Programming extensions needed in Ω

Frank Mittelbach Chris Rowley

Contents

1 Introduction and Motivation

Making Ω the future platform for LaTeX development has two separate goals:

- extending the TeX programming machinery to better support the LaTeX kernel and its current general typographic requirements;
- providing a platform for typesetting a wider range of documents and higher quality, in particular languages currently not (or not fully) supported by LaTeX;

The second item requires several new concepts discussed in Tsukuba and Metz; their refinement and finalisation will need a lot of further work (and hence time).

The first item, programming extensions, on the other hand is mostly either already completely specified (and partly implemented, e.g. in eTeX). Most importantly, they are urgently needed to allow us to move LaTeX kernel (and subsequently package) development to the new system.

Whilst we are very interested in both of these items, and we should very much like to see the LaTeX kernel and the quality/language related concepts being developed in parallel, for the sort/medium term development LaTeX/Lambda to a reasonable and useful time-scale the first item needs to be given priority right now.

This paper will therfore deal largely with the relatively simple (we hope) technical programming extensions.

2 Programming functionalities required for first prototype of Ω

The first extended prototype of Ω should include all items listed in section **??** (i.e., all that is in eTEX and applicable to Ω) and, when completely specified, the extensions listed in sections **??**, **??**, **??**, and **??**.

All other implementation suggestions from section **??** should be considered, but possibly at a later stage.

In addition, the first proto-type should include internal data structures for accessing and manipulating lists of "balanced text" (i.e. elements are list of tokens with balanced braces). Also needed are property-lists (associative arrays, directories). For both data structures full specs can be provided.

3 Current eTEX features

This section discusses all the features of current eTEX that we think should get included in Ω in order to make it a better basis for macro packages such as LATEX.

It uses the eTEX manual [**?**] as its basis and follows the section numbering in that document.

3.1 Section 1 + 2 + 3.1 — Introduction, Generating eTEX, Compatibility and Extended Mode

Ignore. No special compatibility mode conventions are needed when loading or generating formats (as described in section 2.2) because Ω is sufficiently different from TEX that a mode emulating TEX to 100% would be strange (and probably impossible anyway).

3.2 Section 3.2 — Optimisation

Such optimisations could be added but may not be applicable (Ω extensions/changes to save stack handling may have changed the data structure here). In any case these are not necessary in a first prototype since they do affect usability.

3.3 Section 3.3 — Tracing and Diagnostics

Full implementation required: these are extremely useful.

3.4 Section 3.4 — Status Enquires

Ignore `\eTeXversion` `\eTeXrevision`. Consider providing `\Omegaversion` and `\Omegarevision` with a similar meaning instead.[1]

Implement `\interactionmode`.

Implement `\currentgrouplevel` and `\currentgrouptype` but also consider providing a way to query the whole stack of groups since information that the current group is, say, "simple" is not very helpful in most cases.[2]

Implement `\currentif...` although we are not sure how useful those queries can be in practice.

Implement `\lastnodetype`. Again this has only limited applications unless Ω also provides additional primitives that allow the manipulation of nodes (beside `\unkern` ... which are already provided by TEX).[3] This will of course need rethinking if the internal data structures for 'formatting lists' change a lot.

Implement `\fontcharht` and friends (as far as they make sense) but rethink them in terms of Omega font resources and the output side of Ω.

Implement `\parshapeindent`, `\parshapelength`, and `\parshapedimen`.

3.5 Section 3.5 — Expressions

Implement something better :-)

In particular, even if `\dimenexpr` is needed to trigger the parsing (so that in circumstances where full control is available the parsing can be optimised) the algorithm should be clever enough to figure out that a sub-expression is of type number etc.

Perhaps some proper delimiting would help (unlike TEX's current number scanning!).

Should stay expandable (if possible)!

Implement `\gluestretch`, `\glueshrink`, `\gluestretchorder`, and `glueshrinkorder`.

Implement `\gluetomu` and `\mutoglue`.

3.6 Section 3.6 — Additional Registers and Marks

Additional registers are already in Ω; perhaps look at the sparse array implementation?

[1] Or is there a more general and useful concept?

[2] Information might be hidden on TEX's save stack and not easily available without storing it explicitly elsewhere.

[3] There might be technical reasons why TEX doesn't provide such functionality (e.g., floating-point arithmetic problems in case of accents (!); also one would need to think about ways to store/assign things like whatsit nodes (data structure for this?) and even glyph nodes.

Implement mark classes.

3.7 Section 3.7 — Input Handling

`\readline` doesn't seem to give anything special but should probably be implemented just for the sake of completeness.

Implement `\scantokens`.

Implement `\everyeof`.

3.8 Section 3.8 — Breaking Paragraphs into Lines

`\lastlinefit` should be looked at in relation to other paragraph algorithm refinements. As such it may not be exactly what is needed for this particular case (but obviously TEX's algorithm is defective at this point). Implement if possible and consider replacement/extension as a later stage. If it interferes with extensions already provided it should perhaps be postponed completely.

Implement `\interlinepenalties`, `\clubpenalties`, `\widowpenalties`, and `\displaywidowpenalties`.

3.9 Section 3.9 — Math Formulas

Implement.

3.10 Section 3.10 — Hyphenation

Ω's way of doing hyphenation (via OTP) isn't sorted out this is an essential extension which would remove one of the severe restrictions of current TEX for multi-lingual hyphenation (the need for a single lc table throughout the format). Implement as a first step to remove that restriction and later replace the code using it by a new mechanism when Ω's paragraph making is revised.

3.11 Section 3.11 — Discarded Items

Implement.

3.12 Section 3.12 — Expandable Commands

Implement all, with the following exceptions:

- `\iffontchar` may not make sense in the final Ω but should probably be implemented at this point of the development.

- `\eTeXrevision` should be replaced by something suitable for Ω.

The most important of these primitives is perhaps `\protected`.

3.13 Section 4 — eTEX enhancements

Ignore.

3.14 Section 5 — Syntax extensions for eTEX

Produce syntax description of *all* Ω commands similar to this! (And more precise semantics as in this manual would be useful too, if there are any literate students to spare:-).

4 New Features — from Oldenburg discussions

In this part we discuss suggested features as they were discussed in a workshop involving the eTEX team, the LATEX3 project team and the Context team in Oldenburg, February 1998 [**?**].

Again we follow the sections from that paper. We realise that some of this may be insufficiently specified even where we have not explicitly acknowledged this.

Throughout, 'returns' means 'expands to'.

4.1 Section 3 — Expansion Control

Implement the following two commands:

`\expanded`⟨***general text***⟩ Expandable command that expands to the full expansion of the tokens in ⟨*general text*⟩: much more useful than it perhaps appears to be.

`\expandahead`⟨***number***⟩ One level expand the nth token ahead. (`\expandafter` = `\expandahead\tw@`)[4]

Ignore `\expandfromtoken` unless it is very simple to provide.

Ignore `\undef` as it is not providing anything new.

Consider implementing the following:

[4]This command was called `\expandlater` in the Oldenburg notes.

Real Register A data type that has the same behaviour/arithmetic (including expressions) as ⟨*dimen*⟩ but without the need to supply units. (This would avoid the many uses of `\strip@pt` in LaTeX, where units are added just to do arithmetic and then need to be removed.)

A problem with this naive and simple approach is what happens when casting between integer, dimension and real; so it perhaps needs further thought.

Another possibility is to add a completely separate full arithmetic set-up (including floating point) with only explicit casting to the current TeX 'arithmetic' stuff.

Ignore the suggestion for a boolean type, at least until we get a good specification for this.

The following needs much better specification (what exactly is meant?) but should probably be implemented then:

`\ifinsidebox` Similar to `\ifinner` but different. (See Hans' email).

Interpretation of the above: this should be 'true' if the list under construction is not the MVL (main vertical list) or an outer hlist directly entered from the MVL; ie it is within some box.

Within a `\setbox` declaration it could be interpreted as 'false' if not being inside an "inner box" ie not the one being assigned.

The next should be properly specified and then implemented:

Local/Global Assignments Some mechanism to specify that within the current scope global assignments within nested groups are 'global' only to the current group but no further (ie they are local to the current group). And maybe some further refinements.

During the Oldenburg meeting Rainer Schöpf made sketch which unfortunately never made it into the notes.

4.2 Section 4 — Line breaking (hmode)

Implement the following two items:

Full typography in hmode Remove all optimisations when hboxes are constructed (inner hmode) since they lead to problems when unboxing such boxes into outer hmode. Some things currently are only active in outer hmode (language nodes etc).

Discretionaries after - Explicit switch to turn off automatic insertion of discretionaries after (ligatures ending in) 'hyphenchar'.

The suggestions for "Hyphen desirability levels" needs further evaluation since in Ω in contrast to eTEX this area could be handled using concepts different from `\patterns`.

There are no doubt many other factors affecting line break choice that have not yet been considered, such as avoiding ladders of hyphens (or other confusing artefacts).

4.3 Section 5 — Page building (vmode)

Implement all with the following remarks:

Vertical discretionary This is important but needs a precise specification. In particular it means that building the page may need a somewhat different algorithm since with variants introduced one need to have a global optimising algorithm similar to the paragraph breaking algorithm rather than the simple algorithm used right now which simply finds the next champion break point (we have some thoughts on this in a separate paper).

`\forcebreakpenaltylimit` In some cases forced penalties are used to signal something to the page builder (e.g., the presence of a float or a forced column break (while collecting material for a full page)). In such cases one might reuse the collected material without having those penalties trigger the output routine again (`xor.sty` is a good example of the horrible gymnastics you have to code due to the fact that this is currently not possible.

A different semantics: it might be best to have this limit result in ignoring all penalty values smaller than the limit, so that the default would be to set it to `-\maxdimen`. Or one could think of allowing the specification of a "range" of values to be ignored.

Of course it may be better one day to have a completely different mechanism for handling galleys and pages but that is another story.

`\outputenalty`**10000** This is also related to the use of penalties as signals to the page builder. TEX's output routine handling has the deficiency that it will replace the value of the penalty node triggering the output routine by the value 10000; the result is that at this point TEX will never break again.

This has the side-effect that using penalties for signalling to the OR (e.g. to indicate a float or a marginpar) can disable that particular line as a page break thus changing the subsequent formatting choices. So it would be better if such a penalty node were completely removed as that would allow breaks at the (typical) remaining glue later on. The output routine can easily reinsert it if it is needed in some cases.

`\nthmark` **and** `\nummarks` Implement as specified.

holdinginserts No need to do something special because of compatibility reasons in Ω, so `\holdinginserts=2` could be used as an implementation.

buildpagehook This need further investigation with respect to utility and exact semantics. The idea is to give more control over when the page-builder/breaker acts, making outer more like inner for vertical mode.

4.4 Section 6 — Node handling

Would be nice if something could be implemented here, but might be difficult.

4.5 Section 7 — Parsing general texts

Ignore: OTPs (or something more powerful) are more appropriate for this.

4.6 Section 8 — Mathematics

Consider implementing all that is applicable and well-specified.

4.7 Section 9 — Inner loop (lig/kern problems)

This is relevant to the character to glyph translation process. Although the outlined algorithm may not be applicable to what Ω will eventually do in this area, the general problem that this suggested algorithm attempts to resolve is relevant and needs to be handled by Ω.

4.8 Section 10 — Reconsider reconsider paragraph

The implementation suggestion is probably not relevant to Ω, assuming that Ωeventually provides access to the character data from the typographical data structure, since that would allow the retrial of paragraph breaking if the current result seems unsatisfactory.

However, additional access to information about the typeset paragraph, such as "maximum badness" etc. seems worth having, possibly beyond what is suggested here.

4.9 Section 11 — Word grabbing

The spec is not quite clear here: do we grab "words" or rather "letter-other-space"? It seems that this could be achieved using OTPs but it may nevertheless be sensible to provide this functionality separately. We need to investigate how this is related to OTP "bubbles" as discussed at GUTenberg 2001, Metz.

4.10 Section 12 — Named reference points

Applications for this extension are probably numerous. Needs precise specification though probably anything that provides the general capability would do.[5]

4.11 Section 13 — Special specials

Supporting a non-immediate special which is, in some sense, evaluated when writing to the dvi file might be very helpful to communicate positional and other formatting information to the external rendering process. To make this happen one could, for example, use new primitives which expand to the current x- and y-position on the page or within the current box, when the box is shipped out.

4.12 Section 14 — Combining penalties

This is an Important new idea. Something much better than the current treatment should be implemented but, as mentioned, the exact algorithm isn't that important. The first stage is to decide when and how the combining algorithm acts.

4.13 Section 15 — Symbolic characters

Ignore: obsolete given Ω's internal character data model.

4.14 Section 16 — Parameter matching

Potentially a somewhat useful set of extension, though most of the programming needs are probably better served by providing additional basic data structures for lists and property lists (associative arrays, directories) and perhaps extending OTPs to some level of regexp processing.

References

[1] The eTEX Manual, Version 2, February 1998 (`etex_man.tex`)

[2] Notes on Oldenburg etex/latex3/context Meeting, February 1998. `http://www.latex-project.org/papers/etex-meeting-notes.pdf` and `http://www.latex-project.org/papers/etex-math-notes.pdf`.

[5] Chris may have more precise specs written up; this is also related to 'position nodes' as is the next subsection.