LaTeX News

Issue 31, February 2020

Contents

Introduction

This document is under construction ...

Experiences with the LaTeX-dev formats

As reported in the previous *LaTeX News*, we have made a pre-release version of the LaTeX kernel available as `LaTeX-dev`. Overall, the approach of having an explicit testing release has been positive: it is now readily-available in TeX systems and is getting real use beyond the team.

The current release has been tested by a number of users, and we have had useful feedback on a range of new ideas. This has allowed us to fix issues in several of the new features described below. We thank the dedicated users who have been trying out the development formats, and encourage others to do so. Pre-testing in this way does mean that for the vast majority of users, problems are solved before they even appear!

Concerning this pre-release ...

In TeXLive 2020 the LuaLaTeX format will use the new LuaHBTeX engine, which is LuaTeX with an embedded HarfBuzz library. HarfBuzz can be used by setting a suitable renderer in the font declaration. A basic interface for that is provided by fontspec. This additional font renderer will greatly improve the shaping of various scripts, which are currently handled correctly only by XeTeX. To simplify the testing of the new engine, binaries have been already added to MiKTeX and TeXLive 2019 and both have changed the LuaLaTeX-dev format to use it.

Improved load-times for expl3

The LaTeX3 programming layer, expl3, has over the past decade moved from being largely experimental to broadly stable. It is now used in a significant number of third-party packages, most notably xparse for defining interfaces in cases where no expl3 code is "visible". In addition, most LaTeX documents compiled using XeTeX or LuaTeX load fontspec, which is written using expl3.

The expl3 layer contains a non-trivial number of macros, and when used with the XeTeX and LuaTeX engines, it loads a large body of Unicode data. This means that even on a fast computer, there is a relatively large load time for using expl3.

For this release, the team have made adjustments in the LaTeX 2_ε kernel to pre-load a significant portion of expl3 as the format is built. This is transparent at the user level, other than the significant decrease in document processing time: there will be no "pause"

for loading Unicode data files. Loading of expl3 in documents and packages can be done as usual; eventually, it will be possible to omit

```
\RequirePackage{expl3}
```

entirely, but to support older formats, this is still recommended at present.

Improvements to LaTeX's font selection mechanism (NFSS)

Extending the shape management of NFSS

Over time more and more fonts have become available for use with LaTeX. Many such font families offer additional shapes, e.g., small caps italics (`scit`), small caps slanted (`scsl` or swash letters (`sw`). By using `\fontshape` those shapes can be explicitly selected and for the swash letter shapes there is also `\swshape` and `\textsw` available.

In the original font selection implementation a request to select a new shape always overrode the current shape. With the 2020 release of LaTeX this has changed and `\fontshape` can now be used to combine small capitals with italics, slanted or swash letters, either by explicitly asking for `scit`, etc., or by asking for italics when typesetting already in small caps and so forth.

Using `\upshape` will still change italics or slanted back to an upright shape but will not any longer alter the small caps setting. To change small capitals back to upper/lower case you can now use `\ulcshape` (or `\textulc`) which in turn will not change the font with respect to italics, slanted or swash. There is one exception: for compatibility reasons `\upshape` will change small capitals back to upright (`n` shape), if the current shape is `sc`. This is done so that something like `\scshape...\upshape` continues to work, but we suggest that you don't use that deprecated method in new documents.

Finally, if you want to reset the shape back to normal you can use `\normalshape` which is a shorthand for `\upshape\ulcshape`.

The way that shapes combine with each other is not hardwired but is customizable and extensible if there is ever a need for it. The mappings are defined through `\DeclareFontShapeChangeRule` and the details for developers are documented in `source2e.pdf`.

The ideas for this interface extension have been pioneered in fontspec by Will Robertson for Unicode engines and in fontaxes by Andreas Bühmann and Michael Ummels for pdfTeX and used in many font support packages.

Extending the series management of NFSS

Many of the the newer font families also come provided with additional weights (thin, semi-bold, ultra-bold, etc.) or several running lengths such a condensed or extra-condensed. In some cases the number of different series values is really impressive, for example, Noto Sans offers 36 fonts from ultra-light extra condensed to ultra-bold medium width.

Already in its original design, NFSS supported 9 weight levels from ultra-light (`ul`) to ultra-bold (`ub`) and also 9 width levels from ultra-condensed (`uc`) to ultra-expanded (`ux`) so more than enough even for a font family like Noto Sans. Unfortunately, some font support packages nevertheless invented their own names so in recent years you could find all kinds of non-standard series names like `k`, `i`, `j` and others making it impossible to combine different fonts successfully using the standard NFSS mechanisms.

Over the course of the last year a small number of individuals, notably, Bob Tennent, Michael Sharpe and Marc Penninga worked hard to bring this unsatisfying situation back under control and today we are happy to report that the internal font support files for more than a hundred font families are back to following the standard NFSS conventions so that combining them is now again rather nice and easy (of course, there is still the task of choosing combinations that visually work well together, but from a technical perspective they can now easily matched).

In the original font selection implementation, a request to select a new series always overrode the current one. This was reasonable because there were nearly no fonts available that offered anything other than a medium or a bold series. Now that this has changed and families such as Noto Sans are available, combining weight and width into a single attribute is no longer appropriate. With the 2020 release of LaTeX the series management therefore changed to allow for independently setting the weight and the width attribute of the series.

For most users this change will be largely transparent as LaTeX offers only `\textbf` or `\bfseries` to select a bolder face (and `\textmd` and `\mdseries` to return to a medium series) but no high-level command for selecting a condensed face, etc. However, with the NFSS low-level interface, it is now possible to ask for, say, `\fontseries{c}\selectfont` in a marginal note to get a condensed face and that would still allow using `\textbf` inside. This then would select a bold condensed face and not a rather odd-looking bold-extended face in the middle of condensed type.

The expectation is that this functionality is largely used by class and package designers, but given that the low-level NFSS commands are usable on the document level and not really difficult to apply, there are probably also a number of users who will enjoy using the new possibilities that bring LaTeX back into the front league when it comes to font usage.

The way different series values combine with each other is not hardwired but is again customizable

and extensible. The mappings are defined through `\DeclareFontSeriesChangeRule` and the details for developers are documented in `source2e.pdf`.

Font series defaults per document family

With additional weights and widths available in many font families it becomes more likely that somebody wants to match, say, a medium weight serif family with a semi-light sans serif family or that with one family one wants to use the bold-extend face when `\textbf` is used while with another it should be bold (not extended) or semibold, etc.

In the past this kind of extension was made available with the mweights package by Bob Tennent which has been used in many font support packages.

With the 2020 release of LaTeX this feature is now available out of the box. In addition we also offer a document-level interface to adjust the behavior of the high-level series commands `\textbf`, `\textmd` and their declaration forms `\bfseries` and `\mdseries` so that they can have different effects for the serif, sans serif and typewriter families used in a document.

For example, specifying

```
\DeclareFontSeriesDefault[rm]{bf}{sb}
\DeclareFontSeriesDefault[tt]{md}{lc}
```

in the document preamble would result in `\textbf` producing semi-bold (`sb`) when typesetting in roman typeface and that typewriter is by default (medium series `md`) using a light-condensed face. The optional argument here can be either `rm`, `sf` or `tt` to indicate one of the three main font families in a document; if omitted you will change the overall document default instead. In the first mandatory argument you specify either `md` or `bf` and the second mandatory argument then gives the desired series value in NFSS nomenclature.

Emphasis handling generalized

With previous releases of LaTeX nested `\emph` commands automatically alternated between italics and upright. This mechanism has now been generalized and you can now specify for arbitrary nesting levels how emphasis should be handled.

The declaration `\DeclareEmphSequence` expects a comma separated list of font declarations corresponding to increasing levels of emphasis. For example,

```
\DeclareEmphSequence{\itshape,%
                     \upshape\scshape,\itshape}
```

uses italics for the first, small capitals for the second, and italic small capitals for the third level (provided you use a font that supports these shapes). If there are a more nesting levels than provided, LaTeX uses the declarations stored in `\emreset` (by default `\ulcshape\upshape`) for the next level and then restarts the list.

The mechanism tries to be "smart" and verifies that the given declarations actually alter the current font. If not it continues and tries the next level—the assumption being that there was already a manual font change in the document to the font that is now supposed to be used for emphasis. Of course, this only works if the declarations in the list entries actually change the font and not, for example, just the color. In such a scenario one has to add `\emforce` to the entry which directs the mechanism to use the level, even if the font attributes appear to be unchanged.

Providing font family substitutions

Given that pdfTeX can only handle fonts with up to 256 glyphs a single font encoding can only support a few languages. The `T1` encoding, for example, does support many of the Latin based scripts, but if you want to write in Greek or Russian you need to switch encodings to `LGR` or `T2A`. Given that not every font family offers glyphs in such encodings, you may end up with some default family (e.g., Computer Modern) that doesn't blend in well chosen document font. For such cases NFSS now offers `\DeclareFontFamilySubstitution`, for example:

```
\DeclareFontFamilySubstitution{LGR}
     {Montserrat-LF}{IBMPlexSans-TLF}
```

tells LaTeX that if you are typesetting in the sans serif font `Montserrat-LF` and the Greek encoding `LGR` is asked for, then LaTeX should use `IBMPlexSans-TLF` to fulfill the encoding request.

The code is based on ideas from the substitutefont package by Günter Milde, but implemented differently.

Providing all text companion symbols by default

The text companion encoding `TS1` was originally not available by default, but only when the textcomp package was loaded. The main reason for this was limited availability in fonts other than Computer Modern and memory restrictions back in the nineties. These days neither limitation exists any more so with the 2020 release all the symbols provided with the textcomp package are available out of the box.

Furthermore, an intelligent substitution mechanism has been implemented so that missing glyphs are automatically substituted with defaults that are sans serif if you typeset in `\textsf` and monospaced if you typeset using `\texttt` and not always serifed.

This is most noticeable with `\oldstylenums` which are now taken from `TS1` so that you no longer get 123 but 123 when typesetting in sans serif fonts `and 123 when using typewriter fonts.`

If there ever is a need to use the original (inferior) definition, then that remains available as `\legacyoldstylenums` and to fully revert to the old behavior there is also `\UseLegacyTextSymbols`. That declaration reverts `\oldstylenums` and also

changes the footnote symbols, such as `\textdagger`, `\textparagraph`, etc. pick up their glyphs again from the math fonts instead of the current text font (this means they always keep the same shape and do not nicely blend in with the text font).

With the text companion symbols as part of the kernel it is normally no longer necessary to load the `textcomp` package, but for backwards compatibility this package will remain available. There is, however, one use case where it remains useful: if you load the package with the option `error` or `warn` then substitutions will change their behavior and result in a LaTeX error or a LaTeX warning (on the terminal), respectively. Without the package the substitution information only appears in the `.log` file. If you use the option `quit`, then even the information in the transcript is suppressed (which is not really recommended).

Other changes to the LaTeX kernel

New `alias` size function for use in `.fd` files
Most of the newer fonts supported in TeX have been set up with the `autoinst` tool by Marc Penninga. In the past this program did set up the font faces using the face names chosen by its designer, e.g., "`regular`", "`bold`", etc., and then mapped those via substitution to the standard NFSS shape names, i.e., "`m`" or "`b`". As a result one got unnecessary substitution warnings such as "`Font T1/abc/bold/n not found using T1/abc/b/n instead`".

We now provide a new NFSS `alias` size function that can and will be used by `autoinst` in the future. It provides the same functionality as the `subst` function but is less vocal about its actions, such that only relevant font substitutions show up as warnings.

UTF-8 characters in package descriptions
In 2018 we made UTF-8 the default input encoding for LaTeX but we overlooked the case of package descriptions in declarations such as `\ProvidesPackage` which worked (sometimes) before but now died always. This has been corrected. *(github issue 52)*

Suppress unnecessary font substitution warnings
Many sans serif fonts do not have real italics but usually only oblique/slanted shapes, so the substitution of slanted for italics is natural and in fact many designers talk about italic sans serif faces even if in reality they are oblique. With nearly all sans serif font family the LaTeX support files therefore silently substitute slanted if you ask for `\itshape` or `\textit`. This is also true for Computer Modern in `T1` encoding but in `OT1` you got a warning on the terminal even though there is nothing you can do about it. This has now been changed to an information message only written to the `.log` file. *(github issue 172)*

Fix inconsistent hook setting when loading packages
When a package is loaded `\`*package*`.sty-h@@k` is set, but it was loaded several times it was unset again. Relevant only to package developers. *(github issue 198)*

Avoid spurious warning if `LY1` is made the default encoding
Making `LY1` the default encoding as done by some font support packages gave a spurious warning even if `\rmdefault` was changed first. This was corrected. *(github issue 199)*

Ensure that `\\` remains robust
In the last release we made most document-level commands robust, but `\\` became fragile again if `\raggedright` typesetting was used. *(github issue 203)*

Allow more write streams with `filecontents` in LuaTeX
Most TeX engines only support a maximum of sixteen concurrently open write streams and if those have been used up, then `filecontents` or any other code trying to open another one will fail. In LuaTeX more write streams are available and those can now be utilized as well. *(github issue 238)*

Changes to packages in the graphics category

Make color & graphics user-level commands robust
Some of the user-level commands of color, graphics and graphicx such as `\textcolor` or `\includegraphics` were still fragile, so didn't work in moving arguments without extra protection. All of them have now been made robust. *(github issue 208)*

Changes to packages in the tools category

Fixed column depth in boxed `multicols`
The `multicols` environment was setting `\maxdepth` when splitting boxes but the way the internal interfaces of LaTeX are designed it should have used `\@maxdepth` instead. As a result balanced boxed multicols sometimes ended up having different height even if they had exactly the same content. *(github issue 190)*

Ensure that `multicols` is not losing text
The `multicols` environment needs a set of consecutive boxes to collect column material. The way those got allocated could result in disaster if other packages allocated most boxes below box 255 (which TeX always uses for the output page). In the original implementation that problem was identified because one could only allocate boxes below 255, but nowadays the LaTeX allocation routine allows allocating boxes below and above 255. So the assumption that asking for, say 20 boxes you get a consecutive sequence of box registers was no longer true and so some of the column material could end in box 255 and be overwritten. This has now been corrected by allocating all necessary boxes

above 255 if there aren't enough registers available. *(github issue 237)*

Allow spaces in \hhline arguments
The verb `\hhline` command which allows specification of rule segments in `tabular` environments now allows (and ignores) spaces between its tokens so `\hhline{: = : =}` is now allowed and equivalent to `\hhline{:=:=}`. This matches similar token arguments in LaTeX such as the `[h t p]` argument on floats. A similar change has been made to the extended `\hhline` command in the `colortbl` package. *(github issue 242)*

Primitive requirements

Since the finalization of ε-TeX in 1999, a number of additional 'utility' primitives have been added to pdfTeX. Several of these are broadly useful and have been requirements for expl3 for some time, most notably `\pdfstrcmp`. Over time, a common set of these 'post-ε-TeX' primitives have been incorporated into XeTeX and (u)p-TeX; they were available in LuaTeX already.

A number of the additional primitives are needed to support new or improved functionality in LaTeX. This is seen for example in improved UTF-8 handling, which uses `\ifincsname`. The following primitive functionality (which in LuaTeX may be achieved using Lua code) will therefore be *required* by the LaTeX kernel from the start of 2021:

- `\expanded`
- `\ifincsname`
- `\ifpdfprimitive`
- `\pdfcreationdate`
- `\pdfelapsedtime`
- `\pdffiledump`
- `\pdffilemoddate`
- `\pdffilesize`
- `\pdflastxpos`
- `\pdflastypos`
- `\pdfmdfivesum`
- `\pdfnormaldeviate`
- `\pdfpageheight`
- `\pdfpagewidth`
- `\pdfprimitive`
- `\pdfrandomseed`
- `\pdfresettimer`
- `\pdfsavepos`
- `\pdfsetrandomseed`
- `\pdfshellescape`
- `\pdfstrcmp`
- `\pdfuniformdeviate`

For ease of reference, these primitives will be referred to as the 'pdfTeX utilities'. With the exception of `\expanded`, these have been present in pdfTeX since the release of version 1.40.0 in 2007; `\expanded` was added for TeX Live 2019. Similarly, the full set of utility primitives have been available in XeTeX from the 2019 TeX Live release, and have always been available in LuaTeX (some by Lua emulation). pTeX and upTeX gained all of the above bar `\ifincsname` for TeX Live 2019, and will have that primitive from the 2020 release.

At the same time, engines which are fully Unicode-capable must all provide the following primitives

- `\Uchar`
- `\Ucharcat`
- `\Umathcode`

Note that it has become standard practice to check for Unicode-aware engines with the existence of the `\Umathcode` primitive. As such, this is already a requirement: engines lacking these primitives cannot access Unicode features of the LaTeX 2_ε kernel or of expl3. Note that upTeX has facilities for handling Unicode but is not classed as a Unicode engine by the base LaTeX code.

References

[1] Frank Mittelbach: *The LaTeX release workflow and the LaTeX dev formats*. In: TUGboat, 40#2, 2019. `https://latex-project.org/publications/`

[2] LaTeX Project Team: *LaTeX 2_ε font selection*. `https://latex-project.org/documentation/`

[3] *LaTeX documentation on the LaTeX Project Website*. `https://latex-project.org/documentation/`