## LaTeX

# Never again active characters! Ω-Babel

Yannis Haralambous, John Plaice and Johannes Braams

*»Weißt Du, wo ich das Wasser des Lebens finden kann?«*
*»An der Grenze Phantásiens«, sagte Dame Aiuóla.*
*»Aber Phantásien hat keine Grenzen«, antwortete er.*
*»Doch, aber sie liegen nicht außen, sondern innen.«*

M. Ende, Die unendliche Geschichte

## Abstract

This progress report of the Ω development team (the first two authors) presents the first major application of Ω: an adaptation of Babel, the well-known multilingual LaTeX package, developed by the third author. We discuss problems related to multilingual typesetting, and show their solutions in the Ω-Babel system.

The paper is roughly divided into two parts: the first one (sections 1–4) is intended for average LaTeX users, especially those typesetting in languages other than American English; the second part (section 5) is more technical and will be of more interest to developers of multilingual LaTeX software.

## 1 Introduction

Ω-Babel is the first real-world application of Ω: we are in the process of adapting the multilingual LaTeX package Babel by Johannes Braams (1991a, 1991b) to take advantage of the functionality of Ω. This allows safer and more complete LaTeX typesetting of languages other than American English. Problems due to technical limitations of TeX are solved; for example, the LaTeX macros `\MakeUppercase` and `\MakeLowercase` have been replaced by Ω filtering processes. The whole process is simpler and more natural.

In this progress report we present the first step in adapting Babel to Ω. There will be (at least) two more steps which we describe below; for more information, see section 5 (the technicalities).

1. Babel adapted to Ω; we use DC font output. `\MakeUppercase` and `\MakeLowercase` macros are being replaced by macros launching translation processes. Various input encoding translation processes are being written. The `inputenc` and `fontenc` packages are being adapted (their Ω counterparts are called `inpenc` and `fntenc`). This step has been completed.
2. UC (Unicode Computer Modern) fonts will be released by spring 1996. These fonts contain Latin, Greek and Cyrillic characters, and a certain number of dingbats and graphical characters. Ω-Babel uses UC fonts for output: new languages will be dealt with (Bulgarian, Esperanto, Greek, Latvian, Lithuanian, Maltese, Russian, Vietnamese, Welsh, etc.). Alan Jeffrey's `fontinst` will be adapted to make extended virtual fonts in UC encoding.

   Latin letters with dieresis will be provided in two versions: with high or low accent: Frenchmen like dieresis (→ tréma) to be high, Germans prefer to have it (→ Umlaut) a little lower.
3. Arabic alphabet languages, Hebrew and Yiddish will be added at some point during 1996.
4. Soft hyphenation will be done through Ω translation processes. This allows dynamic loading of hyphenation algorithms, independently of the process of format creation. There will be no need to recompile formats when adding or changing hyphenation patterns. It will also be possible to add new features to hyphenation (automatic processing of German „ck→k-k“, preferential hyphenations, etc.).

## 2 Practically, what does this mean for me?

It means that if you are typesetting in some non-American English language covered by current Babel (Bahasa, Breton, Catalan, Croatian, Czech, Danish, Dutch, English, Estonian, Finnish, French, Galician, German, Hungarian, Irish, Italian, Lower or upper Sorbian, Norwegian, Polish, Romanian, Scottish, Slovakian, Slovenian, Spanish, Swedish, Portuguese or Turkish), then Ω can make it easier for you and give you better results. All you have to do is download the Ω implementation for your machine from `ftp://ftp.ens.fr/pub/tex/yannis/omega/systems` or the top-level `omega` directory[1]; if you use a TeX implementation not already covered, then ⁂kindly⁂ suggest to the implementor that they consider including Ω support). Then download the Ω-Babel package from `ftp://ftp.ens.fr/pub/tex/yannis/omega/macros/obabel` or , install it on your machine and use it! The basic syntax is the same as in Babel, but we will discuss new options and functionality in section 4.1.

**Figure 1**: Allegory: input (on the left) and output (on the right) encodings, too close together.

## 3 General philosophy of combined Ω and Babel

In Fig. 1, the reader can see an allegory of how TeX works: input (the worker on the left) and output

[1] At press time Ω has been ported to several UNIX machines (Sun, Silicon Graphics and others), DOS (by Kraus Kalle) and Macintosh (by Tom Kiffe).

(the one on the right) are close together. For example, consider the fact that hyphenation patterns—which are a language-intrinsic feature—are described in the output font encoding. In Fig. 2 you can see how Ω remedies this situation: input and output are clearly separated and, in between, there is a big container (imagine a huge barrel), the Unicode encoding.

Whatever you type is first of all converted into Unicode. This conversion is language dependent; for example, the ASCII character 'i' does not mean the same thing in Turkish and in the other Latin-alphabet languages. Unicode is very big; so you will hardly ever ask for something not available; even if that happens there is a "private zone" where we can temporarily store characters of our own choice.

Once inside Unicode we deal with pure device-independent information: we can process it in many different ways. In Fig. 2 we list five possible transforms, all pertaining to Ω-Babel, and we will discuss these in turn. The Ω translation processes needed for every language are activated when you enter the corresponding Ω-Babel environment. Our goal is to keep the technical aspects hidden: the average user typesetting in a given language does not need to be aware of the different transformations we have just described.

### Aliasing

We call "aliasing" the making of aliases. An *alias* is the expression of a character in some convenient way: for example in 7-bit ASCII. In German Babel, when you write `"s` instead of `\ss{}`, this is an alias: (a) it is 7-bit, (b) you can very well avoid it if your keyboard and screen support 8-bit characters, and (c) in TEX, it used to be handled using active characters.

### Inherent transforms

These go deeper than aliases; they are:

1. either transforms that traditionally belong to TEX syntax, like `---` for "—", or ` `` ` for the English opening double quote, or `?‘` for the Spanish inverted question mark "¿";
2. or transforms that historically derive from dactylographical keyboard traditions (for example, the French `<<` that gets converted into "«" with the appropriate spacing, or the Catalan `l.l` which produces "l·l");
3. or things that are supposed to be hidden from the user: for example, the Dutch `ij` which always produces the "ij" ligature, unless the user places something invisible in between: `bewijs` vs. `bi{}jektie`; or, in Turkish and Portuguese, the fact that there should be no 'ff', 'fi', ... ligatures, etc.

See 4.4 for a good example of the difference and combined use of aliases and inherent transforms.

### Hyphenation

Hyphenation must be done on the level where the information is most device-independent: that is, the Unicode level. Not only can you define hyphenation algorithms using all possible 16-bit characters (for example to hyphenate Welsh, which uses letters such as "ŵ"), but your algorithm is automatically valid for any input or output encoding. We'll come back to this issue later in 1996, when we reach step 3 of Ω-Babel development.

### Upper/lowercasing

This is certainly not a trivial process: different letters may share the same glyph for their upper or lower forms (example: both the Icelandic eth "ð" and the Croatian dz "đ" share the glyph "Đ" for their upper form); letters may have different upper forms depending on the semantics of the word (example: one possible upper form of the German "ß" is "SS", another one is "SZ"), or on local traditions (in bad French typography, upper forms of accented letters are not always accented), or on the language itself (example: the upper form of the Turkish letter "i" is "İ" and not "I"—the lower form of "I" is "ı" and not "i") these are major incompatibilities, and hence we should be able to dynamically change the upper/lowercasing process.

Finally, as Martin Dürst pointed out on the `omega` list[2], there are six kinds of letter cases:

1. regular lowercase (glyph becomes uppercase when we apply the uppercasing process);
2. regular uppercase (glyph becomes lowercase when we apply the lowercasing process);
3. fixed lowercase (glyph is invariant under the uppercasing process), for example the "m" and "b" in the German „GmbH";
4. fixed uppercase (glyph is invariant under the lowercasing process), for example the first letter of a name;
5. fake lowercase (the glyph is uppercase, it becomes lowercase when we apply the uppercasing process), for example the "i" in the modern German word "STUDENTiNNEN" (=politically correct male and female students);

[2] Join us on the Ω e-mail discussion forum `omega@ens.fr`, by sending the usual subscription message to `listserv@ens.fr`.

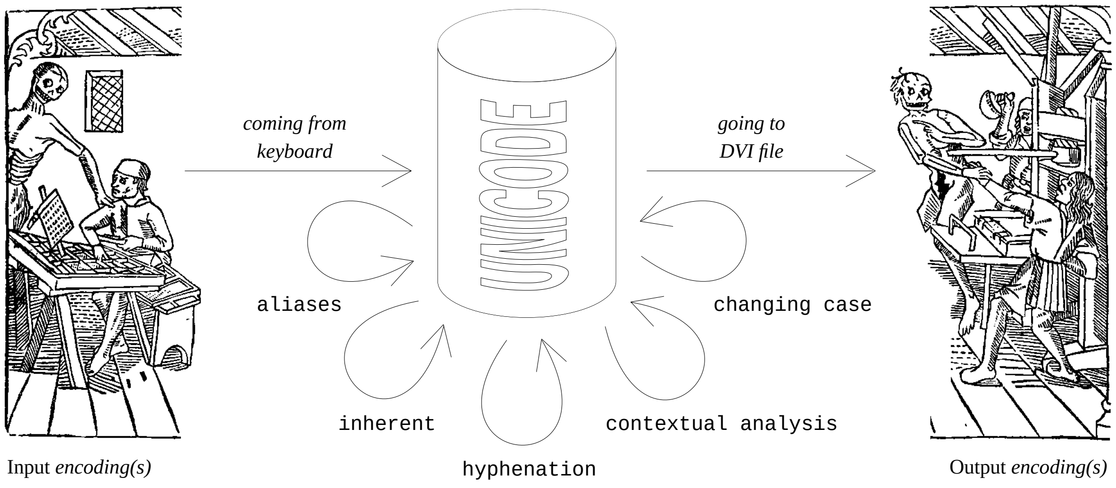

**Figure 2**: The Ω way: inserting Unicode between input and output encodings.

6. fake uppercase (the glyph is lowercase, it becomes uppercase when we apply the lowercasing process), for example the "I" in the the same modern German word "StudentInnen", written in lowercase type.

Ω introduces two new commands to handle "fixed" case (`\fixedcase`) and "fake" case (`\fakecase`). Using these commands, you can write

```
\fixedcase{T}ruth
```

to be sure that your word will always be "Truth, with a big T" or

```
\fixedcase{S}tudent\fakecase{I}nnen
```

to obtain correctly spelled politically correct (fe)male students in German. The argument of `\fixedcase` or `\fakecase` must be a single letter.

In Table 1 we present some examples of the use of the different case-related commands. Note in the first example, that—contrary to the usual TEX uppercasing commands—math mode is not affected.

### Contextual analysis

This transform is perhaps less important for Latin-alphabet languages (although it becomes important when you want to use a "long s" at the beginning of a word). It is extremely important for Greek (final or medial sigma), Hebrew (there are five letters with medial and final form), and especially for the Arabic alphabet.

## 4 Details on each language

Before we start considering languages one by one, a brief description of Ω-Babel syntax.

### 4.1 Ω-Babel syntax

To be able to use Ω-Babel you have to load the `omega` package: this is done automatically if you use either `inpenc` (the input encoding package) or `fntenc` (the output font encoding package). The former can take as an option an input encoding (the reader can find a list of such options on Table 3), the default value is `lat1` (ISO Latin-1). The latter can take only one option for the moment: `T1`. As of step 2 of Ω-Babel development a new encoding will be added: `UT1`. Here is an example of the loading of these packages:

```
\usepackage[stmac]{inpenc}
\usepackage[T1]{fntenc}
```

As in the original Babel package, you load Ω-Babel by using the command `\usepackage`, and by giving the names of languages you are going to use as an optional argument. These names are separated by commas, and the last one becomes the "default" language, as in

```
\usepackage[turkish,german,francais]{obabel}
```

The names for languages are the same as in original Babel.

Once Ω-Babel is loaded, you are in the "default language": captions, date, hyphenation, 7-bit input and typographical specifications are adapted.

Contrary to the original Babel where the command `\selectlanguage` was used to switch between languages, in Ω-Babel you have to use an environment, called `lang`. Here is an example:

```
\documentclass{article}
\usepackage[stmac]{inpenc}
\usepackage[T1]{fntenc}
\usepackage[germanb,turkish,%
```

| | |
|---|---|
| `\MakeUppercase{is it an $a$ or a $b$?}` → IS IT AN *a* OR A *b* | |
| `\MakeUppercase{mon \oe il}` → MON ŒIL | |
| (in German) `\MakeUppercase{ma"se oder ma"ze?}` → MASSE ODER MASZE? | |
| `\MakeLowercase{\MakeUppercase{ma"ze}}` → maße | |
| (in Turkish) `\MakeUppercase{kap\i y\i{} i\c ceri}` → KAPIYI İÇERİ | |
| `\MakeLowercase{BROWN}` → brown | |
| `\MakeLowercase{\fixedcase{B}ROWN}` → Brown | |
| `\MakeLowercase{\fixedcase{S}TUDENT\fakecase{i}NNEN}` → StudentInnen | |

Table 1: Some examples of upper/lowercasing.

```
francais]obabel}
\begin{document}

Histoires d'oiseaux en deux langues :

\begin{lang}{german}
"Uber allen Gipfeln ist Ruh,
die V"oglein schweigen im Walde...
\end{lang}

\begin{lang}{turkish}
Kedinin yakalad\i\u g\i{} ku\c sun
t\"uyleri havada u\c cu\c suyordu...
\end{lang}

\end{document}
```

This is because Ω uses a stack for translation processes: every time you enter a `lang` environment, Ω pushes a set of translation processes on the stack (see also section 5); every time you leave it, Ω pops a set of translation processes from the stack.

The environment approach is also essential for typesetting reasons: in right-to-left languages (Arabic, Hebrew, etc.), line breaking is done differently depending on whether we are in "global" or in "encapsulated" right-to-left mode.

As in the original Babel, the `\languagename` macro contains the name of the language.

Finally you have the following additional Ω commands: `\fixedcase` and `\fakecase` as described above, and a command `\omegaversion` which returns the version of your Ω implementation.

Let us now take one by one the languages covered by Ω-Babel to see what has changed.

## 4.2 French

French features are loaded by the `francais` option. Besides the usual hyphenation, caption and date changes, Ω-Babel transforms the punctuation you type, whether you include blank spaces or not.

For example, whether we write `<< Ciel, mon mari !  >>` or `<<Ciel, mon mari!>>` (or even ``` ``Ciel, mon mari!'' ``` if the `clever` option is on), the result will be the same: « Ciel, mon mari ! ». Not a single character is active, so there can be no possible interference with other TeX or LaTeX macros.

When we pass to step 2 of Ω-Babel development, `francais` will also act on the "Unicode → output font" level and switch to low dieresis letters[3].

## 4.3 German

The German features are loaded by the `germanb` option. All original Babel aliases have been kept:

| | |
|---|---|
| `"a` | `\"a`, also implemented for the other lowercase and uppercase vowels. |
| `"s` | to produce the German ß (like `\ss{}`). |
| `"z` | to produce the German ß (like `\ss{}`). |
| `"ck` | for `ck` to be hyphenated as `k-k`. |
| `"ff` | for `ff` to be hyphenated as `ff-f`, this is also implemented for l, m, n, p, r and t |
| `"S` | for `SS` to be `\MakeUppercase{"s}`. |
| `"Z` | for `SZ` to be `\MakeUppercase{"z}`. |
| `"\|` | disable ligature at this position. |
| `"-` | an explicit hyphen sign, allowing hyphenation in the rest of the word. |
| `""` | like `"-`, but producing no hyphen sign (for compound words with hyphen, e.g. `x-""y`). |
| `"~` | for a compound word mark without a breakpoint. |
| `"=` | for a compound word mark with a breakpoint, allowing hyphenation in the composing words. |
| `"‘` | for German left double quotes (looks like „). |
| `"’` | for German right double quotes. |
| `"<` | for French left double quotes (similar to <<). |
| `">` | for French right double quotes (similar to >>). |

[3] Default letters with dieresis will carry a "low" accent, French "higher" accented letters will be the exception, introduced by an additional translation process: this decision of the Ω team has no political connotation ☺, it comes just from the fact that French letters with tréma are far rarer than German letters with Umlaut.

(description taken from the source file, `germanb.dtx`). Maybe it is not very clear from the description above, when you should use `"z` instead of `"s`: in fact, they both produce exactly the same result, but when they are uppercased, the first becomes “SS” and the second “SZ” (for example, „MASSE“ comes from „Masse“ and „MASZE“ from „Maße“).

It should be noted also that although “French double quotes” are called “French”, they are not typeset with the proper French spacing, as in the French Ω-Babel style.

The UC fonts will also contain an “ft” ligature and closely kerned versions of “ck” and “ch” (as requested by Frank Mittelbach in 1991), and the possibility of typesetting with a “long s” as in old German (the “long s” *is* in fact a Unicode character, and the UC fonts will contain all kind of ligatures “long s + i”, “long s + l”, ...).

## 4.4 Dutch

Dutch features are loaded by the `dutch` option. Once again, all original Babel aliases have been kept. We have included one automatic transformation: `ij` and `IJ` produce the “ij” ligature, in lower and upper form. To avoid this ligature, it suffices to introduce an empty group between the letters, or any other “invisible” command.

Dutch is a fine example of separation of *aliases* and *inherent transforms*. Consider for example the case of the letter “ï” in the word „ongeïnteresseerd”. When this word is hyphenated as „onge-interesseerd”, the letter “i” loses the dieresis (the dieresis is there to indicate that “ei” is not a diphthong; by hyphenating at that location there is no doubt any more that this is the case, so the dieresis is useless).

On the TEX level, Babel solves this problem by using a `\discretionary` command. This is an essential transformation, inherent to the Dutch language. Hence, we have both an alias and an inherent transform:

$$\texttt{"a} \xrightarrow{\text{alias}} \texttt{0x00e4} \xrightarrow{\text{inherent}} \texttt{\textbackslash allowhyphens\%}$$

```
\discretionary{-}{U}{^^^^00dc}%
\allowhyphens
```

## 4.5 Portuguese

Portuguese features are loaded by the `portugues` option. Portuguese has just a few aliases, similar to those of the German style. The important fact is that Portuguese has a special inherent transform to avoid ‘ff’, ‘fi’, ‘fl’, ‘ffi’ and ‘ffl’ ligatures. We obtain this by inserting between these letters the character ZERO WIDTH SPACE, which—when going to the DC output fonts—becomes a `\kern0pt` command.

This is not the best way of solving this problem: we are forced to use DC fonts (until we reach step 2 of Ω-Babel development), and these fonts have an automatic ligaturing mechanism to produce the ligatures: by inserting a zero-length skip we avoid the ligature, but lose a possible kern between the letters.

The forthcoming UC fonts will have no internal ligatures: all ligatures will be provided by translation processes, so that we can activate and deactivate them *ad libitum*.

## 4.6 Catalan

The Catalan features are loaded by the `catalan` option. We provide the following inherent transforms: `l.l` and `L.L` which produce “l·l” and “L·L” respectively. Of course, these can also be typeset by using the aliases `"ll` and `"LL`, as in the original Babel style.

We must point out that for the moment this character is produced in a very unorthodox way (28 lines of code!!). We will obtain *real* typesetting of Catalan only after step 2 of Ω-Babel development, since “l·l” and “L·L” are characters of the UC fonts (this is not the case for DC fonts: the dots have to be dragged to the right place...).

## 4.7 Spanish

The Spanish features are loaded by the `spanish` option. All aliases requested by the author of the original Babel style have been included in the Ω-Babel adaptation, *except one*: `~n ~N` for ñ, Ñ. In TEX, the character `~` is traditionally used to obtain a non-breakable space, and this should be valid for all languages.

In some languages (like in Greek) one can argue that letters carrying the tilde accent do not appear at the beginning of a word. This is unfortunately not the case of Spanish (we found four such words in a pocket dictionary: *ñandú*, *ñoño*, *ñudo*, *ñudoso*, there might be more...). That’s why we decided not to retain this alias for Spanish (fortunately, the author of the Spanish style also has a second alias for the same letter: `'n 'N`).

## 4.8 Turkish

The Turkish features are loaded by the `turkish` option. This language has two versions of letter “i”: with and without dot. The glyph “I” is the uppercase form of letter “ı” and the glyph “İ” is the uppercase form of letter “i”. The distinction is essential, both for hyphenation and for upper/lowercasing. To solve this problem, we have defined new codes in (the private zone of) Unicode, for the *Turkish uppercase form of ‘i’* and the *Turkish lowercase form* *of ‘I’*. Whenever you switch to Turkish, the inherent translation process sends all of your (otherwise innocent) ‘i’s and ‘I’s to these far away locations in the code (`0xe083` and `0xe084`, that is decimal 57475 and 57476!!), so that Ω has no doubt on the information it is processing. Of course, after processing, we return to the usual ‘i’ and ‘I’ glyphs of the DC fonts.

Like French, Turkish needs special punctuation spacing for the colon, exclamation mark and equal sign. This is also done through the inherent translation process.

Finally, like Portuguese, Turkish avoids ‘ff’-like ligatures: the same methods are applied.

### 4.9 Breton, Danish, Estonian, Finnish, Galician, Polish, Slovene, Upper Sorbian

These language styles use aliases, mostly similar to those of the German style (Breton aliases and inherent transforms are similar to the French ones, a coincidence?). We have adapted these aliases; there have been no further changes.

### 4.10 Bahasa, Croatian, English, Czech, Irish, Italian, Lower Sorbian, Hungarian, Norwegian, Romanian, Scottish, Slovakian, Swedish

Last, but not least, these language styles use no aliases at all (either due to their simplicity, or to the wishes (or keyboard facilities?) of the respective authors). They all have the same trivial alias and inherent translation processes.

### 4.11 Forthcoming language styles

To prepare a Babel or Ω-Babel language style the most difficult task is to create the hyphenation patterns. We already have Welsh, Esperanto and Lithuanian hyphenation patterns. As for non-Latin alphabet languages, we have hyphenation patterns and fonts for Greek, Russian, Bulgarian and Serbian: all these languages will be covered by Ω-Babel, in step 2 of Ω-Babel development.

## 5 Let’s get technical!

There are two main differences between TEX and Ω: (a) the latter works with bigger numbers (more code positions, more fonts, more boxes, more registers, etc.), (b) it uses *translation processes*. Translation processes are organized in *lists*, which are pushed on a *stack*. Inside a list, a translation process has an ID, a (not necessarily integer) number between 1 and 4095. When you push a stack on a list, translation processes with the same IDs replace each other;

| | |
|---|---|
| Input encoding (for example, Macintosh Standard Roman → Unicode, or codepage 437 → Unicode, etc.) | `500` |
| Aliases (for example, `"a` → ä, `"z` → “special” ß, etc.) and inherent transforms (for example, French guillemets, Dutch ‘ij’ ligature, avoiding ‘ff’ ligatures in Turkish and Portuguese, etc.) | `1000` |
| Hyphenation (not available yet) | `2000` |
| Contextual analysis (Greek final sigma, Hebrew final letters, Arabic alphabet contextual forms, etc.) | `2500` |
| Case change (lower to uppercase, case inversion, etc.) | `3000` |
| Output encoding (for example, Unicode to DC, or Unicode to UC, etc.) | `3500` |

Table 2: IDs for usual translation tasks.

when there is a process of a given ID in a given list, and you push upon it a list which has no process with the same ID, then that process remains active.

So, for example, if you attribute ID 2500 to a contextual analysis process, and ID 1000 to an aliasing and inherent transform, then you can push the latter upon the former: both will still remain active. But if you wish to change the language, you will push a process of ID 1000 (for the new language), and it will replace the previous one.

This means that we have to be consistent in our choices of translation process IDs: we can always insert additional processes between the existing ones, but to be able to push them away automatically when conditions change (for example when we switch languages), we must keep the same IDs. In Table 2 we present some IDs we have chosen for the tasks described in this paper.

So, in each Ω-Babel language style a CTP list is defined (CTP stands for “compiled translation processes”), consisting of a CTP of ID 1000 which contains both aliases and inherent transforms.[4] Each time we open a new `lang` environment, we push new CTPs over it; when we leave the environment it is automatically popped off the stack (CTP loading obeys TEX’s grouping rules).

### 5.1 Description of available CTPs

#### 5.1.1 Input encoding CTPs

In Table 3 we list the input encoding CTPs we have prepared.

[4] The original idea was to split this CTP into one for aliases and one for inherent transforms, of IDs 1000 and 1500; but for speed reasons we have decided to bundle these two into a single CTP.

| | |
|---|---|
| ISO Latin-1 | `lat1uni.ctp` |
| ISO Latin-2 | `lat2uni.ctp` |
| ISO Latin-3 | `lat3uni.ctp` |
| ISO Latin-4 | `lat4uni.ctp` |
| ISO Latin-5 | `lat5uni.ctp` |
| ISO Latin-6 | `lat6uni.ctp` |
| Macintosh Standard Roman | `stmacuni.ctp` |
| Macintosh Central Europe | `cemacuni.ctp` |
| Macintosh Croatian | `hrmacuni.ctp` |
| Macintosh Icelandic | `ismacuni.ctp` |
| Macintosh Turkish | `trmacuni.ctp` |
| Windows ANSI | `ansiuni.ctp` |
| IBM codepage 437 | `cp437uni.ctp` |
| IBM codepage 850 | `cp850uni.ctp` |
| IBM codepage 852 | `cp852uni.ctp` |
| IBM codepage 857 | `cp857uni.ctp` |
| IBM codepage 860 | `cp860uni.ctp` |
| IBM codepage 861 | `cp861uni.ctp` |

Table 3: Available input encoding CTPs.

Any suggestions for further enhancement of this list will be welcome (for the moment we are dealing only with Latin alphabet encodings, Greek and Cyrillic will follow, the rest is for... later ☺).

There is not much to say about these translations. In the Macintosh encodings we have chosen to send $\pi$ and $\Omega$ to the corresponding Greek letters in Unicode (although they are supposed to be math symbols), $\Delta$ to the math symbol INCREMENT and *not* to the Greek letter Delta, $\Diamond$ to the geometric shape LOZENGE, and the Macintosh Apple to the (private) Unicode character `0xe090` (yes, we have reserved a slot for the Apple logo: after all Yannis owes that to those little machines "for the rest of us"[5]).

In the codepage 437 to Unicode translation, we send `0xd5` (a small vertical stroke) to Unicode `0x02c8` MODIFIER LETTER VERTICAL LINE. Once again, any suggestions will be welcome.

#### 5.1.2 CTPs for aliases and inherent transforms

There is one CTP for every $\Omega$-Babel language style, except for Bahasa, Croatian, English, Czech, Irish, Italian, Lower Sorbian, Hungarian, Norwegian, Romanian, Scottish, Slovakian and Swedish which share a minimal CTP called `minalias.ctp`. The name of each CTP is formed by the two-letter ISO code for the corresponding language, as described in Haralambous (1992a), and the extension `.ctp`: for example: `fr.ctp`, `de.ctp`, etc.

[5] No, we have not reserved any slot for the Windows logo ☺.

| | |
|---|---|
| `0xe000` ↓ `0xe0ff` | "Direct-to-Unicode" 8-bit table (see below) |
| `0xe100` ↓ `0xe17f` | Fake ASCII (see below) |
| `0xe180` | Uppercase German ß (glyph: SS) |
| `0xe181` | Lowercase special German ß (`"z`) |
| `0xe182` | Uppercase special German ß (glyph: SZ) |
| `0xe183` | Uppercase Turkish ı (glyph: I) |
| `0xe184` | Lowercase Turkish İ (glyph: i) |
| `0xe185` | Uppercase letter kra (glyph: K) |
| `0xe186` | Uppercase Afrikaans ’n (glyph: ’N) |
| `0xe187` | Uppercase long s (glyph: S) |
| `0xe188` | Uppercase ẖ (glyph: H̱) |
| `0xe189` | Uppercase ẗ (glyph: T̈) |
| `0xe18a` | Uppercase ẘ (glyph: W̊) |
| `0xe18b` | Uppercase ẙ (glyph: Y̊) |
| `0xe18c` | Uppercase ả (glyph: Ả) |
| `0xe18d` | Uppercase ȷ (glyph: J) |
| `0xe18e` | TeX character ȷ |
| `0xe18f` | Uppercase ǰ (glyph: J̌) |
| `0xe190` | Apple Macintosh logo |
| `0xe191` | Fixed case modifier |
| `0xe192` | Fake case modifier |
| `0xe1a0` ↓ `0xe1cf` | Needed for Greek and Armenian |

Table 4: How $\Omega$ uses the Unicode private zone.

Language styles with no inherent transforms (other than '`ff`' ligatures, `---` punctuation etc.) use the common CTP `minilang.ctp`.

#### 5.1.3 CTPs for hyphenation

These are under development.

#### 5.1.4 CTPs for contextual analysis

These are also under development; the Arabic one is up and running, but we have to bundle the complete package.

#### 5.1.5 Case change CTPs

We have prepared CTPs for uppercasing and lowercasing, called `uppercas.ctp` and `lowercas.ctp`. These are symmetric: both uppercase followed by lowercase, as lowercase followed by uppercase are equal to the identity. This has been achieved by attributing a few special code positions in the private zone of Unicode. In Table 4 the reader can find an overview of the private zone of Unicode as we are using it at the present time.

This table will certainly change in the future, but we promise to keep the changes upwards compatible.

What are the "Direct-to-Unicode" slots for? Suppose you are writing a macro which will produce a character in the 8-bit range of Unicode (which is similar to the ISO Latin-1 encoding), for example Ç, which has code `^^c7`. If you write `^^c7` or `\char"C7` (or Ç, using an ISO Latin-1 screen font), then Ω will pass this character through the input encoding filter: if your macro is used on a machine with a different input encoding, the output will not be the same. One needs a way to produce a Unicode character `^^^^00c7` independently of the input encoding. This can always be done by temporarily deactivating the input encoding: a more simple solution is to apply an offset of `0xe000` to the codes that must remain invariant by the input re-encoding process: use `^^^^e0c7` instead of `^^^^00c7` in your macro, and the result will always be Ç (of course, the end user will be able to utilize macros like `\c{C}`: this is how these macros are defined in our system).

And what is this "fake ASCII" all about? It is a trick similar to the previous one, but it sends the information further down the chain of translation processes. It is used to send information to one of the last translation processes (for example the output encoding translation process) which should not be modified at all by the intermediate processes, *while staying in the same buffer*. A buffer is a sequence of bytes read by Ω and processed by translation processes: Ω will stop reading a buffer as soon as it encounters a character that is not of catcode 11 or 12. Why do we need that?

Suppose you are writing Arabic. Contextual analysis is entirely done *inside* a buffer.[6] We may also want the central letter of a word to be typeset in red[7] (this can be of great help in an Arabic grammar book). Inserting a `\red` command inside the word will completely break the contextual analysis. We solve the problem by transposing the characters `\`, `r`, `e`, `d` to the private Unicode zone (by a fixed offset of `0xe000`). The contextual analysis translation process is aware of the fact that characters in that range shall not interfere in the process of performing contextual analysis. Once the contextual analysis is done, and our candidate for being red has the right contextual form, we perform the opposite offset and our string becomes once again a regular TEX command. We will return to this in more detail when Arabic Ω is released.

The fixed and fake case modifiers affect the character following them: they are introduced by the `\fixedcase` and `\fakecase` commands, and are only considered by the `uppercas` and `lowercas` translation processes, in the following way: a character following the fixed case modifier is not affected by either `uppercas` or `lowercas`; a character following the fake case modifier (like the 'I' in StudentInnen) is converted "the other way around": `uppercas` will call `lowercas`, and `lowercas` will call `uppercas`, for the specific character only.

The various strange characters you see on the lists are not included in Unicode for various reasons, we need them in particular to keep `uppercas` and `lowercas` symetric.

### 5.1.6 Output font CTPs

We have already written the CTP for the Unicode → DC font encoding: all characters included in the DC font table are used as is, others are constructed by using the `\accent` primitive. For step 2 of Ω-Babel development, we will prepare a Unicode → UC translation process. We expect users to write their own translation processes for other output font encodings.

## 6 Conclusion

We want TEX implementors to join us in the Ω adventure and consider including Ω in their TEX distributions; those based in Web2C should find it very easy. There will be virtual UC fonts, based upon real fonts in the `0x20-0xff` range.

We want DVIware developers to consider making their DVI software compatible with our XVF and XFM files (extended VF and extended TFM), which are 16-bit. For the moment we have extended Peter Breitenlohner's *dvicopy* into a 16-bit version (which we call *xdvicopy*); using this utility we are able to de-virtualize extended virtual fonts into their underlying real fonts, which for the moment are all only 8-bit. De-virtualized DVI files are then compatible with every DVI software in the market. But it would be quicker and more practical for the user if software recognized XVF and XFM files directly (after all, the DVI standard allows up to 32-bit characters: our DVI files conform to the standard).

Finally, we want users to give Ω and Ω-Babel a hard try, and us a hard time in debugging the software, the macros and (later on) the fonts. Ω will soon grow a lot: several Oriental TEX packages are

[6] Otherwise we would have to record somewhere the form of the last-read letter: this is possible of course, but we won't know *why* the buffer was terminated: was it something harmless, like an empty group (in which case the contextuality is kept), or was it something as horrible as a `\newpage` command? In which case we had better finish our word before going to the next page.

[7] This is rendered as a grey scale in *TUGboat*.

already ready and waiting to be adapted to Ω (see references below), but before this happens, we want to be sure that the Latin-Greek-Cyrillic part of it is clean and robust.

## References

Andulem (Amnulehe), A. "The road to Ethiopic TEX". *TUGboat* **10**(3), 352–354, 1989.

Braams, J. "Babel, a multilingual style-option system for use with LATEX's standard document styles". *TUGboat* **12**(2), 1991a.

Braams, J. "An update on the Babel system". *TUGboat* **14**(1), 1991b.

Haralambous, Y. "TEX and those other languages ... ". *TUGboat* **12**(4), 539–548, 1991.

Haralambous, Y. "TEX Conventions Concerning Languages". *TTN* **1**(4), 3–10, 1992a.

Haralambous, Y. *TEX et les Langues Orientales.* Paris, 1992b.

Haralambous, Y. "Typesetting the Holy Qur'ān with TEX". In *Proceedings of the 2nd International Conference on Multilingual Computing—Arabic and Latin script (Durham).* 1992c.

Haralambous, Y. "The Khmer Script tamed by the Lion (of TEX)". In *Proceedings of the 14th TEX Users Groups Annual Meeting (Aston, Birmingham).* 1993a.

Haralambous, Y. "Un système TEX berbère". In *Actes de la table ronde internationale « Phonologie et notation usuelle dans le domaine berbère », INALCO.* 1993b.

Haralambous, Y. "*Indica*, a Preprocessor for Indic Languages—Sinhalese TEX". In *Proceedings of the 15th TEX Users Groups Annual Meeting (Santa Barbara).* 1994a.

Haralambous, Y. "Tiqwah: a Typesetting System for Biblical Hebrew, based on TEX". In *Proceedings of the Fourth International Colloquium "Bible and Computer: Desk and Discipline, The impact of computers on Bible Studies" (Amsterdam).* 1994b.

Haralambous, Y. "Sabra: a Syriac TEX system". In *Proceedings of the First International Forum on Syriac Computing (Washington, D.C.).* 1995.

Haralambous, Y. and J. Plaice. "First Applications of **Ω**: Greek, Arabic, Khmer, Poetica, ISO 10646/UNICODE, etc.". In *Proceedings of the 15th TEX Users Groups Annual Meeting (Santa Barbara).* 1994.

Mittelbach, F. "E-TEX: Guidelines for Future TEX Extensions". *TUGboat* **11**(3), 1991.

Plaice, J. "Progress in the **Ω** Project". In *Proceedings of the 15th TEX Users Groups Annual Meeting (Santa Barbara).* 1994.

Velthuis, F. J. *Devanagari for TEX*, 1991.

Haralambous, Y. and J. Plaice. "**Ω**, a TEX Extension Including Unicode and Featuring Lex-like Filtering Processes". In *Proceedings of the 1994 EuroTEX Conference (Gdansk).* 1994.

◇ Yannis Haralambous
187, rue Nationale
59800 Lille, France
Email: `haralambous@univ-lille1.fr`
URL: `http://www.ens.fr/~yannis`

◇ John Plaice
Université Laval
Québec, Canada
Email: `plaice@ift.ulaval.ca`

◇ Johannes Braams
TEXniek
Kooienswater 62
The Netherlands
Email: `JLBraams@cistron.nl`