Multilingual Support in LaTeX3: What are the Issues?

Marcel Oliver

Email: oliver@member.ams.org

Abstract

This document summarizes the discussion on the LATEX-L mailing list concerning input, output, and internal character encodings in LaTeX. The goal of this summary is to ensure that future versions of LaTeXprovide a superior environment for typesetting non-English, non-Latin, and multilingual documents.

Few of these ideas are mine, but responsibility for inaccuracies and bias lies entirely with me. I encourage everybody to comment on, patch, or amend this document, and will try to keep it up to date for as long as necessary.

1. User visible encodings

I start the discussion with *user visible encodings*, by which I mean encodings consciously encountered when preparing and processing input files. We will defer TeX internal and font encoding issues later to later sections as, at least in principle, TeX/LaTeX/ and auxiliary programs will take care of choosing, displaying, converting, and printing the document in appropriate fonts. The how is of no interest to the user as long as it works. From the technical point of view this division is certainly artificial, but it's important to set a clear target for the ''user experience'' with future versions of LaTeX.

1.1. Input encodings

Currently, it is difficult to enter many non-English or multilingual scripts. Most Latin based languages are well supported, but non-Latin scripts or multiple languages in one document are substantially more difficult to work with. While it is possible to produce high quality print, the way to get there is often not particularly user friendly.

Two input strategies can be used: One can provide plain ASCII input, or type the document in a language specific encoding which is made known to LaTeX via the inputenc package.

— Typing ASCII can be very tedious, and makes it hard to proofread the `.tex` file. Portability is good in theory, but since one must frequently depend on add-on packages which are neither part of core LaTeX nor found in standard distributions, migrating files from one installation to another can be an adventure nonetheless. A document requiring nothing but babel can probably be considered portable, even though Greek and Cyrillic fonts are missing in a default install, for example.

— Setting an input encoding works well for single languages. However, it is not a solution for multilingual work unless the chosen input encoding is effectively UNICODE. Moreover, the current inputenc still misses important encodings; Greek ISO-8859-7, for example, must be downloaded separately.

1.2. Diagnostic messages

Diagnostic messages are often cryptic and therefore less helpful than they ought to be. One fundamental issue is that error messages contain the full expansion of the input stream up to the point when the error is detected, and can thus contain parts of macro definitions. The situation is even worse when the input contains non-ASCII symbols which are echoed by character code. If the script is overwhelmingly non-ASCII, the diagnostic output is often useless. Further, errors and warnings behave differently with respect to non-ASCII characters:

— TeX error messages echo the input encoding.

— Overfull box messages use a format related to the output encoding of the current font.

Example: ''Müll'' in a German Latin-1 encoded file will be echoed as `M^^fcll` in an error message, but as `M^^?ull` in an overfull `\hbox` message—unless T1 font encoding is selected, which returns `M^^fcll` in both cases.

Most TeX implementations now allow messages to be printed without conversion to `^^` format. However, this does not provide sufficient generality for a clean general-purpose solution. Note, for example, that screen output (on Unix) goes to an Xterm, while the input encoding is generated by an editor, which do not necessarily use the same encoding scheme.

1.3. The case for UTF-8

The two issues could, at least in principle, be resolved by leveraging UTF-8 as *the* default input and diagnostic output encoding. The most important arguments are listed here, see [14] for further details.

— UTF-8 encodes UNICODE, hence covers virtually all scripts of this world.

— The UNICODE character model is fairly stable and well documented. (There are still lot's of empty slots that someone will eventually fill. The specification for mathematical symbols, in particular, is not yet complete.)

— All ASCII characters have their usual position in UTF-8. In other words, current ASCII `.tex` files would continue to work without change.

— UTF-8 has 1–6 byte characters, but by looking at a single byte one knows how many will follow. This makes parsing (especially on 8-bit TeX) relatively straightforward. Note that UTF-16 does not have this property, and may better be handled by a front-end filter.

— Support for editing and displaying UTF-8 exists on all major platforms, and is rapidly growing in popularity. Programming library support is also fairly good. Some details on platform support are listed in Appendix 9.

— Diagnostic messages could (although not with current TeX engine) be output in the correct script. This would be a major improvement for users, and fit well into a paradigm ''anything that goes in and comes out is UTF-8, unless specified otherwise.'' (DVI output is not relevant here as it is always handled by special purpose applications.)

1.4. Existing Implementations

— There is an implementation for UTF-8 input on a TeX engine (xmltex by David Carlisle [3]) that also uses UTF-8 internally.

— There also exists a UTF-8 option for the inputenc package [19]. Every UNICODE code point is given a TeX name, which may make it too unwieldy for typical single-alphabet use.

— The ''combining characters'' of UNICODE are difficult to handle with a TeX based parser. It is not possible for TeX to parse UTF-16 correctly (not really essential anyway). See Appendix 6 for details.

— It is not trivial to handle input errors gracefully (i.e. give meaningful error messages) with TeX based parsers.

— Ω as a native UNICODE implementation of TeX is designed to handle general UTF-8 input. More in Section 3.

1.5. Macro names in native scripts

If LaTeX is to be used as a truly international typesetting engine, it is desirable to allow for native macro names. While the occidental scholar, or someone typesetting a single Latin-alphabet language, will find English macro names perfectly acceptable, users of non-Latin keyboards, especially when typesetting right-to-left scripts, are currently at a clear disadvantage.

However, non-ASCII characters in macro names must interact differently with input encoding translations than regular text. The resulting problems have so far been underexplored, with the exception of Javier Bezos' prototype lambda package [2] discussed further in Section 4.3.

1.6. Lost character conditions

The way LaTeX/TeX/ handles lost characters—characters that do not exist in the selected fonts—should be improved. The current behavior is to ignore the character and write a warning message. The following options (or a combination thereof) may be better:

— Substitution with a character from another font (LaTeX can do this for characters represented by encoding-specific commands, but there isn't a mechanism in TeX for explicit character tokens).

— Typeset a suitable representation (e.g., `U+0312` in a suitably distinct font) of the UNICODE for the character. Especially useful in verbatim-like context: it would have eased the compilation of this document in several places.

— An error message.

2. Internal encodings and font encodings

The LaTeX/TeX/ internal character handling is considered a much more important topic of discussion, as arbitrary input and diagnostic handling could, at least in theory, be put on top of TeX, or patched without affecting its inner workings. Apart from not being optimally suited for handling the user interface aspects of ''modern'' encodings like UTF-8, the current TeX character model is deficient in three aspects: Hyphenation, need for virtual fonts, and kerning.

2.1. Hyphenation and virtual fonts

With the proliferation of output encodings and growing support for multilingual multi-encoded documents, hyphenation patterns are increasingly difficult to maintain:

— Internal hyphenation patterns are stored in terms of an output encoding.

— The hyphenation *rules* are also frequently written in directly in terms of the output encoding. Is possible to use symbolic representations of characters (`\ss`) rather than hex code values so that a pattern can be used with different font encodings. However, already at the time of format creation the pattern is compiled into the internal format. In other words, each font encoding must be specified when the format is created, and different font encodings must logically be treated as different languages.

This behavior is logically wrong, as hyphenation has nothing to do with font encoding. One should be able to add a new output encoding without the need for running `initex`. Moreover, new fonts must first be mapped into a TeX specific encoding (OT1, T1, ...) through the creation of a virtual font (VF). While this is a one-time effort and does not directly affect the user, it is nonetheless a complication we may live without. Appendix 8 gives some more details.

2.2. Kerning

The output encoding is limited to 8-bit fonts, which is not be enough to get correct kerning for some languages. Examples:

— Greek, if one requires (as LaTeX currently does) that visible ASCII is part of the font encoding.

— A full swash italic font with automatic ligaturing for uppercases (including accented ones), additional ligature pairs, and special ending characters. (An example could be the full set of AJenson; also see [18]).

— Japanese, to a certain extent. But there are more typographic oddities which can't be handled properly (i.e., automatically) within TeX (Kerning between Japanese characters, Latin characters, and punctuation marks; vertical typesetting, underline, justification by filling space between glyphs by inserting a special filler comparable to the Arabic ''keshideh'', and other issues. Werner Lemberg's CJK package [15] has implemented some of these).

3. Ω

3.1. Principles of Ω

Ω [11] is a UNICODE-capable extension of TEX. It is largely compatible, but adds a number of additional capabilities. This section gives only a brief overview—for a more complete description, see the Ω draft documentation [12].

— Internal 32-bit character representation.

— Features OTPs (Ω Translation Processes) which are ''stackable'' finite state machines that operate on the input stream. OTPs can be used for input and output encoding translations, for ''shape'' changes (case, script variants, etc.), and for handling language typographical features without explicit markup. (For example, German `"ck`, Spanish `"rr`, Portuguese `f{}i`, and Arabic ligatures [10] can all be handled by OTPs.)

— In TEX, only the protected expansion (aka. LICR, see Section 4.1) is under our control. In Ω, once tokens are expanded, primitives are evaluated as in TEX, but chars can be further processed using OTPs. If the result contains macros, these are expanded and evaluated in turn, and so on.

Main advantages:

— Less need for virtual fonts. (''Must I create several hundreds of VF files only to remove the fi ligature?'')

— Reduce the need for active characters [13] and pre-passes.

RP mentions his work on FarsiTEX [9]: Before Ω, he needed a pre-pass to do contextual shaping, and active Tatweels inserted between letters to stretch them to fit the line of text.

3.2. Shortcomings of Ω

In current Ω, OTPs act only in two places, namely when the file is initially read before the input gets tokenized, and when building a horizontal list. The following problems therefore remain.

— The Ω command `\InputTranslation` selects a translation for all the characters of the file—macro names, macro definitions and material for typesetting. The command is primarily intended for ''technical translations'' such as one-byte to two-bytes or little-endian to big-endian, but could

possibly be used for a whole-sale UTF-8 to internal UNICODE translation. As explained in Section 4.5, changing the `\InputTranslation` will cause unexpected interfere with macro expansion.

— If the translation to UNICODE is done via OTPs hooked into the `hlist` builder (as is implemented in LAMBDA, see Section 4.3), then most of the processing is still done in the (typically 8-bit) input encoding, see the discussion in Section 4.4.

— Hyphenation still works as in TeX, namely on the font encoding. The Ω documentation [12] anticipates future work in this area.

— Ω appears to still have the same problems with error and warning messages as TeX.

4. Models for internal character representation

4.1. The current LICR

Current LaTeX has conceptionally only three levels: Source, internal representation (LICR), and output. Protected expansion (`\protected@edef`) transforms the source into an ICR that is conceptionally well-defined—despite its restriction to 7-bit ASCII by the limitations of TeX. Characters which cannot be directly represented this way are kept (e.g., written into `.aux` files) in the form of macros like `\cyrA` for letters of the Cyrillic alphabet.

4.2. Requirements for alternative ICRs

— Internal UNICODE cannot completely replace named symbols and other complex objects. In particular, the set of possible ''embellished'' letters and symbols used in math, while not infinite, is much too large to want to address even using the UNICODE private area.

— The UNICODE character representation is not unique. For example, ''d'' can be coded as `<A>+<COMBINING UMLAUT>` or `<A-UMLAUT>`. Software should behave exactly the same when encountering these two. For proper hyphenation (and possibly other reasons), the character stream has to be *normalized* as documented in the UNICODE specs [6].

— An isolated element of the internal representation must have a unique, semantically well-defined meaning at any time.

— The requirements for a math ICR are different, as the input typically maps directly into glyphs (''anything that looks different *is* different''). In this

sense math is easier, as we do not have to deal with context dependent shape variants, and it seems a UNICODE ICR would not gain much.

— While the idea that OTPs select between variant glyphs in a font seems sound, this must not interfere with the ICR. In other words, the rendering of glyphs goes beyond the ICR, and will always involve contextual analysis (e.g., traditional ideograms in Japanese vs. simplified ones in Chinese). For some interesting background, see [5].

4.3. Javier Bezos' lambda package

Javier Bezos has written a package to exploit the additional features of Ω when running LaTeX on Ω [2]. The name of the package is (maybe unfortunately) lambda, not to be confused with `lambda` the executable, which is the name for calling the standard LaTeX format on Ω (and referred to as Λ in the Ω documentation).

lambda the package is to provide support for multilingual and multi-encoded documents in a clean way, and to supply the necessary interfaces for class and package writers. Javier has released the code as a working proposal, and has stated that he is pausing development pending important design decisions for both LaTeX3 and Ω. Meanwhile he is looking for comments on his code. The following description of lambda draws heavily on Javier's original posts:

Let us recall how TeX handles non-ASCII characters. While TeX can read UNICODE files, as xml|tex [3] demonstrates, non-ASCII characters cannot be represented internally by TeX as such. Instead, TeX uses macros generated by inputenc, and which are finally expanded into a true character (or a TeX macro) by fontenc:

```
é --- inputenc --> \'e  --- fontenc --> ^^e9
```

Even Cyrillic, Arabic, etc., characters are processed this way!

Ω can represent non-ASCII chars internally. Hence, actual chars are used instead of macros (with a few exceptions—OTPs can output control sequences. Note, however, that the characters that are output by an OTP obtain their catcode at the time the replacement is done. This means that ''private'' names containing `@` cannot be used.). Trivial as it seems, this actually makes a *huge* difference. For example, the path followed by é will be:

```
    é--an encoding OTP-|           |-- T1 font OTP-->  ^^e9
                       +-> U+00E9 -+
 \'e --‘‘fontenc’’---|           |- OT1 font OTP -> \OT1\'{e}
```

The name ''fontenc'' for the input translation is used because of the technical similarity of the code. This reflects that conversion to UNICODE happens *after* full expansion of the input stream. In other words, things like `\'e` are preserved when written to an `.aux` file or moved around, as the following table illustrates:

```
=================================================
           A       B        C       D         E
-------------------------------------------------
LaTeX  a)  "82     \'e      *   - - - - - >   "E9
       b)  \'e     \'e      *   - - - - - >   "E9
=================================================
Lambda a)  "82     "82      "82     "00E9     "E9
       b)  \'e     \'e      "82     "00E9     "E9
Now        \'e     \'e      e"0301  "00E9     "E9
```

A: The source file.

B: ICR, as created by `\protected@edef` or `\protected@write`. We see that LaTeX has a unique 7-bit ASCII ICR, while lambda operates mostly on the encoding of the input file.

C: Evaluation, with fully expanded tokens. In TeX this is is the final step (= step E in Ω) with the font codes. Note that in the current implementation (row ''Now''), lambda does *not* convert `\'e` to ''é'', but to `e U+0301` (i.e., the corresponding combining char). Normalization takes place in the internal UNICODE step D. In fact, the definition of `\'` in `la.sd` is:

```
\DeclareScriptCommand\'[1]{#1\unichar{"0301}}
```

D: After input encoding translation. (Note that this is already ''output'' as far as the high level format LaTeX/lambda is concerned! It is still possible that OTPs for case change, contextual shaping, etc., act at this level.)

E: After font encoding translation and the final step in Ω.

OTPs can be divided into two groups: those generating UNICODE from arbitrary input, and those rendering the resulting UNICODE using suitable (or maybe just available) fonts. The UNICODE text may also be analyzed and transformed by external OTPs at the right place. lambda further divides these two groups into four:

— Encoding: convert the source text to UNICODE.

— Input: set input conventions. Keyboards have a limited number of keys, and hands a limited number of fingers. We want to provide an easy way to enter UNICODE characters using the most basic keys of keyboards (which means ASCII characters for Latin keyboards). Examples could be:

- `---` → em-dash (a well known TEX input convention)
- `ij` → U+0133 (in Dutch)
- `no` → U+306E (the corresponding Hiragana character)

Now we have the UNICODE (with TEX tags) memory representation which has to be rendered:

— Writing: contextual analysis, ligatures, spaced punctuation marks, and so on.

— Font: conversion from UNICODE to the local font encoding or the appropriate TEX macros (if the character is not available in the font).

This scheme fits well in the UNICODE design principles. (UNICODE is designed to deal with memory representation, while text rendering or fonts are left to ''appropriate standards''. Hence, most of so-called UNICODE fonts cannot render properly text in many scripts because they lack the required glyphs.)

4.4. Critique of ''late'' conversion to UNICODE

The strategy of ''late'' conversion to UNICODE as employed by lambda (step D in the table of the previous section) has been criticized for a number of reasons.

— Using OTPs in this way means that they must act on arbitrary byte sequences rather than a well defined Ω internal character representation (OICR). This however is wrong, at least conceptionally.

— Transformations of character token strings (e.g., reading from and writing to `.aux` files) can't be controlled by OTPs unless we (pretend to) typeset something. A large amount of document processing (building a table of contents or arranging data for page representation) is concerned with character string manipulation not related to typesetting at all. Thus is seems interesting to think about whether or not a similar concept (not necessarily the same!) should be made available for this part of the process. This could be an OTP aware version of `\edef`.

— Normalization takes place too late. For example, LATEX code that needs to determine if two pieces of text are equal will break.

4.5. Critique of current Ω's input translations

The first problem is that switching the `\InputTranslation` affects code and data alike, and does not apply to the results of macro expansion:

```
\def\myE{E}
\InputTranslation <an encoding>
E\myE             % only explicit I is transcoded
```

The following case is more severe. Changing the translation OTP does not help when the input is already tokenized:

```
\ocp\OCPa=inutf8   % OTP for UTF-8 input

\def\foo{abc^^e4d} % default seems to be Latin-1
\show\foo

% the following fails and can't be corrected later on:
\def\bar{ab\InputTranslation currentfile\OCPa c^^c3^^a4}
\show\bar
```

(Note that in this example the escaped character codes are assumed to be single 8-bit characters in the input file.) The macro `\bar` will now contains the tokens

```
\bar=macro:
->ab\InputTranslation currentfile\OCPa c^^c3^^a4.
```

Thus if you use `\bar` later on you will get the wrong characters because the input was umlaut-a in UTF-8, but what is stored in `\bar` are the *two* characters uppercase-A-with-tilde and currency-sign. Furthermore, if `\bar` is used anywhere, it will change the input translation from the next line on to UTF-8. This could be in a completely different file.

Since we have been asked to provide input encoding changes for LaTeX within paragraphs, e.g., for individual words, something like this would happen if such a change appears, say, inside the argument of `\section`.

One possible solution: Not reimplement `\InputTranslation` to select an encoding specific OTP, but select a single OTP that does something equivalent to the current active character mechanism where, as is the case now, the exact meaning of a character depends on the current definition of a macro. Although imperfect, this approach doesn't introduce any new problems. People who need something better can always abandon the legacy encodings and use UNICODE directly.

5. Transition issues

5.1. Compatibility Goals

— Input file compatibility: Reasonable LaTeX 2ε files should run without problems in LaTeX3. ''On the other hand, independent of any encoding issues, any successor to LaTeX 2ε will have to make a cut, in my opinion or else it will not be much better. Realistically, compatibility is probably less than we tried for switching from 209 to 2ϵ. In particular, there will need to be a heavy organized rewrite of external packages, perhaps even financially supported.'' (FM)

— Pixel compatibility of the generated output: Desirable (old documents should not suddenly get bad line-breaks), but not absolutely crucial. One could always keep an old teTeX tarball for those special cases where strict identity is essential.

LaTeX *is* currently being used as an archival format (arXiv, for example), so there should not be unnecessary breakage.

5.2. Must-haves for a new TeX engine

If TeX is to be retired as the underlying engine for a future LaTeX, the following are essential.

— PDFTeX functionality. Although PDF for printing can easily be produced via the dvi–ps–pdf route, currently only PDFTeX can break embedded hyperlinks correctly across lines. The problem is that, unlike color support where the beginning and end of a color region are just marks, a PDF hyperlink is a rectangular region, so if a link spans more than one line, multiple links need to be created. The geometry of the lines, however, is hard to determine from the `.dvi` file.

(The PDFTeX maintainer most likely faces the same dilemma: To jump or not to jump. Synchronizing any such decision would be important.)

— Must be available on all major and minor platforms.

Ω is *now* available on Web2C based installations, which includes teTeX for Unix (Linux), MiKTeX and fpTeX for Windows, as well as a recent port to MacOS. Commercial TeX systems may not be as far. Textures, which has a respectable user base on MacOS, appears to be based on a heavily modified version of TeX—it has a flash mode where the output is computed real-time and shown in a different window.

— Long-time stability: The engine should not change substantially underneath LaTeX. Although Ω has matured for some time now, there are still a number of fundamental deficiencies (Section 3.2) whose eventual resolution is tied together with the high level format in nontrivial ways.

— ε-TeX [8] has a few items that would help, but nothing indispensable, and is not currently considered in a state to build upon.

5.3. Impact of UTF-8 on LaTeX on Ω

Let's assume we moved to some version of Ω with UTF-8 as the default input encoding (which appears to be a reasonable long-term solution to shoot for). What issues are likely to arise in the transition?

— Any existing otherwise LaTeX3 compliant `.tex` would work. If it's plain ASCII, it would work anyway, if it's neither ASCII nor UTF-8, it could, as is the case now, load a suitably adopted `inputenc`.

— Upgrading to a new LaTeX would need to be accompanied by the installation of new executables in addition to the class and style files of core LaTeX3. This may be less of an issue as conveniently packaged TeX distributions are increasingly used.

— Significant, although not dramatic, performance hit. Λ takes about 1.9 to 2.7 times as long as LaTeX (the largest slowdown is seen in files with heavy use of `\boldsymbol`, which is a big performance hog in any case). This is negligible considering the growth in average machine performance of the time scale that LaTeX has been in existence. . .

— Most English language documents contain accented letters in names or other foreign words. Default LaTeX will not hyphenate such words, so the change will relieve authors from taking explicit action in such cases (i.e., learning about font encodings or specifying break points).

— Exchange of multilingual documents will drastically improve, as there will be one standard way of doing things. Mainstream users will be able to easily process `.tex` files containing non-Latin characters (e.g., Chinese authors' names, even when the main text is English).

— The current Λ (=LaTeX on Ω) has a high degree of compatibility with standard LaTeX (usually, but not always, generating bit-identical output). However, it is clear that the high level format must change in some way to take advantage of the features that Ω offers, which is likely to break lots of third-party packages. It is not yet clear how much compatibility must be sacrificed.

5.4. Arguments in favor of jumping *now*

— If LaTeX does not get state-of-the-art multilingual support with version 3, it may be a long time before another such major change.

— Basing LaTeX on Ω poses a hen-and-egg problem that will not go away automatically. Ω will only become completely stable if there is unequivocal support from the user interface community (i.e., the LaTeX people), and LaTeX needs some of the features Ω provides to become a serious multilingual typesetting system.

— UNICODE is currently receiving a lot of attention and publicity. So it may be advantageous to ride that wave, in particular as it seems technically sound (within the constraints of providing upward compatibility from various legacy standards—for comments on some of the limitations of UNICODE see [16]).

5.5. Soft transition

We could advance LaTeX and Λ in parallel, i.e., have a version of LaTeX for TeX and one for Ω, until the development of both Ω and LaTeX-on-Ω has sufficiently stabilized to allow a complete switch. The core LaTeX team considers this strategy the most realistic.

— LaTeX3 for Latin alphabet languages could probably be completed sooner, and does not depend on the schedule of a LaTeX external development effort.

— So one should try for finding an LICR that can be incorporated in an Ω LICR in a way that it would be transparent for those parts of LaTeX which do not deal with features Ω provides alone.

— A diversion of paths between the two systems must be avoided. ''It would bring Ω into a corner not getting enough users and wouldn't be good for LaTeX on TeX either. It should be possible to build on identical principles even if some of the kernel is technically differently solved. As long as the outer and inner (i.e. the slightly higher) interfaces are identical, one can keep both user groups together until it is possible to switch.'' (FM)

— A soft transition has the potential danger that a suboptimal design gets frozen, as some aspects must remain untested until the engine is replaced.

6. Parsing UTF in TeX

Some details (taken more or less literally from contributions by RP and DC) about the mess when trying to parse UTF-8 correctly using the current TeX engine.

The main problem are combining characters. In principle, every character could be made active to look forward to find the combining character sequence after it, and ''put it over its own head''. One needs to loop until a non-combining char is found.

The hard bit is that having made every character active, `\begin` no longer parses as the `begin` token but as `\ b e g i n`. Therefore one has to make the active definition of \ look ahead to grab all the ''letters'' where ''letter'' means those characters that were catcode 11 until you made them 13, so you have to maintain a list of all those, and check one by one with what's in the token stream. Similarly, matching `{ }` no longer works (unless you cheat and leave those catcode 1 and 2), and in the end you have to write TeX's tokeniser in TeX. Which is possible but not especially fast and hard to do without breaking some add-on LaTeX package, somewhere.

Incidentally, one reason why xmltex [3] can not support UTF-16 is that TeX buffers to the end-of-line marker (`^J` or `^M`) and throws away any bytes with value 32 that occur at the end of this buffer, which might just be half of a 16-bit quantity that you'd rather keep. There's no way to control this behavior from within TeX.

7. Internal UNICODE in TeX

Achim Blumensath: ''Some rough ideas how to implement UNICODE support in TeX.''

— Internally UNICODE characters can be encoded as command sequences of the form `\HEX_CODE`, i.e., 'A' would become `\0041`.

— Each font would define these sequences appropriately, i.e, `\def\0041{A}`. Characters not included in the font would raise an error message.

— To convert the input file to the internal representation one could write a preprocessor in TeX which is invoked by the `\documentclass` command. That's IMHO the easiest way and I don't think the run-time penalty would be that great. The preprocessor should leave command sequences and braces alone, i.e. `\begin{bar}` would become `\begin{\0062\0061\0072}`.

The only problem I see with this approach are `\catcode`-changes.

8. The virtual font mechanism

Lars Hellström, maintainer of the of the fontinst package [20] writes:

This is certainly a design flaw in TEX which should be fixed in its successor, but it is nothing you cannot live with as a user. Even when you have to use one `\language` code for each language–encoding pair you have you don't reach TEX's limit for the number of different languages, because there are never more than a handfull of font encodings that are useful for any given language (possibly with the exception of the a–z only languages, but then you can reuse the same hyphenation patterns anyway). The problems that exist are more at an administative level, which means things get more complicated for those that implement language support in LATEX, but those are manageable.

It's *not* hard to make a virtual font, provided you use some tool like fontinst to do all the file format conversions and shuffling numbers around (without such a tool I can agree it is a nightmare). I certainly found it much easier to use fontinst ''out of the box'' than I found babel (although I may be a bad example as I have now ended up being the maintainer of fontinst). What has been missing though is the basic support for non-latin scripts, but there the situation has improved lately with Vladimir Volovich's T2 package [21] which covers the cyrillic script.

Support for other scripts (even if it is no more than the basic descriptions of some encoding as an `.etx` file) is welcome and will be added to the distribution if only someone contributes it. I suspect a partial reason noone seems to have done so for the other scripts is that you need to understand a lot of esoterical matters (many of which are poorly documented, or at least very hard to find the documentation for, if documented at all) to develop support for new encodings (whether for LATEX or for fontinst) and thus the set of people who are qualified to do it may in many cases be almost empty. But in any case it is a onetime effort; if you do it for one font for a script, there's very little work involved in later doing it for any other font for that script (unless you're a font freak who absolutely has to produce the very best results with the glyphs available in the base fonts).

My experience is that the driver map files are troublesome much more often than the VF files are, but that problem should also become less important, as fontinst v1.9 can now write the necessary map files for e.g. dvips as well.

9. Platform support for UTF-8

''How many of the people on this list can easily (in their favorite editing system) edit or generate a UTF-8 encoded file? Hands up?''

— The standard encoding of BeOS is UTF-8.
— Xterms in XFree 4.0 are UTF-8 ready, so they can be expected to soon become default in all major Linux distributions.
— Emacs [7] has had the basic capabilities in place for some time. However, an add-on (Mule-UCS [17]) is necessary to read and write UTF-8. Surprisingly, the current Emacs 21 beta does come with UTF-8, either, although future support seems to be planned, see [4].
— TeTeXdit Plus (there are other editors on the Mac which allow to save in UTF-8 encoding).
— The AlphaTk text editor [1], available on any platform that has Tcl/Tk.

Comment (JB): ''I think the question is wrong. The right one is, how many of the people on this list will use an editor which can generate a UTF-8 encoded file *within 4 or 5 years*?''

10. Ligatures vs. Kerning

Ω is moving towards using OTPs for selecting glyph variants and ligatures. In TEX, however, both ligature and kerning information is contained in the `.tfm` files (TEX font *metrics*). Michael Downes gives a historic perspective of why this is so:

The pattern-matching required to identify ligature points is almost identical (in the standard English cases) to the pattern-matching required for kerning.

For kerns between pairs of characters, it is natural to store the information in the `.tfm` file because it is highly dependent on the glyph shapes. Then one needs to run some sort of pattern-matching process to catch pairs of characters in the typesetting sequence in order to insert kerns where applicable. For ligatures a very similar pattern-matching process is needed, only somewhat more generalized. (For more complicated requirements one needs something even more general, like the Ω OTPs.)

The idea of repeating the similar processing in two separate steps instead of combining them as much as possible into a single subroutine would doubtless have seemed horrifyingly wasteful to Knuth. This would fall within the fabled ''inner loop'' that he mentions so often in `tex.web` as an area of special concern.

And then it is natural to store the ligature pattern data in the same place (the `.tfm` file) to make using it as simple as possible.

But consequently if one wants to typeset some material with ligatures turned off, the need to call a different ''no-ligatures'' font tends to be a real hindrance in practice.

Contributors

Hans Aberg, Donald Arseneau, Barbara Beeton, Javier Bezos, Achim Blumensath, Thierry Bouche, David Carlisle, Alexander Cherepanov, Michael John Downes, William F. Hammond, Lars Hellstrfm, Werner Lemberg, Frank Mittelbach, Marcel Oliver, Roozbeh Pournader, Bernd Raichle, Chris Rowley, Apostolos Syropoulos, Karsten Tinnefeld.

Bibliography

[1] Alphatk text editor home page.
`http://www.santafe.edu/~vince/alphatk/about.html`

[2] J. Bezos-López, lambda: Language selection system for Ω.
`http://perso.wanadoo.es/jbezos/archive/lambda.zip`

[3] D. Carlisle, xmltex: A system for typesetting XML files with TEX.
`http://www.ctan.org/tex-archive/macros/xmltex/`

[4] O. Cheung, *Unicode encoding for GNU Emacs*.
`http://www.cs.uu.nl/~otfried/Mule/`

[5] D. Connolly, *''Character Set'' Considered Harmful*, expired HTML working group draft, Internet Engineering Task Force, 1995.
`http://www.w3.org/MarkUp/html-spec/charset-harmful`

[6] M. Davis and M. Dürst, *Unicode Normalization Forms*, Unicode Standard Annex #15.
`http://www.unicode.org/unicode/reports/tr15`

[7] Free Software Foundation, *GNU Emacs page*.
`http://www.gnu.org/software/emacs/`

[8] *varepsilon*-TEX Project Team, *The varepsilon-TEX reference site*.
`http://www.brics.dk/~engberg/home_export_share_TeXdoc/doc/html/e-tex/etex.html`

[9] FarsiTeX Project Team, FarsiTEX: A Persian/English bidirectional typesetting system based on TEX.
`http://farsitex.sourceforge.net/`

[10] Y. Haralambous, *Simplification of the Arabic Script: Three Different Approaches and their Implementations*.
http://genepi.louis-jean.com/omega/arabic-simpli98.pdf

[11] Y. Haralambous and J. Plaice, *The Ω Project Home Page*.
http://omega-system.sourceforge.net/

[12] Y. Haralambous and J. Plaice, *Draft documentation for the Ω system*, Version 1.8, March 1999.
http://na.uni-tuebingen.de/~oliver/latex/doc-1.8.dvi

[13] Y. Haralambous, J. Plaice, and J. Braams, *Never again active characters! Ω-Babel*, TUGboat **16** (1995), No. 4, 418–427.
http://genepi.louis-jean.com/omega/never-again.pdf

[14] M. Kuhn, *UTF-8 and Unicode FAQ for Unix/Linux*.
http://www.cl.cam.ac.uk/~mgk25/unicode.html

[15] W. Lemberg, CJK: East Asian fonts.
http://www.ctan.org/tex-archive/fonts/CJK/

[16] F. Mittelbach and C. Rowley, *Application-independent representation of multi-lingual text*, 10th International Unicode Conference, Mainz, 1997.
http://na.uni-tuebingen.de/~oliver/latex/rowley-mittelbach.dvi

[17] Mule-UCS FTP site.
ftp://ftp.m17n.org/pub/mule/Mule-UCS/

[18] S. Toledo, *Exploiting rich fonts*, TUGboat **21** (2000), No. 2, 121–128.

[19] D. Unruh, ucs: Support for using UTF-8 as input encoding in LaTeX documents.
http://www.unruh.de/DniQ/latex/unicoden/

[20] A. Jeffrey, S. Rahtz, and U. Vieth, fontinst: Simplify the installation of PostScript or TrueType text fonts.
http://www.ctan.org/tex-archive/fonts/utilities/fontinst/

[21] V. Volovich, T2: Standard Cyrillic font encodings.
http://www.ctan.org/tex-archive/macros/latex/contrib/supported/t2/