LaTeX News

Issue 41, June 2025 (LaTeX release 2025-06-01)

Contents

Introduction

We are continuing to work on the support of tagged PDF output and on wider kernel development, much of which is needed to make this happen. Probably the most notable changes in this latest release of the kernel are those in the output routine and in the mark mechanism: these are described in the first two sections.

Work also continues apace on further aspects of the tagging project, where some highlights are: new sockets, better graphics tagging, improved math mode support and the promotion of PDF 2.0.

In addition to this tagging-focussed work, we have also made advances in these areas: a new argument type for use in `\NewDocumentEnvironment` and friends which will make working with verbatim content a *lot* easier; further improvements to case changing; and, of course, several bug fixes.

Finally, we have highlighted a few of the recent changes in the L3 programming layer, where development work continues in parallel with other improvements to the LaTeX kernel; and we are integrating more, formerly "experimental", ideas into this core programming system.

A configurable output routine

For nearly 40 years LaTeX's output routine (the mechanism to paginate the document and attach footnotes, floats and headers & footers) was a largely hardwired algorithm with a limited number of configuration possibilities. Packages that attempted to alter any aspect of this process had to overwrite the internals, which led

to the usual problems: incompatibilities and out-of-date code whenever something was changed in LaTeX.

To improve this, and to support the production of accessible PDF documents, we have started to refactor the output routine by adding a number of hooks and sockets. This means that packages needing to adjust the output routine can do so safely, avoiding the dangers previously associated with such activities.

For packages that need to hook into the output routine we have implemented the following hooks:

build/page/before, build/page/after These two hooks enable packages to prepend or append code to the processing of each page in the output routine. They are implemented as mirrored hooks. Technically, they are executed at the start and the end, respectively, of the internal LaTeX 2_ε `\@outputpage` command. Currently, a number of packages change this command by adding code in exactly these two places—so they can now instead simply add code to these hooks.

build/page/reset Packages that set up special conventions for the main text (such as catcode changes, etc.) can use this hook to undo these changes within the output routine, so that they aren't applied to unrelated material such as the text for running headers or footers.

build/column/before, build/column/after These two hooks enable packages to prepend or append code to the column processing in the output routine. They are implemented as mirrored hooks. Technically, they are executed at the start and the end, respectively, of the internal LaTeX 2_ε `\@makecol` command. A number of packages alter `\@makecol` to place code in exactly these two places—they can now instead simply add their code to these hooks.

We have also added a number of sockets: for configuring the algorithm and also to support tagging. Two of these sockets are of interest for use in class files and also in the document preamble.

The first and most complex of these is the socket `build/column/outputbox`, which controls how the column text, the column floats (top and bottom) and the footnotes are combined in a column: i.e., their order and spacing.

Thus in order to change the layout, all one now has to do is to assign a suitable plug to this socket, like this:

`\AssignSocketPlug{build/column/outputbox}`
`{`⟨*plug-name*⟩`}`

For this socket we have provided the following plugs:

space-footnotes-floats After the galley text there is a vertical `\vfil` followed by the footnotes, followed by the bottom floats, if any.

footnotes-space-floats As before but the `\vfil` is between the footnotes and the floats.

floats-space-footnotes The floats come directly after the text, followed by a `\vfil` and then the footnotes at the bottom.

space-floats-footnotes Both floats and footnotes are pushed to the bottom, the footnotes coming last.[1]

floats-footnotes All excess space is distributed across the existing glue on the page: e.g., within the text galley, the separation between blocks, etc. The order is text, floats, footnotes.

footnotes-floats Like the previous plug, but floats and footnotes are swapped. This is the LaTeX default for newer documents, i.e., this plug is assigned to the socket when `\DocumentMetadata` is used.

footnotes-floats-legacy Like the previous plug, but LaTeX's bottom skip bug is not corrected: i.e., in ragged bottom designs where footnotes are supposed to be directly attached to the text, they suddenly appear at the bottom of the page when the page ends with a `\newpage` or `\clearpage`. While this is clearly a bug, it has been like this since the days of LaTeX 2.09; thus, for compatibility, we continue to support this behavior. This plug is assigned to the socket when `\DocumentMetadata` is *not* used.

By default the separation between the last line of text and the footnotes (`\skip\footins`) is not measured from the baseline of the last text line, but from its bottom. This goes back to plain TeX where it is done in this way. Similarly, `\textfloatsep` is added between text and bottom floats, not starting from the baseline of the last text line. Typographically speaking this is suboptimal, because it means that with `\flushbottom` in effect, the position of the last text line, when it is followed by footnotes or floats, depends on whether or not that line contains characters with descenders.

For this reason there is now also a socket `build/column/baselineattach` with a plug `on`: this causes the attachment of footnotes/floats to be measured from the baseline of the last text line. To mimic the behavior of old documents, this socket is, by default, assigned the plug `off`. For documents using `\DocumentMetadata`, the plug `on` will probably become the default here.

There are more configuration possibilities, mainly for class developers to use: documentation of these can be found in [4, §54 `ltoutput.dtx`].

Replacement for the legacy mark mechanism

LaTeX's legacy mechanism supported only two classes of marks, left and right marks, and setting the left mark

[1] There are two more permutations but neither of them has ever been requested; so these two are not set up by default—doing that in a class would be trivial though.

(with \markboth) always altered the state of the right mark as well, i.e., they were far from independent. For generating running headers with "chapter titles" on the left and "section titles" on the right, they work reasonably well but without much flexibility: e.g., \leftmark always generated the last "left"-mark on the page, while \rightmark always generated the first "right"-mark.

A few releases ago [2, p. 76] we therefore introduced a new mark mechanism for LaTeX, one that supports any number of truly independent mark classes. This mechanism also offers the ability to query the mark status at the top of the page, something that wasn't previously available at all.

Up to now these two mechanisms coexisted, with completely separate implementations; but we have now retired the legacy code and reimplemented its public interfaces using the new concepts. Thus the old commands (\markboth, \markright, \leftmark and \rightmark) remain supported, but internally these commands all use \InsertMark, etc.

Existing document classes, and documents using the legacy interfaces, will therefore continue to work without any modifications; but they now use a single underlying implementation. Also, new documents can benefit from the additional flexibility, e.g., by being able to display not only the first right-mark (\rightmark or \FirstMark{2e-right}) but also, or alternatively, the last right-mark (\LastMark{2e-right}) or the top right-mark (\TopMark{2e-right}), etc.

More information concerning all of this extended functionality can be found in [3].

News from the Tagged PDF project

In the Tagged PDF project we have now reached a state where, within certain limits, it is possible to generate accessible PDF output that conforms to PDF/UA for arbitrarily complex documents as long as they only use (a growing number of) compatible packages and classes.

The focus for this release was on adding special sockets for tagging support, on improving the tagging of math formulas, and on extending the tagging support for various types of graphics.

New Metadata keys, to activate tagging

Up to now users had to activate tagging by loading modules from latex-lab with the help of the testphase key. Further configuration of the tagging then had to be done by using the \tagpdfsetup command. We now offer Metadata keys for this that do not use "test" in their names, reflecting the fact that producing tagged PDF documents has become "production-ready".[2]

The tagging key allows for the activation and deactivation of the tagging support. It accepts the three values on, off and draft. When this key is used it loads the tagpdf package and all the modules that we currently recommend should be loaded.[3] The list of loaded modules will be adjusted as needed as the project progresses. For reference, it is also written to the log. Setting tagging=off loads the same set of modules and then deactivates the tagging commands in the class/before hook; and tagging=draft leaves the tagging commands active, so as to preserve warnings and errors in the tagging, but it deactivates the writing of the structure tree at the end of the compilation. This can save time when drafting a long document.

The tagging-setup key allows configuration of the tagging. It accepts as values all the keys that can be used in \tagpdfsetup, such as the math/setup key described below. It knows about both the key modules, which allows overwriting of the set of loaded modules, and the key extra-modules, which allows loading of experimental modules that are not yet in latest. The tagging-setup key implies tagging=on so that, if this key is used, then it is not necessary to also set the tagging key unless you want to turn tagging off, or to set it to draft.

With these new Metadata keys a standard setup might look like this:

```
\DocumentMetadata{
   pdfstandard={UA-2,A-4f},
   tagging=on,
   tagging-setup=
      {math/setup=mathml-SE,
       extra-modules=verbatim-alt}
 }
```

New value latest for testphase key

With the new keys for enabling tagging the use of the testphase key is now of minor importance and mainly of interest for developers and for backwards compatibility.

With this release it also supports the value latest. This will load all modules that we currently recommend should be loaded, so that it is not necessary to specify a long list of individual modules. The list of loaded modules will be adjusted as needed when the project progresses. For reference, it is also written to the log.

Sockets for tagging support

A lot of the tagging support in packages is handled through the socket-and-plug mechanism that was introduced in LaTeX 2023-11-01 [2, p. 93]. Sockets offer an easily used interface for package developers to invoke variable code at pre-specified places: code that then

[2] To be fully precise, this is true provided only compatible packages and classes are used: these are listed at https://latex3.github.io/tagging-project/tagging-status/.

[3] This set of modules can also be loaded with the key testphase=latest.

can be changed from outside the package by assigning a different plug to alter the processing.

For the tagging support, a specialized set of sockets is available: their plugs are invoked by using the `\UseTaggingSocket` command, instead of the normal `\UseSocket` command. This allows tagging to be turned off or on at high speed by the commands `\SuspendTagging` and `\ResumeTagging`, without the need to individually reassign plugs to each of the many tagging sockets [2, p. 97]. This is very useful when there is a need to typeset material several times during trials.

In the current release we now also offer three dedicated declaration commands for these "tagging sockets": this works better than directly using the underlying general socket interface. These new commands also better support the special conventions used for "tagging sockets". They are: `\NewTaggingSocket`, `\NewTaggingSocketPlug` and `\AssignTaggingSocketPlug`.

Setting the version to PDF 2.0

Creating a PDF 2.0 version is considered essential for any document that has substantial mathematical content. This is because only this PDF version supports the straightforward use of tags from the MathML namespace.

When `\DocumentMetadata` is used, LaTeX will therefore, by default, set PDF 2.0 as the PDF version. A different PDF version can, if required, be set by explicit use of the `pdfversion` key.

Setting up math tagging

With the LuaTeX engine there are now various options for the production of accessible math which are described in full detail in `latex-lab-math.pdf`. To simplify the setup, a new key `math/setup` can be used in `\tagpdfsetup` (or in `tagging-setup` as shown above) that accepts a comma list with the values `mathml-SE` (add MathML structure elements), `mathml-AF` (attach MathML associated files) or `tex-AF` (attach the TeX sources).

The use of $$...$$ for math displays

Use of the plain TeX method `$$...$$` in LaTeX, to mark up a display math formula, is not officially supported because it produces a fixed visual result that is not receptive to style changes such as the `fleqn` option. Instead, the recommended way is to use `\[ ... \]` or the `displaymath` environment. However, since many authors have used this input method in their documents, we are doing our best to support it for the production of accessible PDFs; but users should be aware that it has some limitations.

However, these accommodations for tagged PDF clash with the direct use of `$$` in environment definitions for special math environments (such as those defined in amsmath). The kernel therefore now contains the two commands `\dollardollar@begin` and `\dollardollar@end`. These new commands must be used by packages and classes to specify where inside an environment the displayed math formula starts and ends: their use is essential in order to make the package or a class compatible with tagging, and to allow its use when producing accessible documents. No more explicit `$$`s in code, please!

Package and class developers can prepare code to meet this new requirement by adding these two commands:

```
\providecommand\dollardollar@begin{$$}
\providecommand\dollardollar@end{$$}
```

and replacing every occurrence of `$$` with the appropriate start or end command.

Adding these `\providecommand` lines to classes and packages doesn't hurt but ensures that they will work with older LaTeX kernels.

Fixing the spacing after display math

When LaTeX produces accessible (tagged) PDF it has to add structure data in the PDF to mark (i.e., tag) individual elements. If the pdfTeX engine is used this has to be done with the help of `\pdfliteral`s, which are whatsit nodes like `\special` or `\write`. This means that they should be added only in places where these extra nodes do not affect the spacing—TeX can't, for example, look backwards past such a whatsit node, so consecutive spaces, that are normally collapsed into one, suddenly both appear when such a node separates them.

The situation is especially complicated in displayed math because there TeX adds penalties and spaces using low-level procedures that are not directly accessible from the macro level. Moreover, the PDF tagging structures have to be added somewhere in the middle of this processing: this is needed to ensure that the formula and these PDF structures do not get separated by a page break. Because of this it is necessary to use some fairly complex methods (essentially, we disable TeX's mechanisms and reprogram them on the macro level) to get the structure data in the right places.

Our first attempt to do this was slightly faulty and, in some cases, resulted in the addition of an incorrect `\parskip` space; this has now been corrected. The implementation that achieves this is a rather "interesting" study in obfuscated TeX coding—it is described in `latex-lab-math.pdf` for the interested.

When using LuaTeX the situation is much better because the necessary extra structures can be added at a later stage, after the formula has been typeset.

(tagging-project issue 762)

Local changes to spacing around math displays

Due to TeX's low-level handling of display math, it is very difficult to add the code needed for tagging

within such display math formulas whilst ensuring that such code always stays on the same page as the formula. This is because such code must be placed after the end of the display, but before the TEX engine adds a `\postdisplaypenalty` to the page. However, there is no way to add code in the middle of this low-level TEX processing, which is why we have to resort to complex gymnastics as already hinted at: we set `\postdisplaypenalty` locally to 10000 and also make sure that `\belowdisplayskip` when used by TEX is negative. Then we let TEX do its job and afterwards regain control via `\aftergroup` and insert the tagging code. Finally, we add the real `\postdisplaypenalty` and make a space correction.

With our first implementation of this approach it was not possible for a user to add an explicit `\postdisplaypenalty` or `\belowdisplayskip` setting inside the formula. In this release we have slightly altered our algorithm to make such user adjustments possible again. *(tagging-project issue 809)*

Extended support for pictures

The tagging of graphics has been reimplemented and now uses tagging sockets (see above). Document authors can choose between four tagging flavors on a per-graphic basis: as illustrative figures, as artifacts (i.e., decorations), as replacements for symbols, and if applicable as normal text (for example, "todo notes"). To this effect the options of `\includegraphics` and the environments `picture` and `tikzpicture` have been extended and now accept keys such as `alt` (for the description text of illustrative figures), `actualtext` (to set the symbol), and `artifact`. The code supports graphics produced using the tikz packages and "todo notes" from the todonotes package. The extended documentation in `latex-lab-graphics.pdf` lists the full set of options and also describes what authors of other graphic packages can do to make their packages tagging aware.

New or improved commands

Socket-and-plug conditionals

It is sometimes necessary, or helpful, to know whether a particular socket or plug exists (or whether a plug is assigned to a certain socket) and, based on such information, to take different actions. With the current release we added conditionals, such as `\IfSocketExistsTF`, to support such scenarios. Corresponding L3 programming layer conditionals are also provided. *(github issue 1577)*

Accessing the current counter

Counter commands such as `\alph`, `\stepcounter`, can now use the argument `*` to denote the *current counter* (in the sense used by `\label`). This is compatible with the use by the enumitem package of `\alph*` in item labels; and it is now generally available. Not all commands accept `*`; for example, `\counterwithin` and `\counterwithout` still require counter names as before. *(github issue 1632)*

Collecting environment bodies verbatim

The mechanisms provided with `\NewDocumentCommand`, etc., offer a powerful way to specify a range of types of document command and environment syntax. This includes the ability to collect the entire body of an environment, for cases where treating it as a standard argument is useful. It is also possible to use this mechanism to define arguments which grab their content verbatim. To date, however, it was not possible to combine these two ideas.

In this release a new specifier, `c`, has been introduced for use in `\NewDocumentEnvironment` and friends: this collects the body of an environment in a verbatim-like way. As with the existing `+v` specification, each separate line is marked by the special `\obeyedline` marker, which by default issues a normal paragraph. Thus, this new specifier is usable both for typesetting and for collecting file contents (the letter `c` indicates "collect code"). Thus, we may use[4]

```
\NewDocumentEnvironment
  {MyVerbatim}{!O{\ttfamily} c}
  {\begin{flushright}#1 #2\end{flushright}}
  {}
\begin{MyVerbatim}[\ttfamily\itshape]
  % Some code is shown here
  $y = mx + c$
\end{MyVerbatim}
```

to obtain

```
% Some code is shown here
$y = mx + c$
```

Code improvements

Refinement of `\MakeTitlecase`

We introduced `\MakeTitlecase` as a late addition to the June 2022 release, making use of the improved case code in the L3 programming layer. Compared to upper and lowercasing, making text titlecased is even trickier to get right: it can be applied either to the whole text, or on a word-by-word basis.

A subtle issue was reported concerning the L3 programming layer (`https://github.com/latex3/latex3/issues/1316`); this is related to how we deal with the case changing of "words" but it also shows up when you titlecase some text stored in a command.

[4] Proper support for Tagged PDF needs additional code which is not shown here.

We have looked again at how to implement `\MakeTitlecase` in order to make it as predictable as possible, and we have made a change in this release. The command no longer tries to lowercase text before applying titlecasing, and it therefore gives correct results for text stored in commands.

We have also added an additional key to the optional argument to `\MakeTitlecase` which allows the user to decide if the case change gets applied only to the first word (the default) or to all the words.

Tab character as a special character

In LaTeX News 38 [2, p. 95], we described a change to `\verb`, etc., that makes the tab character equivalent to a space; we have now completed this work by adding the tab character to the list of characters covered by `\dospecials`. This allows tab to be used, for example, in a `v` specification document command without the need for additional steps.

Refinement of `v` specification category codes

Work on verbatim argument handling has highlighted that it is problematic to store all characters as "other" (category code 12) when using a `v` specification in `\NewDocumentCommand`, etc. We have therefore now revised this so that characters of category code letter retain their original category code.

Logging declarations of commands and symbols

For thirty years the documentation claimed that `\DeclareTextSymbol`, `\DeclareTextCommand` and friends all log their changes. However, in contrast to their math counterparts, they never in fact did so. This behavior has now finally been corrected.

(github issue 1242)

Improved management of the NFSS font series

LaTeX's font selection mechanism (NFSS) supports 9 weight levels, from ultra-light (`ul`) to ultra-bold (`ub`), and also 9 width levels, from ultra-condensed (`uc`) to ultra-expanded (`ux`). In the February 2020 release this mechanism was extended, so that requests to set the weight or the width attributes of a font series are combined in a sensible way [2, p. 52]: for example, if you typeset a paragraph in a condensed face using `\fontseries{c}\selectfont` and then you use `\textbf` inside the paragraph, a bold condensed face is selected. The combination of such values is done by consulting a simple lookup table whose entries are defined by using the command `\DeclareFontSeriesChangeRule`.

Until now, this lookup table was missing some entries, especially with regard to rarely used width values. In such cases, the series values were not combined as expected. This has been fixed (thanks to Maurice Hansen) by adding numerous `\DeclareFontSeriesChangeRule` entries so that, when combining these font series values, the full range of weights (from `ul` to `ub`) and widths (from `uc` to `ux`) is now supported. *(github issue 1583)*

Supporting the `ssc` and `sw` font shapes

The `ssc` font shape (spaced small capitals) is supported in LaTeX through the commands `\sscshape` and `\textssc`. However, until this release there were no font shape change rules defined for this, admittedly seldom available, shape; so

```
\sscshape\itshape
```

changed unconditionally to `it` (italics) rather than to `sscit` (spaced small italic capitals). Thanks to Michael Ummels, the missing declarations have now been added, so shape changes in font families that support spaced small capitals work properly. At the same time we took the opportunity to improve the fallbacks for the `sw` (swash) shapes, which are accessible through the commands `\swshape` or `\textsw`. If an `sw` combination is not available, the rules now try to replace `sw` with `it` rather than falling back to `n`. *(github issue 1581)*

Improving the handling of `\label`, `\index`, and `\glossary`

In standard LaTeX, the three commands `\label`, `\index`, and `\glossary` take exactly one mandatory argument, e.g., `\index{`⟨*entry*⟩`}`. In some extension packages, for example index or cleveref, these are all augmented to accept an optional argument and, in the case of `\index`, also a star form. These extensions conflicted with LaTeX's way of disabling these commands within the table of contents and within running headers because they were, in these places, redefined to expect just a mandatory argument and then do nothing. We have now changed this behavior, so that the redefinitions in these places now accept this extended syntax.

(github issue 311)

Tracing lost characters

In LaTeX News 33 [2, p. 63] we announced that `\tracingall` changes `\tracinglostchars` to an error condition. This change has been reverted and `\tracingall` and `\tracingnone` no longer alter `\tracinglostchars`, so its current setting is retained.

The default value used in LaTeX is set so that lost character information is written as a warning to both the log and the terminal. Users may wish to change this into an error, in which case `\tracinglostchars` should be set to 5 (not 3) as this works in all engines.

(github issue 1687)

Always use the extended pool of registers

As the kernel has grown, the use of registers has risen to the point where rolling back to the classical register allocation approach (using only 256 registers) is no longer viable. We have therefore adjusted the rollback

code so that even when requesting a pre-2015 LaTeX, the extended pool remains in use.

A version of `\input` for expansion contexts

The LaTeX definition of `\input` cannot be used in places where TeX is performing expansion: the classic example is at the start of a tabular cell. There are a number of reasons for this: the key ones are that LaTeX's `\input` records which files are read, and provides pre- and post-file hooks. To support the need to carry out file input in expansion contexts, we have now added `\expandableinput`; this skips recording the file name and does not apply any file hooks, but otherwise behaves like `\input`. In particular, it still uses `\input@path` when doing file lookup (contrasting with the behavior of the TeX primitive, which remains internally available for programmers as `\@@input`). *(github issue 514)*

Bug fixes

Avoid problems with page breaks in the middle of `verbatim`-like environments

If a page break occurs in the middle of an environment that sets up special `\catcode` settings, such as a `verbatim` environment, then these settings will remain active when the output routine is building the page. This is normally harmless, because the material contained in the page had been previously tokenized, so that the `\catcode` changes do not matter. However, in certain circumstances tokenization can happen during this page processing: for example, if processing the header involves reading in a file; or if there is a command that uses `\scantokens` so that it retokenizes some material using the verbatim settings.

This has been fixed and LaTeX now explicitly resets the `\catcode` values to their default settings when entering the output routine. Furthermore, packages that make changes to the tokenization beyond what is done by `verbatim` can use the newly introduced hook `build/page/reset` to add their own resets to the output routine processing. This hook is evaluated after LaTeX has done its reset, so it is also possible, if necessary, to overwrite LaTeX's default behavior. *(github issue 600)*

Fix for nested use of `localmathalphabets`

In 2021 we introduced a method to overcome the problem that classic TeX engines (but not the Unicode engines) have only a very limited number of math alphabets available (so they easily got used up by loading math font packages, even if their symbols got used only occasionally). The idea was to avoid allocating all math alphabets globally, but instead to allow a number of them (defined by counter `localmathalphabets`) to vary from one formula to the next. This means that different formulas can make use of different alphabets, so the chances are much higher that the processing of a complex document succeeds. See [2, p. 69] for details.

Unfortunately, the approach we took back then failed in some cases of nested formulas, with the result that the wrong glyphs were used. This has now been corrected. *(github issues 1101 1028)*

docstrip: Error if an `.ins` file is problematic

If the file to be generated had the same name as a preamble declared with `\declarepreamble` then the preamble definition was overwritten, because the macro used to store it got reused to denote the output stream. The same problem happened with postambles declared with `\declarepostamble`. This situation is now detected and an error message is issued. To circumvent the issue, simply use a different macro name for the preamble or postamble. *(github issue 1150)*

Prevent a `cmd` hook from defining an undefined command

Using `\AddToHook{cmd/FOO/...}` when the command `\FOO` was undefined resulted in this command becoming `\relax`. Thus, if used, it no longer raised an "Undefined control sequence" error, but silently did nothing. This behavior has been corrected, and, if the command `\FOO` does not get defined later, e.g., in a package, it now raises an error when it is used in the document. *(github issue 1591)*

Process global options just once per package

In 2022, we introduced key–value (keyval) option processing in the kernel [2, p. 77]. This also added the idea that keys could have scope: load-only, preamble-only and general use. However, we overlooked that an option given globally (in the optional argument to `\documentclass`) would be repeatedly processed and could therefore lead to spurious warnings. This has now been corrected so that now each global option is seen, by the keyval-based option handling system, exactly once per package. *(github issue 1619)*

Make `\label`, `\index`, and `\glossary` truly invisible in running headers

LaTeX has had this bug since its initial implementation: whilst it correctly ignored any `\label`, `\index`, or `\glossary` command that appears in a mark, it neglected correct handling of the spaces around the command. As a result, one could end up with two spaces in the running header where only one should be present. This was detected as part of working on issue 311 and has now been corrected. *(github issue 1638)*

Fully expand the arguments of the declarations `\counterwithin` and `\counterwithout`

The arguments of the commands `\counterwithin` and `\counterwithout` are two counter names that are used

to reset (or not reset) one counter when the other is stepped. They also redefine the representation of that counter, e.g., `\counterwithin{section}{chapter}` would lead to:

```
\renewcommand\thesection
   {\thechapter.\arabic{section}}
```

However, if one of these counters was not named explicitly, as in this example:

```
\newcommand\sectioncounter{section}
\counterwithin{\sectioncounter}{chapter}
```

then we ended up with

```
\renewcommand\thesection
   {\thechapter.\arabic{\sectioncounter}}
```

which could lead to strange results if `\sectioncounter` got changed later on. This has been corrected: these arguments now get fully expanded when the declaration is made. *(github issue 1675)*

Correction in the float placement algorithm

When floats are added to the current or next page, LaTeX makes several tests in order to find an area that can receive the float. One of these tests calculates how much space is already used on the page and how much additional space is needed to place the float in a particular area. This means that it looks not only at the height of the float but also at the values from `\intextsep` (for `h` floats) or `\textfloatsep` and `\floatsep` (for `t` and `b` floats). The resulting space requirement was then stored in an internal variable and compared to the space still available on the page. If the test failed, the algorithm tried the next area.

Unfortunately, the code was reusing the value in that internal variable as the starting point for the next test, without removing the added space for the float separation (`\intextsep`, `\floatsep`, or `\textfloatsep`). Thus the comparison was being made with the wrong value (i.e., too high); therefore the test may have incorrectly concluded that a float doesn't fit, even when it would in fact have fit. This has now been corrected. *(github issue 1645)*

Correct `\CheckEncodingSubset`

In [2, p. 83], and again in [2, p. 100], we suggested that font maintainers should place an appropriate `\DeclareEncodingSubset` declaration in each `ts1`⟨*family*⟩`.fd` file, so that this is tied to the font definition and so will be available whenever a font family is explicitly selected by `\fontfamily{`⟨*name*⟩`}` instead of using a font support package. Unfortunately, however, this method could result in incorrect selection of glyphs if the font encoding subset setting was evaluated before the `.fd` file was loaded (as subset 9 would then be assumed). This has been corrected: `\CheckEncodingSubset` now first loads the `.fd` file when this is necessary. *(github issue 1669)*

Ensuring late `\write` commands aren't lost

If a non-`\immediate` `\write` command is used after the final page has been shipped out then no write will happen because the system waits for a `\shipout` that will never happen. After the last page has been shipped out, we therefore force all further `\write` calls to be `\immediate`: this ensures that they get written even though we are not going to ship out any more pages. This change of behavior is implemented just before the `enddocument/afterlastpage` hook because this hook may contain such `\write` commands. *(github issue 1689)*

Documentation

Clarifying the handling of spaces by `\textcolor`

In contrast to other `\text`-commands such as `\textbf` or `\textrm`, the command `\textcolor` gobbles spaces at the start of its argument. Thus, for example, `Hello\textcolor{red}{␣World}` will produce the output HelloWorld. There are technical as well as compatibility reasons for this, so the behavior will not change. This is now correctly documented. *(github issue 1474)*

Changes to packages in the amsmath category

`\numberwithin` now aliased to `\counterwithin`

The amsmath package offers a `\numberwithin` declaration to specify that a counter should be reset whenever some other counter is stepped. This is a restricted version of the more general kernel command `\counterwithin` which was introduced in the LaTeX kernel in 2018 and extended in 2021 [2, p. 72]. With the current release we have made `\numberwithin` an alias for the more powerful `\counterwithin` and we suggest that the latter command is used in new documents. *(github issue 1673)*

amsmath: Correct equation tag placement

If there is not enough space to place an equation tag on the same line as the equation then amsmath calculates a suitable offset placement for the tag, above (or below) the equation. In the case of the `gather` environment this offset was not reset correctly, so that it also got applied to these tags in any following environment, which gave incorrect placement in certain situations. The fix for this, implemented in 2024/06, was not entirely correct; so this has been changed to do such resetting at the start of every displayed math environment. *(github issue 1289)*

Changes to packages in the graphics category

More accessibility keys in graphicx

The `\includegraphics` command now accepts `actualtext` and `artifact` keys, which by default do nothing but are used by the tagging code to provide an ActualText string or a boolean flag to indicate that the graphic is an artifact. *(github issue 1552)*

Changes to packages in the tools category

multicol: Full support for extended marks

In 2022 we introduced a new mark mechanism for LaTeX [2, p. 76]. However, the initial implementation covered only the standard output routine of LaTeX. As a result the extended marks were not available within columns produced with the multicol package (where they would be especially useful). This limitation has finally been lifted so that the new mechanism is now fully supported by all of our packages. *(github issue 1421)*

array: Improve preamble code for p, m and b

When the preamble of a `tabular` or `array` is being built, the arguments to `p`, `m`, or `b` columns all get expanded several times. This is normally harmless because that argument usually contains just an explicit dimension. However, in a case such as `p{\fpeval{15}pt}` these expansions resulted in an error; this happened because `\fpeval` was expanded a few times, but not often enough to result in a single number. This has now been corrected: these arguments are not expanded at all. This allows for such edge cases and also for the extensions available with the calc package, such as `p{\widthof{AAAAAA}}`. *(github issue 1585)*

array: Fix handling of empty p-cells

If an `\arraystretch` greater than `1` is used, table rows are spread apart by placing suitable struts (invisible rules) into each row, or in case of p-cells into each cell. If such a cell was empty the placement of the strut was not correct so that the cell appeared to be larger than it should have been. This has now been corrected. *(github issue 1730)*

varioref: How to make `\reftextfaceafter`, etc. empty

In the case that one wants to make a command such as `\reftextfaceafter` produce truly nothing, one has to get rid of the space that is automatically placed in front of the command by `\vref`. This can be done by simply defining the command to remove it, e.g.,

```
\renewcommand\reftextfaceafter{\unskip}
```

The varioref package does not test if such strings are empty, because that would require a lot of tests each time `\vref` is used, and it would nearly always find that the text is not empty. However, as shown above, the solution for this uncommon case is simple, and it is now explicitly documented in the package documentation. *(github issue 1622)*

Changes to files in the L3 programming layer

Work on the L3 programming layer continues in parallel with development of the rest of the LaTeX kernel. Of note for developers is that we have integrated more code into the main l3kernel bundle, and therefore into the functionality available automatically in LaTeX. Most notably, l3benchmark, which provides tools for checking code performance, is now part of l3kernel.

We have also extended the color module to recognize the Oklab and Oklch color models; thanks to Markus Kurtz for contributing this code. The Oklab color space (`https://bottosson.github.io/posts/oklab`) is a perceptual color space which is supported by CSS and so also by modern web browsers; Oklch expresses the Oklab color space in cylindrical form.

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 1–41*. June 2025. `https://latex-project.org/news/latex2e-news/ltnews.pdf`

[3] Frank Mittelbach, LaTeX Project Team. *The `ltmarks.dtx` code*. June 2025. `https://latex-project.org/help/documentation/ltmarks-doc.pdf`

[4] LaTeX Project Team. *The LaTeX 2ε Sources*. June 2025. `https://latex-project.org/help/documentation/source2e.pdf`