LaTeX3 News

Issue 8, July 2012

Extended floating point support

Bruno Le Floch has been re-writing the floating point module to function in an 'expandable' manner. This allows floating point calculations to be computed far more flexibly and efficiently than before. The expandable nature of the new code allows its use inside operations such as writing to external files, for which previously any such calculations would have to be precalculated before any of the writing operations began. Bruno's work on the floating point module has been officially released into the main SVN repository for `l3kernel`; TeX Live 2012 will contain the 'old' code for stability while this year is spent testing the new code in production environments and ironing out any wrinkles. Here's a neat example as suggested in the documentation, which produces '$6.278\,400\,000\,000\,000 \times 10^2$':

```
\usepackage{xparse, siunitx}
\ExplSyntaxOn
\NewDocumentCommand { \calcnum } { m }
  { \num { \fp_to_scientific:n {#1} } }
\ExplSyntaxOff

\calcnum {
  round ( 200 pi * sin ( 2.3 ^ 5 ) , 2 )
}
```

This feature is invaluable for simple (and not-so-simple) calculations in document and package authoring, and has been lacking a robust solution for many years. While LuaLaTeX can perform similar tasks within its Lua environment, the floating point support is written using the `expl3` programming language only, and is thus available in pdfLaTeX and XeLaTeX as well.

Regular expressions in TeX

As if expandable floating point support wasn't enough, Bruno has also written a complete regular expression engine in `expl3` as well. Many reading this will be familiar with the quote attributed to Jamie Zawinski:

> *Some people, when confronted with a problem, think "I know, I'll use regular expressions." Now they have two problems.*

And as humorous as the saying is, it's still fair to say that regular expressions are a great tool for manipulating streams of text. We desperately hope that people will *not* start using the regex code to do things like parse XML documents; however, for general search–replace duties, there's never been anything like `l3regex` for the LaTeX world. As a trivial example, there are 23 instances of the word 'We' or 'we' in this document (including those two). This value is counted automatically in two lines of code.

And again, it is available for pdfLaTeX and XeLaTeX users as well as LuaLaTeX ones; it also bears noting that this provides an easy solution for applying regular expression processing to LaTeX documents and text data even on the Windows operating system that does not have native support for such things.

Separating internal and external code

LaTeX packages are written by a wide range of package authors and consist of code and commands for various purposes. Some of these commands will be intended for use by document authors (such as `\pbox` from the `pbox` package); others are intended for use by other package writers (such as `\@ifmtarg` from the `ifmtarg` package). However, a large portion of them are internal, i.e., are intended to be used only within the package or within the LaTeX kernel and should not be used elsewhere. For example, `\@float` is the LaTeX kernel interface for floats to be used in class files, but the actual work is done by a command called `\@xfloat` which should not be used directly. Unfortunately the LaTeX 2_ε language makes no clear distinction between the two, so it is tempting for programmers to directly call the latter to save some "unnecessary" parsing activity.

The downside of this is that the "internal" commands suddenly act as interfaces and a reimplementation or fix in any of them would then break add-on packages as they rely on internal behavior. Over the course of the last twenty years probably 80% of such "internal" commands have found their way into one or another package. The consequences of this is that nowadays it is next to impossible to change anything in the LaTeX 2_ε kernel (even if it is clearly just an internal command) without breaking something.

In LaTeX3 we hope to improve this situation drastically by clearly separating public interfaces (that extension packages can use and rely on) and private functions and variables (that should not appear outside of their module). There is (nearly) no way to enforce this without severe computing overhead, so we implement it only through a naming convention, and some support mechanisms. However, we think that this naming convention is easy to understand and to follow, so that we are

confident that this will be adopted and provides the desired results.

Naming convention for internals

We've been throwing around some ideas for this for a number of years but nothing quite fit; the issue comes down to the fact that TEX does not have a 'name-spacing' mechanism so any internal command needs to have a specific prefix to avoid clashing with other packages' commands. The prefix we have finally decided on for expl3 code is a double underscore, such that functions like `\seq_count:N` are intended for external use and `\__seq_item:n` is an internal command that should not be used or relied upon by others.

All this is well and good, but it can be inconvenient to type long prefixes such as `\__seq_` before all command names, especially in a package for which nearly *all* package functions are internal.

We therefore also extended DocStrip slightly by adding a 'shorthand' for internal package prefixes. Commands and variables in `.dtx` code may now contain `@@` which is expanded to the function prefix when the `.sty` file is extracted. As an example, writing

```
%<@@=seq>
\cs_new:Npn \@@_item:n
  ...
```

is equivalent to

```
\cs_new:Npn \__seq_item:n
  ...
```

There are clear advantages to this syntax. Function names are shorter and therefore easier to type, and code can still be prototyped using the `@@` syntax (e.g., pasting code between a `.dtx` file and a regular `.tex` document). Most importantly, it is explicitly clear from the code source which commands are intended to be used externally and which should be avoided.

We hope that this syntax will prove popular; in our initial experiments we think it works very well. In fact we found a good number of smaller errors when being forced to think about what is internal and what is an external function.

Continual revolution—the 'small bang'

In addition to the major additions introduced above, Frank Mittelbach has been examining `expl3` with a fresh eye to resolve any outstanding issues in the consistency or logic of the names of functions.

We are very mindful of the fact that for people to find `expl3` a useful tool, it must have a stable interface. This said, there are still some musty corners that we can show where people simply haven't been using certain functions. In select cases, we're re-assessing whether all of the (sometimes esoteric) odds and ends that have been added to `expl3` really belong; in other cases, it's now clear that some naming or behaviour choices weren't correct the first time around.

To address this tarnish, we're carefully making some minor changes to parts of the `expl3` interface and we'd like to allay any fears that `expl3` isn't stable. The `expl3` language now offers a wide range of functions plus their variants, and we're talking about changing but a very small percentage of these, and not common ones at that. We don't want it to become a mess, so we think it's better to tidy things up as we go. Follow the `LaTeX-L` mailing list for such details as they arise.