LaTeX News

Issue 36, November 2022 (LaTeX release 2022-11-01)

Contents

Introduction 1

Auto-detecting key/value arguments 1

A note for font package developers 1
- Encoding subsets for TS1 encoded fonts 1

New or improved commands 2
- Better language handling for case-changing commands 2

Code improvements 2
- Support for slanted small caps in the EC fonts 2
- EC sans serif at small sizes 2
- Improve font series handling with incorrect .fd files 2
- Detect nested minipage environments 2
- Robust commands in package options 2
- Improve l3docstrip integration into docstrip . . 2
- LuaTeX callback efficiency improvement 3
- Rule-based ordering for LuaTeX callback handlers 3

Bug fixes 3
- Prevent TeX from losing a \smash 3
- Resolve an issue with \mathchoice and localalphabets 3
- Reporting of unused global options when using key/value processing 3

Changes to packages in the graphics category 3
- Fix a \mathcolor bug 3

Changes to packages in the tools category 3
- array: Correctly identify single-line m-cells . . . 3

Introduction

The 2022-11 release of LaTeX is largely a consolidation release where we made a number of minor improvements to fix some bugs or improve one or the other interface.

The only really important functionality that was added is described in the next section: the ability to easily define document-level commands and environments that accept a key/value list in one of its (usually optional) arguments, including the ability to determine if the argument does in fact contain such a key/value list or just a single "classical" value.

For the "Tagged LaTeX Project" this functionality is very important because many document-level commands will need to accept such key/value lists, for example, to specify alternative text or overwrite default tagging if that becomes necessary in a document.

Auto-detecting key/value arguments

To allow extension of the core LaTeX syntax, ltcmd now supports a `=...` modifier when grabbing arguments. This modifier instructs LaTeX that the argument should be passed to the underlying code as a set of key/values. If the argument does not "look like" a set of key/values, it will be converted into a single key/value pair, with the argument to `=` specifying the name of that key. For example, the `\caption` command could be defined as

```
\DeclareDocumentCommand\caption
       {s ={short-text}+O{#3} +m}
       {...}
```

which would mean that if the optional argument does *not* contain key/value data, it will be converted to a single key/value pair with the key name `short-text`.

Arguments which begin with `=,` are always interpreted as key/values even if they do not contain further `=` signs. Any `=` signs enclosed within `$...$` or `\(...\)`, i.e. in inline math mode, are ignored, meaning that only `=` outside of math mode will generally cause interpretation as key/value material.

In case the argument contains a "textual" `=` sign that is mistaken as a key/value indicator you can hide it using a brace group as you would do in other places, e.g.,

```
\caption[{Use of = signs}]
        {Use of = signs in optional arguments}
```

However, because `=` signs in math mode are already ignored, this should seldom be necessary.

A note for font package developers

Encoding subsets for TS1 encoded fonts

The text companion encoding TS1 is unfortunately not very faithfully supported in fonts that are not close cousins to the Computer Modern fonts. It was therefore necessary to provide the notion of "sub-encodings" on a per font basis. These sub-encodings are declared for a

LaTeX News, and the LaTeX software, are brought to you by the LaTeX Project Team; Copyright 2022, license LPPL.

font family with the help of a `\DeclareEncodingSubset` declaration, see [5] for details.

Maintainers of font bundles that include `TS1` encoded font files should add an appropriate declaration into the corresponding `ts1`*family*`.fd` file, because otherwise the default subencoding is assumed, which is probably disabling too many glyphs that are actually available in the font.[1] *(github issue 905)*

New or improved commands

Better language handling for case-changing commands

The commands `\MakeUppercase`, `\MakeLowercase` and `\MakeTitlecase` now automatically detect the locale currently in use when `babel` is loaded. This allows automatic adjustment of letter mappings where appropriate. They also accept a leading optional argument. This accepts a key–value list of control settings. At present, there is one key available: `locale`, which can also be accessed via the alias `lang`. This is intended to allow local setting of the language, which can be done using a BCP-47 descriptor. For example, this could be used to force Turkish case changing in otherwise English input

```
\MakeUppercase[lang = tr]{Ragıp Hulûsi Özdem}
```

yields RAGIP HULÛSİ ÖZDEM.

Code improvements

Support for slanted small caps in the EC fonts

For some time LaTeX has supported the combination of the shapes small caps and italic/slanted. The EC fonts contain slanted small caps fonts but using them required the loading of an external package. Suitable font definitions have now been added to `t1cmd.fd` and so from now on

```
\usepackage[T1]{fontenc}
...
 \textsc{\textsl{Slanted Small Caps}};
 \textsc{\textit{Italic Small Caps}};
 \bfseries
 \textsc{\textsl{Bold Slanted Small Caps}};
 \textsc{\textit{Bold Italic Small Caps}}.
```

will give the expected result: Slanted Small Caps; Italic Small Caps; Bold Slanted Small Caps; Bold Italic Small Caps.

Given that the Computer Modern fonts in `T1` do not have real italic small caps but only slanted small caps, the latter is substituted for the former. This is why both work in the above, but there is no difference between the two (and you get a substitution warning for the `\textit\textsc` shape combination). *(github issue 782)*

EC sans serif at small sizes

The EC (T1 encoded Computer Modern) sans serif fonts have errors at small sizes: the medium weight is bolder and wider than the bold extended. This makes them unusable at these small sizes. The default `.fd` file has therefore been adjusted to use a scaled down 8pt font instead. *(github issue 879)*

Improve font series handling with incorrect `.fd` files

By convention, the font series value is supposed to contain no `m`, unless you refer to the "medium" series (which is represented by a single `m`). For example, one should write `c` for "medium weight, condensed width" and not `mc`. This was one of the many space-conserving methods necessary in the early days of LaTeX 2ε.

Some older `.fd` files do not obey that convention but use `mc`, `bm`, etc., in their declarations. As a result, some font selection scheme functionality was not working when confronted with such `.fd` files. We have therefore augmented `\DeclareSymbolFont` and `\SetSymbolFont` to strip any surplus `m` from their series argument so that they do not unnecessarily trigger font substitutions. Regardless of this support such `.fd` files should get fixed by their maintainers. *(github issue 918)*

Detect nested `minipage` environments

Nesting of `minipage` environments is only partially supported in LaTeX and can lead to incorrect output, such as overfull boxes or footnotes appearing in the wrong place; see [1, p. 106]. However, until now there was no warning if that happened. This has been changed and the environment now warns if you nest it in another `minipage` environment that already contains footnotes. *(github issue 168)*

Robust commands in package options

With the standard key-based option handler added in the last release, or with contributed packages offering similar features, users may expect to be able to use a package option such as `[font=\bfseries]`. Previously this failed with internal errors as the option list was expanded via `\edef`. This has now been changed to use the existing command `\protected@edef` so that any LaTeX robust command should be safe to pass to a key value option. *(github issue 932)*

Improve l3docstrip integration into docstrip

In 2020 we merged l3docstrip.tex into docstrip.tex to support the `%<@@=`⟨*module*⟩`>` syntax of expl3; see [2]. However, this support was incomplete, because it didn't cover docstrip lines of the form `%<+...>` or `%<-...>`. This was never noticed until now, because usually `%<*...>`

[1] The LaTeX format contains declarations for many font families already. This was done in 2020 to quickstart the use of the symbols in the kernel, but it is really the wrong place for such declarations. Thus, for new fonts the declarations should be placed into the corresponding `.fd` files.

blocks are used. Now all lines in a `.dtx` file are subject to the `@@` replacement approach. *(github issue 903)*

LuaTeX callback efficiency improvement

The mechanism for providing the `pre/post_mlist_to_hlist_filter` callbacks in LuaTeX has been improved to make it more reusable and to avoid overhead if these callbacks are not used. *(github issue 830)*

Rule-based ordering for LuaTeX callback handlers

In LuaLaTeX the callback handlers used to be called in the order in which they were registered in, but this was often rather fragile. It depends a lot on the load order and any attempts to enforce a different order required unregistering and reregistering the handlers to be reordered. Additionally, even if some ordering constraints where enforced that way, another package loaded later could accidentally overwrite it.

To improve this, we now order the callback handlers based on ordering rules similar to the hook rules.

When registering a callback which should run before or after another callback, `luatexbase.declare_callback_rule` can now be used to record this ordering constraint. For example

```
luatexbase.add_to_callback
 ('pre_shaping_filter', my_handler, 'my_name')
luatexbase.declare_callback_rule
 ('pre_shaping_filter',
            'my_name', 'before', 'other_name')
```

will ensure that `my_handler` will always be called before the handler registered as `other_name`.

This also means that the order in which callbacks are registered no longer implicitly defines an order. Code which relied on this implicit order should now define the order rules explicitly.

Bug fixes

Prevent TeX from losing a `\smash`

When TeX is typesetting a fraction, it will rebox the material in either the numerator or denominator, depending on which is wider. If the repackaged part consists of a single box, that box gets new dimensions and if it was built using a `\smash` that effect vanishes (because a smash is nothing other than zeroing some box dimension, which now got undone). For example, in the line

```
\frac{1}{2} = \frac{1}{\smash{2^X}}
            \neq \frac{100}{\smash{2^X}}
```

the 2 in the denominators was not always at the same vertical position, because the second `\smash` was ignored due to reboxing:

$$\frac{1}{2} = \frac{1}{2^X} \neq \frac{100}{2^X}$$

The differences are subtle but noticeable. This is now corrected and the `\smash` is always honored. Thus now you get this output:

$$\frac{1}{2} = \frac{1}{2^X} \neq \frac{100}{2^X}$$

(github issue 517)

Resolve an issue with `\mathchoice` and `localalphabets`

The code for keeping a number of math alphabets local (introduced in 2021; see [3]) used `\aftergroup` to do some cleanup actions after a formula had finished. Unfortunately, `\aftergroup` can't be used inside the arguments of the `\mathchoice` primitive and as a result one got low-level errors if the freezing happened in such a place. The implementation was therefore revised to avoid the `\aftergroup` approach altogether. *(github issue 921)*

Reporting of unused global options when using key/value processing

Using the new key/value option processor did not properly report any unused global options when it was used in handling class options. This has now been corrected. *(github issue 938)*

Changes to packages in the graphics category

Fix a `\mathcolor` bug

The `\mathcolor` command introduced in [4] needs to scan for following sub- and superscripts, but if it did so at the end of an alignment cell, e.g., in a `array` environment, the `&` was evaluated too early, causing some internal errors. This is now properly guarded for. *(github issue 901)*

Changes to packages in the tools category

array: Correctly identify single-line m-cells

Cells in m-columns that contain only a single line are supposed to behave like single-line p-cells and align at the same baseline. To test for the condition, array used to compare the height of the cell to the height of the strut used for the table rows. However, the height of that strut depends on the setting of `\arraystretch` and if you made this negative (or very large) the test came out wrong. Therefore, we now test against the height of a normal strut to ensure that single-line cells are correctly identified as such (unless their content is truly very tall, in which case aligning is pointless anyway). *(github issue 766)*

References

[1] Leslie Lamport. LaTeX: A Document Preparation System: User's Guide and Reference Manual. Addison-Wesley, Reading, MA, USA, 2nd edition, 1994. ISBN 0-201-52983-1. Reprinted with corrections in 1996.

[2] LaTeX Project Team: *LaTeX 2ε news 32*.
https://latex-project.org/news/latex2e-news/ltnews32.pdf

[3] LaTeX Project Team: *LaTeX 2ε news 34*.
https://latex-project.org/news/latex2e-news/ltnews34.pdf

[4] LaTeX Project Team: *LaTeX 2ε news 35*.
https://latex-project.org/news/latex2e-news/ltnews35.pdf

[5] LaTeX Project Team: *LaTeX 2ε font selection*.
https://latex-project.org/help/documentation/