LaTeX News

Issue 33, June 2021

Contents

Introduction 1

Extending the hook concept to paragraphs 1

Extending the hook concept to commands 2

Other hook business 2
- Shipping out a page while bypassing hooks . . 2
- A new Lua callback in ltshipout, for custom attributes 2

Improved handling of file names 2
- File names with spaces, multiple dots or UTF-8 characters 2
  - Consequences for file names in \include . 2
- Normalization of robust commands in file names 2
- Fix for filecontents with UTF-8 chars in the file name 2

Updates to the font selection scheme 3
- A new hook in \selectfont 3
- Change of font series/shape delayed until \selectfont 3

Glyphs, characters & encodings 3
- Improved copy & paste for pdfTeX documents . 3
- Support for more Unicode characters 3
- More "dashes" in encodings OT1, T1 and TU . . 3
- Poor man's \textasteriskcentered 3
- The characters from textcomp are in the kernel 3
  - A note on the history of "text symbols" . 3

New or improved commands 4
- Adjusting itemize labels with \labelitemfont 4
- Producing several marks for one footnote . . . 4
- Allow \nocite in the preamble 4
- Made \\ generally robust 4
- Allow extra space between name and address in letter class 5
- Additions to \tracingall 5

Code improvements 5
- Execute \par at the end of \marginpar 5
- Execute \AtEndDocument hook in vertical mode 5
- Color groups made permanent 5
- Provide the raw option list to key/value option handlers 5
- New for latexrelease: \NewModuleRelease . . . 6
- Small fix for rolling back prior to 2020-02-02 . 6

Changes to packages in the graphics category 6
- Removed warning when loading graphics files . 6
- Fixed loading of gzipped PostScript files . . . 6

Changes to packages in the tools category 6
- layout: Added language options 6
- array and longtable: Make \\ generally robust . 6
- longtable: General bug fix update 6
- trace: Additions to \traceon 6
- bm: Better support for commands with optional arguments 6

Changes to packages in the amsmath category 6

Introduction

The focus of the June 2021 release is to provide further important building blocks for the future production of reliable tagged PDF output (see [1]); these enhancements are discussed in the next two sections.

Subsequent sections describe quite a number of recent smaller enhancements and fixes. As usual, more detail on individual changes can be found in the `changes.txt` files in the distribution and, of course, in the documented sources [2].

Extending the hook concept to paragraphs

Largely triggered by the need for better control of paragraph text processing, in particular when producing tagged PDF output, we have changed LaTeX so that the kernel gains control both at the start and at the end of each paragraph. This is done in a manner that is (or should be) transparent to both packages and documents.

Besides the addition of internal control points for the exclusive use of the LaTeX kernel, we also implemented four public hooks that can be used in packages or documents (via the normal hook management declarations) to achieve special effects, etc. Until now, such enhancements required redefinitions of `\everypar` or `\par`, which led to the usual issues since such changes can easily conflict with changes made by other packages.

The documentation of these new "paragraph hooks", together with a few examples, is in `ltpara-doc.pdf` and, for those who want to study it, the (quite interesting) code can be found in `ltpara-code.pdf`. Additionally,

both of these files are included as part of the full kernel documentation in `source2e.pdf`.

Extending the hook concept to commands

Up to now, hook management covered hooks for only a few core areas, such as the hooks for the `\shipout` process or those in the `document` environment, as well as some "generic" hooks, both for file loading (helpful for patching such files) and for arbitrary environments (the hooks executed within `\begin` and `\end`). This concept of "generic hooks" has now been extended to provide `/before` and `/after` hooks for any (document-level) command—in theory at least.

In practice, these new generic `cmd` hooks, especially the `cmd/.../after`, hooks may fail with commands that are too complex to be automatically patched, breaking if the hook contains any code. These restrictions are documented in `ltcmdhooks-doc.pdf`. However, given that these hooks are mainly meant for developers who wish to provide better interoperability between different packages, and between packages and the LaTeX kernel, these restrictions are, we hope, of minor importance. Indeed, for commands where this mechanism can't be applied, one is in the same situation as before; and for all others there will be a noticeable improvement.

These hooks will be especially important for our current project to provide accessible and tagged PDF output [1] because we will eventually have to patch many third-party packages, and this must be done in controlled and standardized ways.

Other hook business

Shipping out a page while bypassing hooks

In the 2020 October release, several hooks were added to control the process of constructing and shipping out a page box: these support, for example, the addition of background or foreground material to some or all pages.

We have now added a command, called `\RawShipout`, which does not do any rebuilding of the page box and so does not run most of these hooks. When using this new command, essential internal book-keeping is still carried out, such as updating the `totalpages` counter and adding `shipout/firstpage` or `shipout/lastpage` material when appropriate.

A new Lua callback in ltshipout, for custom attributes

For use just before shipping out a page, there is now a LuaTeX callback `pre_shipout_filter` to contain final adjustments to the box being shipped out. This is particularly useful for LuaTeX packages which flag (using, for example, attributes or properties) elements on a page in order to apply effects (such as the insertion of "color commands") to these elements at shipout.

Improved handling of file names

File names with spaces, multiple dots or UTF-8 characters

In one of the recent LaTeX releases we improved the interface for specifying file names so that they can now safely contain spaces (as is common these days), more than one dot character, and also UTF-8 characters outside the ASCII range. In the past this was only possible by applying a special syntax in the case of spaces, while file names with several dots often failed, as did most UTF-8 characters.

Consequences for file names in `\include`: TeX has a built-in rule saying that you can normally leave out the extension if it is `.tex`. Thus `\input{file}` and `\input{file.tex}` both load `file.tex` (if it exists). While this is convenient most of the time, it is a little awkward in some scenarios (for example, when both `file` and `file.tex` exist) and also when you manually try to implement the rule.

LaTeX therefore had one special syntax for `\include` and `\includeonly`: they always expected that their arguments contain a file name[1] with no extension given, so that it had to be `.tex`. Thus, when you mistakenly wrote `\include{mychap.tex}` (for example, because you changed from `\input` to `\include`), LaTeX went ahead and looked for the file `mychap.tex.tex` for inclusion and tried to use the file `mychap.tex.aux` for internal (auxiliary) information. The reason was that `\include` had to construct both of these file names from the given argument and it didn't bother to do anything special with the supplied extension `.tex`.

With the new implementation this has changed: the extension `.tex` now gets removed/ignored if it was supplied. Thus `\include{mychap.tex}` now no longer looks for `mychap.tex.tex` but loads `mychap.tex` and uses `mychap.aux`. *(github issue 486)*

Normalization of robust commands in file names

The handling of file names has been modified so that `\string` is applied to normalize robust commands within the file name. Previously, for example, `\input{\sqrt{2}}` would cause LaTeX to loop indefinitely whereas with the new normalization it looks for the file named `sqrt {2}.tex` (and therefore very likely reports "file not found"). *(github issue 481)*

Fix for `filecontents` with UTF-8 chars in the file name

Since a few releases back, the `filecontents` environment has allowed UTF-8 characters in the file name. There was, however, a bug that would not allow *over*writing a file with UTF-8 characters in its name. This has been fixed and now `filecontents` allows any characters in the file name. *(github issue 415)*

[1] In the case of `\includeonly`, a comma-separated list of such names.

Updates to the font selection scheme

A new hook in \selectfont

After \selectfont has changed the font, we now run a hook (selectfont) so that packages can make final adjustments. This functionality was originally provided by the everysel package but our implementation is slightly different and uses the standard hook management. *(github issue 444)*

Change of font series/shape delayed until \selectfont

With the NFSS extensions introduced in 2020, the font series and shape settings can be influenced by changes to the font family. The settings of these two are now therefore delayed until \selectfont is executed; this avoids unnecessary or incorrect substitutions that may otherwise happen due to the order of declarations. *(github issue 444)*

Glyphs, characters & encodings

Improved copy & paste for pdfTeX documents

When compiling with pdfTeX, additional information (from the file glyphtounicode.tex) is now added automatically to the PDF file in order to improve copying from, and searching in, text.

In particular, this allows the most common ligatures to be copied as intended from all generated PDF files without the need to explicitly load the package cmap. *(github issue 465)*

Support for more Unicode characters

LaTeX is quite capable of typesetting characters such as "ṃ", but until now it could not access some Unicode characters from the Latin Extended Additional block. This meant that, for example, there were no Unicode mappings for some characters that are used to write Sanskrit words in Latin transliteration (as seen in books about yoga, Buddhist philosophy, etc.). These characters have now been added so that they can be entered directly instead of using \d{m}, etc. *(github issue 484)*

More "dashes" in encodings OT1, T1 and TU

When pasting in text from external sources, one can encounter these three Unicode characters "2011 (non-breaking hyphen), "2012 (figure dash) and "2015 (horizontal bar), in addition to the more common "2013 (en-dash) and "2014 (em-dash). In the past, these first three produced an error message when used with pdfTeX (since they are not available in OT1 or T1 encoded fonts). They now typeset an approximation to the glyph: e.g., the "figure dash" is approximated by an en-dash.

With Unicode engines they either work (when the glyph is contained in the selected Unicode font) or they typeset nothing, producing a "Missing character" warning in the log file.

With all engines these characters can also now be accessed using the command names \textnonbreakinghyphen, \textfiguredash and \texthorizontalbar, respectively. *(github issue 404)*

Poor man's \textasteriskcentered

The \textasteriskcentered symbol, used as part of the set of footnote symbols in LaTeX, is assumed to be implemented by every font with the TS1 encoding (when pdfTeX is used) or with the TU encoding for the Unicode engines. That assumption is unfortunately not correct for all fonts since, for example, the stix2 fonts don't provide this glyph. A result is that one gets missing glyph messages when using \thanks, etc.

Therefore \textasteriskcentered now checks whether there is such a glyph and, if not, uses a normal "*", but slightly enlarged and lowered. This may not be perfect in all cases, but it is certainly better than no glyph showing up. *(github issue 502)*

The characters from textcomp are in the kernel

A couple of releases back, the functionality of the textcomp package was integrated into the LaTeX kernel. Thus it is no longer necessary to load this package in order to access glyphs such as \textcopyright, \texteuro or \textyen.

At this time the opportunity was also taken to bring some order to the chaos surrounding the question: "which glyphs from the TS1 encoding are available in a given font?". This was done using an approach based on font families and collections, with the differing glyph coverage of the 'text symbols' being indicated by assigning to a font family or collection a "sub-encoding number" that indicates which glyphs from the TS1 encoding are guaranteed to be available when using a font from that family or collection. This assignment ensures that LaTeX always errs on the side of caution, possibly claiming that a glyph is not available even when it in fact is.

A note on the history of "text symbols" and the TS1 encoding: The "text symbol encoding" (TS1) was originally designed at the Cork Conference as a companion to the T1 encoding. In it various symbols that are not subject to hyphenation got assembled and the textcomp package was developed to make them accessible. Unfortunately the TeX community was a bit too enthusiastic and included several symbols only available in a few TeX fonts and some, such as the capital accents, not available at all but developed as part of the reference font implementation.

In hindsight that was a very bad idea because it meant that other existing fonts (at the time) and later new fonts that got developed were unable to provide the full set of glyphs that made up the TS1 encoding. For

existing free PostScript fonts people took the extra effort and produced virtual fonts that faked (some) of the missing glyphs. But this was and is a time-consuming effort so it was done for only a few basic fonts. But even then, only some fonts included all glyphs from TS1 so the textcomp already back then contained a long list, dividing fonts into 5 categories according to which glyphs were implemented and which were missing.

When we recently integrated the functionality of the textcomp into the LaTeX kernel many new free fonts had appeared and unfortunately the chaos around the question "which glyphs of the TS1 encoding are implemented by which font" had increased with it. Not only did one find many new holes, it was next to impossible to order the set of fonts into a reasonable set of sub-encodings that are contained in each other in a single sequence.

In the end we decided on nine or ten sub-encodings with a reasonable number of fonts in each so that all fonts implemented all glyphs of the sub-encoding they got mapped to. Thus when typesetting with a font one could be sure that a command like \textcopyleft would either typeset the requested character (if the glyph was part of the sub-encoding the font belonged to) or it would raise an error, saying that the glyph is unavailable in that font. The mapping would ensure that LaTeX always errs on the side of caution, because it might claim a glyph is unavailable even though in fact it is.

For example, the old pcr (PostScript Courier) font (as well as most other older PS fonts) is mapped to sub-encoding 5 and therefore claims that \textasciigrave is unavailable even though in fact for Courier this is not true. If one uses such a font and this becomes an issue then there are a couple (suboptimal) possibilities. For one, one can alter the mapping of Courier and pretend that belongs to a fuller sub-encoding, e.g.

```
\DeclareEncodingSubset{TS1}{pcr}{2}
```

The downside is, that LaTeX then believes other glyphs that are in fact unavailable are also there, so that it is important to check that the final document doesn't have some missing glyphs.

An alternative is to pretend that \textasciigrave can always be taken from the TS1 encoding (no questions asked):

```
\DeclareTextSymbolDefault{\textasciigrave}{TS1}
```

Again there is a danger that this is not true when it is used with a different font and would then generate a missing glyph.

Finally, and possibly the best solution, if not impossible for other reasons, is to simply use a different font, for example, to use the TeX Gyre Cursor font (a reimplementation of Courier with a much more complete glyph set).

New or improved commands

Adjusting itemize labels with \labelitemfont

The command \labelitemfont was introduced already with the LaTeX release 2020-02-02, but back then we forgot to describe it, so we do this now. Its purpose is to resolve some bad formatting issues with the itemize environment and also to make it easier to adjust the layout when necessary. What could happen in the past was that the itemize labels (e.g., the •) would sometimes react to surrounding font changes and could then suddenly change shape, for example to •.

This new command \labelitemfont, which defaults to \normalfont, can be used to provide additional control in the typesetting of each label. Thus by choosing different settings other effects can be achieved. Here are two examples:

```
\renewcommand\labelitemfont
   {\normalfont\fontfamily{lmss}\selectfont}
\renewcommand\labelitemfont
   {\rmfamily\normalshape}
```

The first definition will take the symbols from the font Latin Modern Sans, so that you get ▪, −, ∗ and · ; while the second variant freezes the font family and shape, but leaves the series as a variable quantity, so that an itemize in a bold context would show bolder symbols. Making \labelitemfont empty would give you back the buggy old behavior. *(github issue 497)*

Producing several marks for one footnote

It is sometimes necessary to reference the same footnote several times: i.e., to produce several footnote marks using the same number or symbol. This is now easily possible by placing a \label within the referenced \footnote and referencing this label by using the new command \footref. This means that footnote marks can be generated to refer to arbitrary footnotes (including those in minipages).

This \footref command has previously been available, but only when using certain classes or the footmisc package. *(github issue 482)*

Allow \nocite in the preamble

A natural place for \nocite{*} would be the preamble of the document, but for historical reasons LaTeX issued an error message if it was placed there. This command is now allowed in the preamble. *(github issue 424)*

Made \\ generally robust

In 2018 most LaTeX user-level commands were made robust, including the \\ command. However, \\ gets redefined in various environments and not all these cases were caught: such as, in particular, its use as the row delimiter in tabular structures. This has been corrected so that \\ should now be robust in all circumstances.

This change also fixed one anomaly present in the past: in a tabular preamble of the form

```
{l>{\raggedright}p{10cm}r}
```

a \\ in the second column would have the definition used within \raggedright and so it would not indicate the (premature) end of the tabular. Thus, for example,

```
a & b1 \\ b2 & c \\
```

was interpreted as a single row of the tabular (as intended), whereas

```
a &    \\ b2 & c \\
```

resulted in two rows! This happened because the \\ directly following the & got interpreted while it still had the "end the row" meaning and not yet the "start a new line within the second column" meaning.

With \\ now being robust, the special scanning mode initiated by the & ends immediately when this command is seen: the second column is therefore then started, which results in the \\ being interpreted as being within that column and hence as having its expected, within-column, meaning.

We have restored consistency here: now both of the above lines produce a single tabular row. As before, you can put \raggedright\arraybackslash in the tabular's preamble for a column to ensure that \\ is always interpreted as a tabular row separator when used in that column. And you can use \tabularnewline to explicitly ask for a new table row, even when \\ has a different meaning within the current column.

(github issue 548)

Allow extra space between name and address in letter class

The \opening command in the letter class expects the name and address to be separated by \\, but it didn't allow the use of an optional argument to add some extra space after the name. The code has now been slightly altered to allow this. *(github issue 427)*

Additions to \tracingall

In July 2020 David Jones suggested an extension to TEX engines, that added the possibility to set \tracinglostchars=3 in order to generate an error message in case some character is missing from a font. In previous years, a warning about a missing character was silently printed to the .log file (if \tracinglostchars > 0) and to the terminal (if > 1). This extension was added for TEX Live and MiKTEX (except in Knuth's TEX, of course), so that with \tracinglostchars > 2 you now also get an error message for each missing glyph.

Later, in January 2021, Petr Olšák suggested yet another extension: a new primitive parameter \tracingstacklevels that, when both it and \tracingmacros are positive, will add to the tracing information for each macro a visual indication (using dots) of its nesting level in the macro expansion stack.

These changes have both now been added to LATEX's debugging macros \tracingall and \tracingnone, so that these two new extensions are activated/deactivated as appropriate, so long as the TEX engine supports them. An example document demonstrating these parameters is in the linked GitHub issue. *(github issue 524)*

Code improvements

Execute \par at the end of \marginpar

Previously, LATEX ended a \marginpar without ever explicitly calling \par. This command is now explicitly added because it is essential to the correct working of the paragraph hooks.

Another case where this issue caused problems was the lineno package, where the last line was not numbered if the \marginpar ended without an explicit \par.

(github issue 489)

Execute \AtEndDocument hook in vertical mode

Until now \end{document} executed the code from the \AtEndDocument hook as its first action. This meant that this hook was executed in horizontal mode if the user left no empty line after the last paragraph. As a result, one could get a spurious space added when, for example, that code contained a \write statement. This was fixed and now \enddocument first issues a \par to ensure that it always goes into vertical mode.

(github issue 385)

Color groups made permanent

The use of color in certain LATEX constructs, especially boxes, needs an extra layer of grouping to ensure that the color setting does not *escape* and continue outside the box when it shouldn't. To support this, the LATEX kernel defines a number of commands, e.g., \color@begingroup to be used in such places.

Until now, these commands were initially set as no-ops and only the color packages redefined them to become real groups; this methodology complicates the coding as one has to account for a group being present or not (depending on what is loaded in the document). The kernel therefore now permanently adds these "color groups". *(github issue 488)*

Provide the raw option list to key/value option handlers

Before any further processing of the option list, the original (un-normalized, "raw" and unchanged) list of package or class options is now saved, as \@raw@opt@...; this list is not used by the standard option processing code but it is now available for use by extended class/package processing systems. Note that, for compatibility reasons, the standard option processing code has not been changed.

One aspect of this change does affect the standard processing: any tokens to the right of an `=` sign are removed from consideration when constructing the "unused option list". For example, in this release `clip=true` and `clip=false` both contribute `clip` to the list of options that have been used. *(github issue 85)*

New for latexrelease: `\NewModuleRelease`
To explain the need for this new feature, we shall consider the following example: in the 2020-10-01 release, LaTeX's new hook management system was added to the kernel (see [3]) and, as with all changes to the kernel, it was added to latexrelease; this made it possible to roll back to a date where this module didn't yet exist, or to roll forward from an older LaTeX release to get the hook management system (by loading the latexrelease package). However, this method of rolling back from a later release to the 2020-10-01 release didn't quite work because it would try to define all the commands from lthooks again; and this would of course result in the expected errors from commands defined with `\newcommand` or (as in lthooks) `\cs_new:Npn`.

To solve such issues, we now provide `\NewModuleRelease` so that completely new modules can be defined using the facilities of latexrelease in such a way that, when rolling back or forward, the system will know whether the code of the new module has to be read or completely ignored. More details on this can be found in the latexrelease documentation (get this with `texdoc latexrelease`). *(github issue 479)*

Small fix for rolling back prior to 2020-02-02
Whereas the latexrelease package can usually emulate an older LaTeX kernel without much problem, rolling back to before the 2020-02-02 release didn't work properly: this is because the management of the `\ExplSyntaxOn/Off` status for packages (after an expl3-based package is loaded) cannot be removed by the rollback without messing up the catcodes. This has been fixed so that rollback is now more careful not to leave `\ExplSyntaxOn` after a package ends. *(github issue 504)*

Changes to packages in the graphics category

Removed warning when loading graphics files
A previous release sometimes mistakenly caused a (false) warning message to appear when using a generic graphics rule to find and load a graphics file with an unknown extension. This warning would incorrectly say that the file was not found, whereas the file would in fact be correctly loaded. The warning now doesn't show up in that case. *(github issue 516)*

Fixed loading of gzipped PostScript files
A previous release mistakenly changed the file searching mechanism so that compressed PostScript graphics files would raise an error when being loaded with `\includegraphics`. This has been fixed so that gzipped graphics files now load correctly. *(github issue 519)*

Changes to packages in the tools category

layout: Added language options
This package now recognizes `japanese` and `romanian` as language options. *(github issues 353 and 529)*

array and longtable: Make `\\` generally robust
The fix for this issue was also applied to these packages; see above. *(github issue 548)*

longtable: General bug fix update
This is a minor update to the longtable package that fixes several reported bugs: notably the possibility of incorrect page breaks when floats appear on the page where a `longtable` starts. As this may affect page breaking in existing documents, a rollback to longtable 4.13 (`longtable-2020-01-07.sty`) is supported.
(gnats issue tools/2914 3396 3512)
(github issue 133 137 183 464 561)

trace: Additions to `\traceon`
The `\tracingstacklevels` and `\tracinglostchars` extensions to `\tracingall` (see above) were also added to `\traceon` in the trace package, so its users can also benefit from these new debugging possibilities.
(github issue 524)

bm: Better support for commands with optional arguments
Some uses of optional arguments in `\bm` stopped being supported (in 2004) when `\kernel@ifnextchar` was used internally by the format instead of `\@ifnextchar`. This update handles both versions of this command and restores the original behavior.

In addition, package options for guiding the use of "poor man's bold" in fallback situations were added.
(github issue 554)

Changes to packages in the amsmath category

The fix for issue 548 was also applied in amsmath; see above. *(github issue 548)*

References

[1] Frank Mittelbach and Chris Rowley: *LaTeX Tagged PDF—A blueprint for a large project.* `https://latex-project.org/publications/indexbyyear/2020/`

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

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