>E</sub>Xt knows it has to look for the name of this character in this file where it finds that `\greekalphapsili` is character 162 in the current font. So it now takes character 162 of the current Greek font, and this happens to be the lowercase alpha with a smooth breathing (*psili*), as you can verify in Figure 2.

### 3.3 The fonts

In order to understand how the Greek module handles fonts, we need a bit of general background about fonts in T<sub>E</sub>X. Let’s first have a look at the various types of font that are around:

1. Metafont produces bitmap fonts. This was the system originally developed by Don Knuth for T<sub>E</sub>X but such bitmap fonts look fuzzy on the screen, and they cannot be scaled, but have to be produced in every size you want. They seem a bit dated today.
2. PostScript type 1 are vector fonts that usually come as `pfb` files. They can be scaled at arbitrary values. Older fonts usually contain a maximum of 256 characters. This is the type of font that can directly be used by drivers such as `dvips`.
3. TrueType fonts (`ttf`) can only be used in pdfT<sub>E</sub>X, not if you use the good old T<sub>E</sub>X + `dvips` route. Some of them hold many thousand glyphs. Like PostScript fonts, this is a scalable vector format.
4. OpenType (`otf`) is a new format. It is a wrapper that can contain either PostScript or TrueType fonts. Since they haven’t been around very long, support for OpenType fonts in T<sub>E</sub>X is incomplete at the moment, but there is already a number of tools which can prepare OpenType fonts for work with T<sub>E</sub>X.

ConT<sub>E</sub>Xt produces pdf output by default, so it made sense to support all types of vector fonts which can be used in pdfT<sub>E</sub>X, but not Metafont.

Moreover, we need to understand the way in which  $\text{\TeX}$  uses a font. When  $\text{\TeX}$  breaks an input stream into paragraphs and pages, it only considers the metrical information of a font, stored in a so-called “ $\text{\TeX}$  font metric” or `tfm` for short. After reading the metrical information for the characters,  $\text{\TeX}$  then builds the page by typesetting “empty boxes,” connecting them with “glue,” and splitting this output into lines, paragraphs, and pages. Only at a later stage of the typesetting process does a driver such as `dvips` or the pdf driver in `pdf\TeX` fill the little boxes with the actual character shapes, the “glyphs.” This means that in order to use a font,  $\text{\TeX}$  needs a number of supporting files and information. Let’s have a look at the process:

- As mentioned above,  $\text{\TeX}$  needs a `tfm` file. This contains not just the dimensions of the “boxes” for the letters, but also information about kerning and ligatures. Every `tfm` has a maximum of 256 characters.
- $\text{\TeX}$  needs to be told to which actual font file (`pfb`, `ttf`, or `otf`) this `tfm` is connected. This information is contained in a map file (extension `map`). `Con\TeXt` can load such map files during runtime, but of course, it needs to be told which map to load; this is usually done in a typescript-file. The Greek module comes with such a file, `type-agr.tex`.
- This map file also tells  $\text{\TeX}$  which encoding (`.enc`) to use. This is where things become confusing: this encoding has nothing to do with `Con\TeXt`’s font encoding which we have mentioned in section 3.2. Such an encoding holds exactly 256 slots with names for glyphs, corresponding to the names in the actual font file. We can now formulate the process described in the preceding section with a bit more precision: the file `enco-agr.tex` tells `Con\TeXt` that the named glyph `greekalphapsili` corresponds to character 162. It then looks into the `enc` file with which the font is connected in the `map` file and sees that in this particular font, this character is named `alphaenis` (but it could equally well be a name such as `uni1F00` or even `blabla`). It then picks the character thus named from the actual font file and puts it in the box with the dimensions for character 162.
- Finally, you want users to be able to switch fonts easily, without having to remember the (often technical and awkward) names of the `tfm` files. In  $\text{\LaTeX}$ , this mechanism is provided with packages (`.sty`) and font definition files (`.fd`); in `Con\TeXt`, this is done via “typescripts.” A typescript provides the link between convenient symbolical names and the names of the actual `tfm` files; it groups fonts into families with roman, italics, bold, and bold italics variants, it links these fonts to a `Con\TeXt` encoding vector, and it allows relative scaling of these fonts. There is one big typescript `type-agr.tex` which contains all this information for all the Greek fonts.

Sounds complicated? Once you have understood the mechanism, it’s not too difficult. The thing you have to keep in mind is that we’re dealing with

two sets of names: within ConT<sub>E</sub>Xt, every Greek character has a name which is defined in `enco-agr.tex`. Of course, these names have to be consistent across all fonts. Within the fonts themselves, every glyph has a name which must be known to T<sub>E</sub>X so it can pick the right glyphs from the actual font files. These names vary from font to font. The most important task for us is to provide means for translating from ConT<sub>E</sub>Xt names to font names, and this means that preparing Greek fonts for use with T<sub>E</sub>X demands a lot of tedious work. Unfortunately, font designers cannot agree on standard names for their glyphs, especially not for glyphs outside of the standard range. It is extremely rare to meet two fonts with the same names for Greek glyphs; I have even seen the case that the roman, italic, and bold versions of the same font used different names. Moreover, font designers often don't know much about the Greek language, so they make mistakes and assign wrong names. This means that for basically every font I wanted to use, I had to write a corresponding `enc` file with the Greek glyphs in exactly the order that `enco-agr` defines.

Of course, I found it exciting that I could use so many Greek fonts with T<sub>E</sub>X once I had prepared all these supporting files. The L<sup>A</sup>T<sub>E</sub>X packages mentioned above offer a small (if excellent) selection of of fonts, but now I could have whatever was available! I must admit that I've gone a bit overboard: whenever I discovered a new Greek font somewhere, I just *had* to include support for it in the module. In particular, my module supports all the fonts from the Greek Font Society, Victor Gaultney's wonderful Gentium font, but also a number of commercial fonts (which are not included in the module, of course). At the time of writing, there are 35 fonts supported, quite a few of them in several variants. I surfed the web and looked for free fonts; I haven't discovered any new one in a while (so if anybody knows of any hidden gems, contact me!). Here is how I proceed to prepare a new font:

1. For TrueType and OpenType fonts, I extract the metrical and kerning information they contain into an `afm` file; this is easy with a tool such as `fontforge`.
2. Then, I write an `enc` file which contains the names of the glyphs I need. For some fonts, I provide several slightly differing encodings: some fonts, e.g., have a variant for the letter sigma, the so-called “lunate” or round sigma. With the help of different encoding files, I can either use this lunar form or the traditional one without having to change the input.
3. Another file contains the information about the necessary ligatures.
4. Now, the tool `afm2p1`, developed by Siep Kroonenberg, is used to extract the 256 glyphs contained in the encoding file and write a “property list” (`p1`) which contains the metrical information and the ligature commands that we need for building our accented characters. `p1` files contain exactly the same information as `tfm` files, but in a (more or less) readable form (while `tfms` are binary files); both formats can be translated into each other.

5. This `pl` is then converted into a `tfm` with the tool `pltotf`, developed by Don Knuth himself.
6. Last step: the font has to be registered in the map file and in the typescript that contains all Greek fonts.

In the beginning, I did most of this manually; in the meantime, I have written a small `perl` script which automates most of the tedious work: if I run it on an `afm` file, it will cheerfully provide an encoding vector and ligature file which can then be fed to `afm2pl`. But there will never be a fully automated solution because glyph names are just too idiosyncratic and because almost no font designer adheres 100% to the Unicode specifications, so some manual editing and tweaking is always needed. But I usually don't take more than an hour or so to prepare a new font for inclusion in the module.

It was right at the beginning of my work on the Greek module that I took a decision which later turned out to have some consequences of which I hadn't thought before: I wrote my own encoding and did not adopt the standard  $\text{\LaTeX}$  Greek encoding `LGR` as defined in `lgrenc.def`. This decision meant that my fonts would not be compatible with their  $\text{\LaTeX}$  counterparts since `LGR` is hardcoded into a number of packages and into the Greek hyphenation patterns that ship with `babel` (see below, section 5.3). To be quite honest: I knew almost nothing about encodings and was not aware of these consequences when I was developing the module. The question is: would I change it in retrospect? Probably not.

You can have a look at Figures 1 and 2 to see the difference between my own encoding `agr` and `lgr`. The latter makes room for a number of glyphs that don't strike me as obviously useful for my scholarly work: when I write ancient Greek, I don't need a Euro sign (24); the combination of uppercase vowel + iota subscript (9–11) doesn't exist in ancient Greek, and I don't need to use the archaic number symbols in positions 2–5.

On the other hand, there's a number of glyphs that I do need or want to retain: the combinations of epsilon and omicron with the circumflex accent (18–23 of `agr`), various brackets and braces (8–9, 11–2, 173–5), the “lunate” sigma (1, 13). To me, these characters seem much more useful than the ones included by `lgr`. So even if I could go back, I probably wouldn't want to adopt the `lgr` encoding.

### 3.4 Scaling

Scaling is something which the `psgreek` package implements, and once you've seen it at work, you can't live without it anymore. As readers may know, the design sizes of fonts vary considerably – one font at “10pt” is almost as big as another font at “12pt.” Since the aim of the Greek module was to provide an easy way of mixing Greek passages with “normal” text in a Latin script, this becomes immediately visible when you look at a page of output: compared with the main text, the Greek passages will often appear to be “too small” or

|      | '0 | '1 | '2 | '3 | '4 | '5 | '6 | '7 |     |
|------|----|----|----|----|----|----|----|----|-----|
| '00x | -  | ˆ  | ⊠  | ⊡  | ⊢  | ⊣  | ς  | ϛ  | "0x |
| '01x | ι  | Aι | Hι | Ωι | A  | Υ  | α  | ϐ  |     |
| '02x | ϛ  | ˘  | ϛ  | ϛ  | ϛ  | ϛ  | ⊠  | ⊡  | "1x |
| '03x | €  | ‰  | ə  | ̂  | ˙  | ˚  | ˘  | ˙  |     |
| '04x | ˆ  | !  | ˆ  | ˆ  | ˆ  | %  | ˙  | ˙  | "2x |
| '05x | (  | )  | *  | +  | ,  | -  | .  | /  |     |
| '06x | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | "3x |
| '07x | 8  | 9  | :  | .  | '  | =  | '  | ;  |     |
| '10x | ˆ  | A  | B  | ˆ  | Δ  | E  | Φ  | Γ  | "4x |
| '11x | H  | I  | Θ  | K  | Λ  | M  | N  | O  |     |
| '12x | Π  | X  | P  | Σ  | T  | Y  | °  | Ω  | "5x |
| '13x | Ξ  | Ψ  | Z  | [  | ˆ  | ]  | ˆ  | ˆ  |     |
| '14x | ˙  | α  | β  | ς  | δ  | ε  | φ  | γ  | "6x |
| '15x | η  | ι  | θ  | κ  | λ  | μ  | ν  | ο  |     |
| '16x | π  | χ  | ρ  | σ  | τ  | υ  |    | ω  | "7x |
| '17x | ξ  | ψ  | ζ  | «  | .  | »  | ˘  | —  |     |
| '20x | á  | â  | ã  | ä  | å  | ä  | ä  | ä  | "8x |
| '21x | á  | ã  | ã  | ã  | ä  | ä  | ä  | ä  |     |
| '22x | ã  | ã  | ã  | F  | ä  | ä  | ä  | ˘  | "9x |
| '23x | ḥ  | ḥ  | ḥ  |    | ḥ  | ḥ  | ḥ  |    |     |
| '24x | ḥ  | ḥ  | ḥ  | ḥ  | ḥ  | ḥ  | ḥ  | ḥ  | "Ax |
| '25x | ḥ  | ḥ  | ḥ  | ḥ  | ḥ  | ḥ  | ḥ  | ḥ  |     |
| '26x | ṁ  | ṁ  | ṁ  | ṁ  | ṁ  | ṁ  | ṁ  | ṁ  | "Bx |
| '27x | ṁ  | ṁ  | ṁ  | ṁ  | ṁ  | ṁ  | ṁ  | ṁ  |     |
| '30x | ṁ  | ṁ  | ṁ  | F  | ṁ  | ṁ  | ṁ  | ṁ  | "Cx |
| '31x | ı  | ı  | ı  | ı  | ı  | ı  | ı  | ı  |     |
| '32x | ı  | ı  | ı  | ı  | ı  | ı  | ı  | ı  | "Dx |
| '33x | ı  | ı  | ı  | I  | ı  | ı  | ı  | ı  |     |
| '34x | é  | é  | é  | É  | ó  | ó  | ó  | ó  | "Ex |
| '35x | é  | É  | É  | É  | ó  | ó  | ó  | ó  |     |
| '36x | ı  | ı  | ı  | ı  | ı  | ı  | ı  | ı  | "Fx |
| '37x | φ  | η  | φ  | ϛ  | ϛ  |    |    | ˙  |     |
|      | "8 | "9 | "A | "B | "C | "D | "E | "F |     |

Figure 1: The lgr-encoding

|      | '0 | '1 | '2 | '3 | '4 | '5 | '6 | '7 |     |
|------|----|----|----|----|----|----|----|----|-----|
| '00x | –  | c  | –  | –  | ,  | –  | –  | –  | "0x |
| '01x | {  | }  | –  | [  | ]  | C  | I  | Y  |     |
| '02x | –  | –  | ε  | ō  | ē  | ō  | ē  | ō  | "1x |
| '03x | ī  | ī  | π  | –  | –  | –  | –  | –  |     |
| '04x | –  | .  | ᶓ  | F  | ς  | ρ  | ᶓ  | ,  | "2x |
| '05x | (  | )  | *  | †  | ,  | –  | .  | /  |     |
| '06x | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | "3x |
| '07x | 8  | 9  | ?  | .  | ˙  | =  | ˘  | ;  |     |
| '10x | .. | A  | B  | φ  | Δ  | E  | Φ  | Γ  | "4x |
| '11x | H  | I  | Θ  | K  | Λ  | M  | N  | O  |     |
| '12x | Π  | X  | P  | Σ  | T  | Y  | η  | Ω  | "5x |
| '13x | Ξ  | Ψ  | Z  | [  | .  | ]  | ~  | –  |     |
| '14x | ˙  | a  | β  | ς  | δ  | ε  | φ  | γ  | "6x |
| '15x | η  | ι  | θ  | κ  | λ  | μ  | ν  | ο  |     |
| '16x | π  | χ  | ρ  | σ  | τ  | υ  | φ  | ω  | "7x |
| '17x | ξ  | ψ  | ζ  | ⊥  |    | ⊥  | ~  | –  |     |
| '20x | “  | ”  | !  | í  | í  | í  | í  | í  | "8x |
| '21x | ì  | ì  | ì  | ì  | ì  | ì  | ì  | ì  |     |
| '22x | î  | î  | î  | î  | î  | î  | î  | î  | "9x |
| '23x | ē  | ē  | ē  | ē  | ē  | ē  | ē  | ē  |     |
| '24x | ˆ  | á  | á  | á  | á  | á  | á  | á  | "Ax |
| '25x | ã  | ã  | ã  | ã  | ã  | ã  | ã  | ã  |     |
| '26x | ä  | ä  | ä  | ä  | ä  | ä  | ä  | ä  | "Bx |
| '27x | é  | é  | é  | é  | é  | é  | é  | é  |     |
| '30x | ñ  | ñ  | ñ  | ñ  | /  | ñ  | ñ  | ñ  | "Cx |
| '31x | ñ  | ñ  | ñ  | ñ  | ñ  | ñ  | ñ  | ñ  |     |
| '32x | ó  | ó  | ó  | ó  | ó  | ó  | ó  | ó  | "Dx |
| '33x | ú  | ú  | ú  | ú  | ú  | ú  | ú  | ú  |     |
| '34x | û  | û  | û  | û  | û  | û  | ô  | ô  | "Ex |
| '35x | ó  | ó  | ó  | ó  | ó  | ó  | ó  | ó  |     |
| '36x | õ  | õ  | õ  | õ  | õ  | õ  | õ  | õ  | "Fx |
| '37x | õ  | õ  | õ  | õ  | õ  | õ  | õ  | õ  |     |
|      | "8 | "9 | "A | "B | "C | "D | "E | "F |     |

Figure 2: The agr-encoding

“too big.” Hence, the Greek font needs to be scaled just a bit. ConTEXt offers an easy way to implement scaling at the level of the typescript; this is achieved via the `rscale` (= relative scale) factor. Typically, adding just a very small amount of scaling (roughly between 95% and 105%) will make the proportions between Latin and Greek fonts about right.

### 3.5 Hyphenation

I must admit that proper Greek hyphenation was one of the last parts that was implemented, and it turned out to be more difficult than I had imagined. First, ancient Greek had to be added to the set of languages that ConTEXt supports; this was done in the file `lang-ctx.tex` with this line:

```
\installlanguage[\s!agr][\s!mapping=\s!agr,\s!encoding=\s!agr]
```

This means that ConTEXt will include patterns for the Greek language when it generates its formats.

Most importantly, of course, these patterns themselves had to be provided, and this is where things became difficult. Hans Hagen, the developer of ConTEXt, has decided to ship his own hyphenation patterns with his distribution because he had made some bad experiences with LATEX patterns being renamed, moved, or changed without advance warning, so he wanted to be independent. I was very grateful that I could use Dimitrios Filippou’s improved hyphenation-patterns for ancient Greek, `GRAhyph4.tex`. However, these patterns are provided in ASCII format and will only work with ASCII input, not with Unicode input. Hence, these patterns had to be translated into a format that would work both for ASCII and Unicode input. I must admit that I don’t know enough about hyphenation in TEX to do this conversion myself, but I was lucky: the same demand came up for XELATEX, a new project which works internally with Unicode; they also needed Greek hyphenation patterns which were independent from ASCII and `babel`, and Peter Heslin was very helpful in providing a translation of the old patterns into Unicode. ConTEXt has now adopted these new patterns, converted them into an interior format it uses, and includes them in its distribution under the name `lang-agr.pat`.

I must admit that some things still do not quite work as they should (with ASCII input, some unwanted hyphenation points are added) and that I hold somewhat different opinions about some hyphenations: the patterns we have now hyphenate, e.g.,  $\pi\alpha\rho\alpha\text{-}\gamma\alpha\text{-}\gamma\epsilon\acute{\iota}\nu$ , which I consider wrong on etymological grounds (it should be  $\pi\alpha\rho\text{-}\alpha\gamma\alpha\text{-}\gamma\epsilon\acute{\iota}\nu$ ), but it would require much more time and energy than I can invest these days to produce new and consistent patterns. We get correct results in 95% of the cases (which is already much more than the old default patterns `grhyph.tex` offered), and I can live with the rest, for the moment. But I’m aware that I will have to reopen this box at some point...

## 4 Limitations

For the time being, there are limitations (you could also call them “bugs” if you want, but I don’t see an obvious solution for them) in two areas; one has to do with the way the fonts work and concerns typographic quality; the other has to do with the catcode changes and concerns special environments.

- Most Greek fonts do not have many kerning pairs, but a few of the professional ones do. If this kerning implies vowels, it is obvious that the accented vowel should be kerned exactly as the unaccented one. This is where things become a bit complicated: as we have seen, `babel` input constructs these accented vowels via  $\TeX$ ’s ligature mechanism, so in your source, you have something like `a>’u`. Now if “a” and “u” form a kerning pair,  $\TeX$  would fail to apply kerning here since the two characters are not adjacent. It won’t help to add the composite character `greekalphapsilitonos` to the font’s kerning table: by the time this character is built by the ligature mechanism, the kerning is finished. It does help if the input file contains the character not in `babel` notation, but as a named glyph or as a Unicode character (since `Con $\TeX$ t` translates one into the other, this amounts to the same thing); in this case, the kerning will work. So would it make sense to recommend Unicode input for kerning? Unfortunately, things are even more complex. As we have seen,  $\TeX$ ’s metric files don’t have enough space for all Unicode characters of the range “Greek Extended.” That’s why I had to split up, on the level of the `Con $\TeX$ t` encoding vector, the precombined combination of “breathing/accent + uppercase vowel” into two characters so that the breathing (+ accent) is followed by the letter itself. However, in that case, we again need some kerning since if we just let these characters follow each other, there would be a noticeable gap. I had to add these kerning pairs manually to many fonts. In this case, however, we face the opposite problem: this kerning works beautifully with `babel` input. With Unicode input,  $\TeX$  decomposes the entity into two separate characters, but when this happens, the kerning is already finished. So you can’t get the best of both worlds: with `babel`, you lose kerning with accented lowercase letters; with Unicode, you lose kerning with uppercase letters and their breathings. It’s a minor, purely esthetical problem, and many fonts have few or no kerning pairs, but I find it annoying. I hope the new version of `pdf $\TeX$`  (below, section 5.1) will make it possible to find a solution for this problem when the limit of 256 characters per `tfm` falls; in this case, we can just include the uppercase characters with their breathings in the encoding vector and use Unicode.
- Another limitation has to do with the catcode changes I had to introduce for `babel` input (section 3.1): some  $\TeX$  environments rely on special active characters and will break if these characters do not work they way they are expected. Most noticeably, this is true for all `Con $\TeX$ t` ta-

ble environments, which use the textbar character | to delimit columns. Since our Greek module assigns a different catcode to this character, it is impossible to wrap an entire table or tabulate environment into a `\startgreek ... stopgreek` environment. The workaround for this limitation is simple, if a bit inconvenient: you have to wrap every single Greek passage into a `\localgreek{...}` command.

## 5 The future

Predictions are always uncertain, especially when they concern the future. Nevertheless, I will outline at least a few directions that this module could or will have to take. Some of them are new features that I would like to have; others are new developments in the underlying TeX structure that the module will have to take into account.

### 5.1 pdfTeX and luaTeX

In August 2007, a first public beta of luaTeX was released. This project will one day be the successor to pdfTeX. For the time being, it is still developing rapidly and neither the user interface nor the underlying engine can be considered stable, but we have an experimental release of ConTeXt (called `mkiv`) which makes use of advanced features of luaTeX. It allows to embed the scripting engine lua into TeX code and will bring important changes to the way we use TeX. For our Greek support, the most important development of luaTeX is that it can make native use of OpenType and TrueType fonts: encoding files and `tfm` files are not needed any more; luaTeX directly converts the metrical information it finds in the font file into an internal format and works with these files. For typesetting ancient Greek, this has two momentous consequences:

- luaTeX has native support for Unicode input; conversion to “named glyphs” via Unicode vectors is no longer needed. In this, luaTeX resembles XeTeX, which also works with Unicode. This is excellent news; in my first tests, Unicode input worked immediately with luaTeX. However, this also means that the old tricks for mapping ASCII babel input to Greek characters have to be redesigned. When more and more users and operating systems switch to full Unicode support, I expect babel input will become obsolete, but it will be a legacy that needs to be supported for quite some time. At the moment, I have been working with the luaTeX developers to find a solution. We have played (more or less successfully) with different approaches (map the entire input stream or just replace characters on the level of the font used), but they all have some difficulties, and we haven’t decided which road to take yet.
- With the native support for TrueType and OpenType fonts, the old limit of 256 characters for a `tfm` is no longer relevant; luaTeX can address every

character in a font. This means that we can use all the Greek characters; the tricks regarding uppercase Greek letters and their combinations with breathings and accents are no longer needed. However, this means that it will be more difficult to support Type1 fonts. I am working to find a solution, but this is work in progress so far.

Adapting the Greek module to this new implementation of T<sub>E</sub>X will be a challenge in the next years, but overall, I am certain that it will facilitate things tremendously.

## 5.2 Critical editions

In L<sup>A</sup>T<sub>E</sub>X, there is the `ledmac` package which is a port of the (plain T<sub>E</sub>X) `edmac` macros. It provides support for critical editions with several sets of footnotes, references to line numbers in the main text and other things which are needed for critical editions. ConT<sub>E</sub>Xt already has support for some of these features, but it is not yet complete and not very well documented. I began looking into it a couple of times, but I guess I will only be able to spend the time and energy into really accomplishing something when there is real need for it—i. e., when I really want to do a critical edition. However, this is something I would really love to see in ConT<sub>E</sub>Xt.

## 5.3 A L<sup>A</sup>T<sub>E</sub>X version?

Since I use L<sup>A</sup>T<sub>E</sub>X from time to time, I have of course thought about L<sup>A</sup>T<sub>E</sub>X support for my module. I have a prototype on my own computer which kind of works, but it has some serious limitations. Unfortunately, my L<sup>A</sup>T<sub>E</sub>X skills are not sufficient to overcome these defects. When I asked on newsgroups, L<sup>A</sup>T<sub>E</sub>X users did not seem overly interested in the package, but maybe one of the readers here wants to help? Here’s a short summary of things that do and don’t work:

- Font switching and scaling works just as in ConT<sub>E</sub>Xt; the only thing necessary for this was converting the typescripts into L<sup>A</sup>T<sub>E</sub>X font descriptions (`.fd`) and writing a package `tasgreek.sty` with just a few lines of code.
- Unicode input does not work. It would have to rely on one of the packages `ucs` or `inputenc` with the option `[utf8x]`. I am not sure about the state of support for these packages, and both rely on the `lgr` encoding for converting unicode input, so they don’t work with my own encoding.
- Hyphenation does not work. Again, the babel option `polutonikogreek` relies on `lgr` encoding for the hyphenation patterns. This could (and probably should) be changed to named glyphs from a proper encoding vector, but of course I don’t know enough about the implications and consequences.

So if anyone is interested in helping with support for  $\LaTeX$ , just drop me a line!

## 6 Conclusion

When you use  $\TeX$ , it is easy to forget how complex and involved the underlying structure really is. When you develop your own packages and/or modules, you inevitably encounter problems which will force you to explore the depths of your system. I have learned a lot about  $\TeX$  and friends, about macro writing, about fonts, and about scripting when I prepared this module, and I'm still learning more. I still am pleased as punch when I see my typeset documents with all the beautiful Greek passages and love how easy it is to achieve these results. And my endeavors have taught me a much deeper appreciation of the work of all these maintainers, developers, and package writers who put in a lot of effort to make our lives easier. Polytonic Greek will always be of interest to a very small minority of  $\TeX$  users, but I am proud that my effort makes me part of this wonderful community.