LaTeX3 News

Issue 9, March 2014 (LaTeX release 2014-03-01)

Contents

Hiatus?

Well, it's been a busy couple of years. Work has slowed on the LaTeX3 codebase as all active members of the team have been — shall we say — busily occupied with more pressing concerns in their day-to-day activities.

Nonetheless, Joseph and Bruno have continued to fine-tune the LaTeX3 kernel and add-on packages. Browsing through the commit history shows bug fixes and improvements to documentation, test files, and internal code across the entire breadth of the codebase.

Members of the team have presented at two TUG conferences since the last LaTeX3 news. (Has it really been so long?) In July 2012, Frank and Will travelled to Boston; Frank discussed the challenges faced in the past and continuing to the present day due to the limits of the various TeX engines; and, Frank and Will together covered a brief history and recent developments of the expl3 code.

In 2013, Joseph and Frank wrote a talk on complex layouts, and the "layers" ideas discussed in LaTeX3; Frank went to Tokyo in October to present the work. Slides of and recordings from these talks are available on the LaTeX3 website.

These conferences are good opportunities to introduce the expl3 language to a wider group of people; in many cases, explaining the rationale behind why expl3 looks a little strange at first helps to convince the audience that it's not so weird after all. In our experience, anyone that's been exposed to some of the more awkward expansion aspects of TeX programming appreciates how expl3 makes life much easier for us.

expl3 in the community

While things have been slightly quieter for the team, more and more people are adopting expl3 for their own use. A search on the TeX Stack Exchange website for either "`expl3`" or "`latex3`" at time of writing yield around one thousand results each.

In order to help standardise the prefixes used in expl3 modules, we have developed a registration procedure for package authors (which amounts to little more than notifying us that their package uses a specific prefix, which will often be the name of the package itself). Please contact us via the `latex-l` mailing list to register your module prefixes and package names; we ask that you avoid using package names that begin with `l3...` since expl3 packages use this internally. Some authors have started using the package prefix `lt3...` as a way of indicating their package builds on expl3 in some way but is not maintained by the LaTeX3 team.

In the prefix database at present, some thirty package prefixes are registered by fifteen separate individuals (unrelated to The LaTeX Project — the number of course grows if you include packages by members of the team). These packages cover a broad range of functionality:

acro Interface for creating (classes of) acronyms

hobby Hobby's algorithm in PGF/TiKZ for drawing optimally smooth curves.

chemmacros Typesetting in the field of chemistry.

classics Traditional-style citations for the classics.

conteq Continued (in)equalities in mathematics.

ctex A collection of macro packages and document classes for Chinese typesetting.

endiagram Draw potential energy curve diagrams.

enotez Support for end-notes.

exsheets Question sheets and exams with metadata.

lt3graph A graph data structure.

newlfm The venerable class for memos and letters.

fnpct Interaction between footnotes and punctuation.

GS1 Barcodes and so forth.

hobete Beamer theme for the Univ. of Hohenheim.

kantlipsum Generate sentences in Kant's style.

lualatex-math Extended support for mathematics in LuaLaTeX.

media9 Multimedia inclusion for Adobe Reader.

pkgloader Managing the options and loading order of other packages.

substances Lists of chemicals, etc., in a document.

withargs Ephemeral macro use.

xecjk Support for CJK documents in XeLaTeX.

xpatch, regexpatch Patch command definitions.

xpeek Commands that peek ahead in the input stream.

xpinjin Automatically add pinyin to Chinese characters

zhnumber Typeset Chinese representations of numbers

zxjatype Standards-conforming typesetting of Japanese for XeLaTeX.

Some of these packages are marked by their authors as experimental, but it is clear that these packages have been developed to solve specific needs for typesetting and document production.

The expl3 language has well and truly gained traction after many years of waiting patiently.

A logo for the LaTeX3 Programming Language

To show that expl3 is ready for general use Paulo Cereda drew up a nice logo for us, showing a hummingbird (agile and fast — but needs huge amounts of energy) picking at "l3". Big thanks to Paulo!

Recent activity

LaTeX3 work has only slowed, not ground to a halt. While changes have tended to be minor in recent times, there are a number of improvements worth discussing explicitly.

1. Bruno has extended the floating point code to cover additional functions such as inverse trigonometric functions. These additions round out the functionality well and make it viable for use in most cases needing floating point mathematics.

2. Joseph's refinement of the experimental galley code now allows separation of paragraph shapes from margins/cutouts. This still needs some testing!

3. For some time now expl3 has provided "native" drivers although they have not been selected by default in most cases. These have been revised to improve robustness, which makes them probably ready to enable by default. The improvements made to the drivers have also fed back to more "general" LaTeX code.

Work in progress

We're still actively discussing a variety of areas to tackle next. We are aware of various "odds and ends" in expl3 that still need sorting out. In particular, some experimental functions have been working quite well and it's time to assess moving them into the "stable" modules, in particular the l3str module for dealing with catcode-twelve token lists more commonly known in expl3 as *strings*.

Areas of active discussion including issues around uppercasing and lowercasing (and the esoteric ways that this can be achieved in TeX) and space skipping (or not) in commands and environments with optional arguments. These two issues are discussed next.

Uppercasing and lowercasing

The commands `\tl_to_lowercase:n` and `\tl_to_uppercase:n` have long been overdue for a good hard look. From a traditional TeX viewpoint, these commands are simply the primitive `\lowercase` and `\uppercase`, and in practice it's well known that there are various limitations and peculiarities associated with them. We know these commands are good, to one extent or another, for three things:

1. Uppercasing text for typesetting purposes such as all-uppercase titles.

2. Lowercasing text for normalisation in sorting and other applications such as filename comparisons.

3. Achieving special effects, in concert with manipulating `\uccode` and the like, such as defining commands that contain characters with different catcodes than usual.

We are working on providing a set of commands to achieve all three of these functions in a more direct and easy-to-use fashion, including support for Unicode in LuaLaTeX and XeLaTeX.

Space-skipping in xparse

We have also re-considered the behaviour of space-skipping in xparse. Consider the following examples:

```
\begin{dmath}          \begin{dmath}[label=foo]
[x y z] = [1 2 3]      x^2 + y^2 = z^2
\end{dmath}            \end{dmath}
```

In the first case, we are typesetting some mathematics that contains square brackets. In the second, we are assigning a label to the equation using an optional argument, which also uses brackets. The fact that both work correctly is due to behaviour that is specifically programmed into the workings of the `dmath` environment of breqn: spaces before an optional argument are explicitly forbidden. At present, this is also how commands and environments defined using xparse behave. But consider a pgfplots environment:

```
\begin{pgfplot}
  [
    % plot options
  ]
  \begin{axis}
    [
      % axis options
    ]
    ...
  \end{axis}
\end{pgfplot}
```

This would seem like quite a natural way to write such environments, but with the current state of xparse this syntax would be incorrect. One would have to write either of these instead:

```
\begin{pgfplot}%          \begin{pgfplot}[
  [                          % plot options
   % plot options           ]
  ]
```

Is this an acceptable compromise? We're not entirely sure here — we're in a corner because the humble `[` has ended up being part of both the syntax and semantics of a LaTeX document.

Despite the current design covering most regular use-cases, we have considered adding a further option to xparse to define the space-skipping behaviour as desired by a package author. But at this very moment we've rejected adding this additional complexity, because environments that change their parsing behaviour based on their intended use make a LaTeX-based language more difficult to predict; one could imagine such behaviour causing difficulties down the road for automatic syntax checkers and so forth. However, we don't make such decisions in a vacuum and we're always happy to continue to discuss such matters.

. . . and for 2014 onwards

There is one (understandable) misconception that shows up once in a while with people claiming that

$$\text{expl3} = \text{LaTeX3}.$$

However, the correct relation would be a subset,

$$\text{expl3} \subset \text{LaTeX3},$$

with expl3 forming the Core Language Layer on which there will eventually be several other layers on top that provide

- higher-level concepts for typesetting (Typesetting Foundation Layer),
- a designer interface for specifying document structures and layouts (Designer Layer),
- and finally a Document Representation Layer that implements document level syntax.

Of those four layers, the lowest one — expl3 — is available for use and with xparse we have an instance of the Document Representation Layer modeled largely after LaTeX 2$_\varepsilon$ syntax (there could be others). Both can be successfully used within the current LaTeX 2$_\varepsilon$ framework and as mentioned above this is increasingly happening.

The middle layers, however, where the rubber meets the road, are still at the level of prototypes and ideas (templates, ldb, galley, xor and all the good stuff) that need to be revised and further developed to arrive at a LaTeX3 environment that can stand on its own and that is to where we want to return in 2014.

An overview on this can be found in the answer to "What can *I* do to help The LaTeX Project?" on Stack Exchange,[1] which is reproduced below in slightly abridged form. This is of course not the first time that we have discussed such matters, and you can find similar material in other publications such as those at `http://latex-project.org`; e.g., the architecture talk given at the TUG 2011 conference.

[1] `http://tex.stackexchange.com/questions/45838`

What can you do for The LaTeX Project?

By Frank Mittelbach

My vision of LaTeX3 is really a system with multiple layers that provide interfaces for different kinds of roles. These layers are

- the underlying engine (some TeX variant)
- the programming layer (the core language, i.e., expl3)
- the typesetting foundation layer (providing higher-level concepts for typesetting)
- the typesetting element layer (templates for all types of document elements)
- the designer interface foundation layer
- the class designer layer (where instances of document elements with specific settings are defined)
- document representation layer (that provides the input syntax, i.e., how the author uses elements)

If you look at it from the perspective of user roles then there are at least three or four roles that you can clearly distinguish:

- The Programmer (template and functionality provider)
- The Document Type Designer (defines which elements are available; abstract syntax and semantics)
- The Designer (typography and layout)
- The Author (content)

As a consequence The LaTeX Project needs different kinds of help depending on what layer or role we are looking at.

The "Author" is using, say, list structures by specifying something like `\begin{itemize} \item` in his documents. Or perhaps by writing `<ul> ... </ul>` or whatever the UI representation offers to him.

The "Document Type Designer" defines what kind of abstract document elements are available, and what attributes or arguments those elements provide at the author level. E.g., he may specify that a certain class of documents provides the display lists "enumerate", "itemize" and "description".

The "Programmer" on the other hand implements templates (that offer customizations) for such document elements, e.g., for display lists. What kind of customization possibilities should be provided by the "Programmer" is the domain of the "Document Designer"; he drives what kind of flexibility he needs for the design. In most cases the "Document Designer" should be able to simply select templates (already written) from a template library and only focus on the design, i.e., instantiating the templates with values so that the desired layout for "itemize" lists, etc., is created.

In real life a single person may end up playing more than one role, but it is important to recognise that all of them come with different requirements with respect to interfaces and functionality.

Programming Layer

The programming layer consists of a core language layer (called expl3 (EXP erimental L aTeX 3) for historical reasons and now we are stuck with it `:-)`) and two more components: the "Typesetting Foundation Layer" that we are currently working on and the "Typesetting Element Layer" that is going to provide customizable objects for the design layer. While expl3 is in many parts already fairly complete and usable the other two are under construction.

Help is needed for the programming layer in

- helping by extending and completing the regression test suite for expl3
- helping with providing good or better documentation, including tutorials
- possibly helping in coding additional core functionality — but that requires, in contrast to the first two points, a good amount of commitment and experience with the core language as otherwise the danger is too high that the final results will end up being inconsistent

Once we are a bit further along with the "Typesetting Foundation Layer" we would need help in providing higher-level functionality, perhaps rewriting existing packages/code for elements making use of extended possibilities. Two steps down the road (once the "Designer Layer" is closer to being finalized) we would need help with developing templates for all kinds of elements.

In summary for this part, we need help from people interested in programming in TeX and expl3 and/or interested in providing documentation (but for this a thorough understanding of the programming concepts is necessary too).

Design Layer

The intention of the design layer is to provide interfaces that allow specifying layout and typography styles in a declarative way. On the implementation side there are a number of prototypes (most notably xtemplate and the recent reimplementation of ldb). These need to be unified into a common model which requires some more experimentation and probably also some further thoughts.

But the real importance of this layer is not the implementation of its interfaces but the conceptual view

of it: provisioning a rich declarative method (or methods) to describe design and layout. I.e., enabling a designer to think not in programs but in visual representations and relationships.

So here is the big area where people who do not feel they can or want to program TEX's bowels can help. What would be extremely helpful (and in fact not just for LATEX3) would be

- collecting and classifying a *huge* set of layouts and designs
  - designs for individual document elements (such as headings, TOCs, etc)
  - document designs that include relationships between document elements
- thinking about good, declarative ways to specify such designs
  - what needs to be specified
  - to what extent and with what flexibility

I believe that this is a huge task (but rewarding in itself) and already the first part of collecting existing design specifications will be a major undertaking and will need coordination and probably a lot of work. But it will be a huge asset towards testing any implementations and interfaces for this layer later on.

Document Interface Layer

If we get the separation done correctly, then this layer should effectively offer nothing more than a front end for parsing the document syntax and transforming it into an internal standardised form. This means that on this layer one should not see any (or not much) coding or computation.

It is envisioned that alternative document syntax models can be provided. At the moment we have a draft solution in xparse. This package offers a document syntax in the style of LATEX 2_ε, that is with *-forms, optional arguments in brackets, etc., but with a few more bells and whistles such as a more generalized concept of default values, support for additional delimiters for arguments, verbatim-style arguments, and so on. It is fairly conventional though. In addition when it was written the clear separation of layers wasn't well-defined and so the package also contains components for conditional programming that I no longer think should be there.

Bottom line on what is needed for this layer is to

- think about good syntax for providing document content from "the author" perspective
- think about good syntax for providing document content from an "application to typesetting" perspective, i.e., the syntax and structure for automated typesetting where the content is prepared by a system/application rather than by a human

These two areas most likely need strict structure (as automation works much better with structures that do not have a lot of alternative possibilities and shortcuts, etc.) and even when just looking at the human author a lot of open questions need answering. And these answers may or may not be to painfully stick with existing LATEX 2_ε conventions in all cases (or perhaps with any?).

None of this requires coding or expl3 experience. What it requires is familiarity with existing input concepts, a feel for where the pain points currently are and the willingness to think and discuss what alternatives and extensions could look like.

In Summary

Basically help is possible on any level and it doesn't need to involve programming. Thoughts are sprinkled throughout this article, but here are a few more highlights:

- help with developing/improving the core programming layer by
  - joining the effort to improve the test suite
  - help improving the existing (or not existing) documentation
  - joining the effort to produce core or auxiliary code modules
- help on the design layer by
  - collecting and classifying design tasks
  - thinking and suggesting ways to describe layout requirements in a declarative manner
- help on shaping the document interface layer

These concepts, as well as their implementation, are under discussion on the list `latex-l`.[2] The list has only a fairly low level of traffic right now as actual implementation and development tasks are typically discussed directly among the few active implementers. But this might change if more people join.

And something else . . .

The people on the LATEX3 team are also committed to keeping LATEX 2_ε stable and even while there isn't that much to do these days there remains the need to resolve bug reports (if they concern the 2e core), provide new distributions once in a while, etc. All this is work that takes effort or remains undone or incomplete. Thus here too, it helps the LATEX3 efforts if we get help to free up resources.

[2] Instructions for joining and browsing archives at: `http://latex-project.org/code.html`