LaTeX News

Issue 42, November 2025 — DRAFT version for upcoming release (LaTeX release 2025-11-01)

Contents

Introduction

Fans of Douglas Adams will know that this must be a special issue of the LaTeX news; unfortunately it doesn't give us the answer to "Answer to the Ultimate Question of Life, the Universe, and Everything" yet—we still expect a few more issues to reach that point.

However, with this release we have made further progress in the generating tagged and accessible PDF documents from LaTeX and to indicate this we move away from calling it a prototype solution as by now it is and can be used in production workflows (as long as one restricts the documents to already supported packages). This does not mean that `latex-lab` code (where the tagging support currently resides) is no longer under development, but that the user-facing side of the project is by now fairly stable and usable.

Outside of the tagging project we have added new functionality to the font selection support as well as code improvements and additional functionality in a number of other places, for example, supporting value expansion of key values at the time of declaration, which is useful when specifying template instances and in similar places.

As usual, there have been a few bugs to take care of (odd ones for sure this time) and we also decided to finally retire a few packages from the tools collection. More exactly, we suggest that they should no longer be used as there are better possibilities available by now.

News from the Tagged PDF project

Expanding the `\DocumentMetadata` command

In 2022 we introduced the `\DocumentMetadata` with a twofold purpose: to provide a dedicated place for document wide settings and metadata, and to act as a trigger command to identify documents that want to load new code. The latter allows the use of the new, extended interfaces essential for the Tagging Project but also useful without tagging.

Initially, using `\DocumentMetadata` with an empty argument loaded only the PDF management code and a new hyperref driver was used. Since November 2024 `\DocumentMetadata` changes the default encoding from OT1 to T1; and since June 2025 it also changes the default PDF version from 1.7 to 2.0.

Additional code in latex-lab (needed, e.g., for the tagging project) had to be loaded explicitly by using the `testphase` or the new `tagging` key in the argument of `\DocumentMetadata`. Whilst this allowed for the

selective loading and testing of the new code, it also produced problems for classes and packages adapting their code for the tagging project since it was difficult to test which parts of the latex-lab code were active.

In this release we therefore extend `\DocumentMetadata` even further: it will now load directly all the code that one would get when using the `tagging=off` or the `testphase=latest` key.

The values `phase-I`, `phase-II`, `phase-III` of the `testphase` key will no longer load different code variants but only activate tagging. Extra modules not yet incorporated in the `latest` set of modules can still be loaded by using the `testphase` key.

For documents that want to load the PDF management but do not want the new tagging support code we provide a dedicated package. Such documents should replace

```
\DocumentMetadata{pdfversion=1.7,
  pdfstandard=a-3b}
```

by

```
\RequirePackage{pdfmanagement}
\SetKeys[document/metadata]{pdfversion=1.7,
  pdfstandard=a-3b}
```

Checking the compatibility with the Tagging support code

We maintain a database showing the compatibility of classes and packages with the Tagging support code. This data can be viewed online [7]. We have now exported a part of the data into a small package latex-tagging-status and added a key `check-tagging-status` to `\DocumentMetadata`. When used, the status of the packages and the class used by the document will be shown at the end of the log-file.

This is only a rough overview and a debugging aid, not a final report! Using packages that are classified as incompatible or partially incompatible does not mean that the tagging is necessarily broken. For example, hyperref is partially incompatible as the form fields are not properly tagged (this requires the use of the l3pdffield package), but in documents without form fields it is unproblematic. In case of partially-compatible or incompatible packages the full table should be checked as it often contains an explanation what is not yet working.

The package latex-tagging-status will be regularly updated to reflect changes in packages and the status database. Erroneous messages should be reported at the github of the Tagging Project[8]. It is also possible to create a pull request to update or correct the data.

Requiring or testing for the Tagging support code

Classes or packages that are written only for the new code loaded by `\DocumentMetadata` can use the new command `\NeedsDocumentMetadata` at the start of the class or package file. It will produce a suitable error message if the tagging support code has not been loaded.

Classes and packages that want to support both legacy documents and newer documents using `\DocumentMetadata` can now use `\IfDocumentMetadataTF` to test whether the new code has been loaded – eventually in combination with a test of the date of the format. To test whether the PDF management has been loaded, the test `\IfPDFManagementActiveTF` is provided.

Moving paragraph tagging into sockets

Paragraphs in LaTeX can be nested, e.g., you can have a paragraph containing a display quote, which in turn consists of more than one (sub)paragraph, followed by some more text which all belongs to the same outer paragraph.

To model such "semantic paragraphs" LaTeX uses a structure named `text-unit`[1] and uses `text` (role mapped to `P`) only for (portions of) the actual paragraph text.

This is semantically clear and allows processors who care to identify the complete paragraphs by looking for `text-unit` tags. But we got a request for an option to disable the tagging of the "semantic paragraphs", so with this release we moved the relevant tagging code into sockets. The "semantic paragraphs" can now be disabled by assigning the `noop` plug to these sockets:

```
\AssignTaggingSocketPlug{para/semantic/begin}
  {noop}
\AssignTaggingSocketPlug{para/semantic/end}
  {noop}
```

Hooks for `\includegraphics` keys

The three key definitions `alt`, `actualtext` and `artifact` used by `\includegraphics` contain now hooks, named `Gin/alt`, `Gin/actualtext`, and `Gin/artifact`[2]. The first two are hooks with two arguments and get as first argument the purified (with `\text_purify:n`) value of the key which is also used in the PDF and as second argument the raw value. The hooks are processed even if tagging is not activated. With them it is, for example, possible to store the alternative text:

```
\AddToHookWithArguments{Gin/alt}
  {\gdef\myalttext{#2}}
\includegraphics[alt=Hello World]
  {example-image}
The alt text of the graphic was \myalttext.
```

Symbolic structure names

The names of structure elements tags may be taken from the standard PDF namespaces like `Sect`, `H1` or `Figure` but they can also use alternative names provided the

[1] The name is under review and is likely to change in future.

[2] `Gin` refers to the family name used by keys in the graphicx package

latter are role mapped to a standard name. The second approach is useful for three reasons:

- It looks nicer, if, e.g., a bible uses tag names such as `Testament`, `Chapter` or `Book` instead of `Sect`.
- It is possible to formulate additional constraints on such structures in a schema and thus ensure that there is no `Testament` inside a `Book`, something that cannot be done if `Sect` is used everywhere.
- We can provide a uniform LaTeX set of names for tags.

Currently it is difficult for document authors to change tag names as the tagging support code uses either a fixed name or some ad hoc internal variable. We therefore added three commands that offer an interface to declare, use and reassign *symbolic structure names*. `\NewStructureName` takes one argument and declares a *symbolic structure name*. The expandable command `\UseStructureName` takes one argument and allows to use name in a `\tagstructbegin` command. `\AssignStructureRole` allows to assign the symbolic structure name a role.

In the coming months the various tag names in the tagging code will be replaced by such symbolic names. Once the process is finished, document and class authors will have a flexible tool to set up the tag names of their documents.

Normalizing key names for block environments

The display block environments, such as `itemize`, `center`, `verbatim`, etc. have all been reimplemented to become tagging aware a while back. That was also the first time we used the template/instance mechanism to offer consistent layout configuration possibilities (heading commands will be next to use that approach). Doing this meant experimenting with different setups to see what works best. However, as a side effect of these trials and rewrites we ended up with a rather inconsistent set of key names across the different templates, so after the dust had settled it was about time to take a look at the complete set and standardize the key names as much as possible. This task has been largely completed, though some changes are still likely while we develop more templates covering other areas.

These changes are basically transparent for users who are just interested in producing tagged and accessible documents out of the box. However, for people who have started to customize the layout of the environments, using for example `\EditInstance`, the key name changes need to be reflected, accordingly.

Contexts in typesetting

Sometimes document elements should change their layout depending on where they are used, for example, lists might use less vertical space when used inside a footnote or a float. To allow for such designs in a consistent and easy way we introduce the concept of "named contexts": while typesetting the document LaTeX keeps track of a current "primary context" and a current "secondary context". They are changed automatically when certain commands, such as `\footnote` or environments, such as floats, etc. are typeset.

For the primary context, LaTeX distinguishes by default between, typesetting material in the main galley (context name is ⟨*empty*⟩), in a `footnote`, `marginal`, `float`, `caption`, `header`, or `footer`.

The "secondary context" is by default used to identify typesetting in different font sizes and therefore knows knows about `tiny`, `scriptsize`, `footnotesize`, `small`, `large`, `Large`, `LARGE`, `huge`, `Huge`, and ⟨*empty*⟩ (denoting typesetting in `\normalsize`).

In theory it would be possible for commands and environments to query the current context and then alter their behavior, however that would require comparably complex coding. Instead, the main usage for the context is with template instances that are used to define layouts. If a template instance is used via `\UseInstance{`⟨*type*⟩`}{`⟨*inst-name*⟩`}` then this normally results in calling up an instance of type ⟨*type*⟩ with the name ⟨*inst-name*⟩.[3]

However, when the "primary context" and/or the "secondary context" is non-empty then `\UseInstance` searches for an instance that is especially tailored to the current context. This works as follows:

- The string `:`⟨*primary context*⟩`:`⟨*secondary context*⟩ is appended to ⟨*inst-name*⟩ and if that instance exist it is used.[4]
- If not then ⟨*inst-name*⟩`:`⟨*primary context*⟩ is tried next.
- If that doesn't exist either then ⟨*inst-name*⟩ is used as usual.

This means it becomes trivial to alter the behavior of instances if they appear in a special typesetting context. For example, if `itemize-1` is the instance name for first-level itemize lists then one can define another instance named `itemize-1:footnote` to describe a special layout used in footnotes. More details on this can be found in `latex-lab-context.pdf` or by using `texdoc␣latex-lab-context` on the command line.

At this point in time the mechanism is still rather experimental, i.e., we provide and use it in `latex-lab` to gain experience and we also encourage developers to

[3] Such instances are defined with a `\DeclareInstance` or `\DeclareInstanceCopy` declaration; see the documentation in the file `lttemplates-doc.pdf`.

[4] Note that this means that if the ⟨*primary context*⟩ is empty we effectively append `::`⟨*secondary context*⟩.

experiment with it and provide feedback. Details of the implementation are likely to change though.

MathML intent attributes
Two new commands, `\MathMLintent` and `\MathMLarg` are added. They are defined in the format as no-op so they may be added to command definitions in packages. If luamml is enabled to generate MathML, these commands allow *intent* and *arg* attributes to be specified.

A definition such as

```
\newcommand\abs[1]{%
\MathMLintent{absolute-value($x)}
             {{\lvert\MathMLarg{x}{#1}\rvert}}%
}
```

would cause `\abs{y}` to generate

```
<mrow intent="absolute-value($x)">
  <mo>|</mo><mi arg="x">y</mi><mo>|</mo>
</mrow>
```

which will allow Assistive Technology (AT) to correctly read the ambiguous notation $|y|$ as "the absolute value of y" or some similar reading depending on the chosen language.

Correctly handle tagging of math in tabular cells
Mathematical content in tabular cells was not correctly tagged when a MathML representation was automatically generated by LuaTEX. Also tabular preambles of the form `>{$}c<{$}` or `>{\(}c<{\)}` failed. This has been corrected. *(tagging-project issues 973 983)*

New or improved commands

Support separate font families for script fonts
In TEX's math processing separate fonts can be selected for text, script and scriptscript sizes. LATEX's NFSS traditionally uses the same font family at different sizes, handling adjustment needed for making fonts appear better in a script location through the use of optical sizes. This works great for traditional TEX fonts, but for OpenType fonts this leads to issues. OpenType MATH assumes the font in a script location has separate features set and therefore received specific adjustments.

To support this without relying on heuristics based on the font size, a new command `\DeclareMathScriptfontMapping` has been added. It takes 3 pairs of encoding/family arguments to indicate that for the first pair when used as the math main font the second and the third should be used as the script and scriptscript font, respectively. *(github issue 1707)*

Programming support for LATEX's font metafamilies
LATEX knows three main document font families: `\rmfamily` for the document's serifed font family, `\sffamily` for its sans serif font family, and `\ttfamily` for its monospaced font family. In addition, other font families can be used by the user or in a document class or package by explicitly loading them through `\fontfamily{`⟨*name*⟩`}\selectfont`.

In some cases it is helpful to know which of the three metafamilies (if any) is currently used for typesetting, and this information is now made available for programmers in `\@currentmetafamily`. It returns either `rm`, `sf`, `tt`, or `??` (in case none of the metafamilies is currently used).

As a small application of this, the LATEX kernel now also contains `\@restoremetafamily`. If the current metafamily is ⟨*name*⟩ it executes `\`⟨*name*⟩`family`, e.g., `\sffamily`, and that then executes the hook ⟨*name*⟩`family` besides other re-initializations. This can be useful if that hook contains conditional code and the condition has changed and therefore requires re-initialization.

Recovering the argument specifier for document commands
In LATEX News 38 [4] we explained that we had *removed* `\GetDocumentCommandArgSpec` since we felt that it was only required for debugging. However, there are some specialist use cases where access to the argument specification is useful: see, for example, `https://github.com/latex3/latex3/pull/1799`. We have therefore looked again at this area and added a *code* interface `\cmd_arg_spec:N` for accessing the argument specification. The use of a code-level rather than design-level name here reflects the fact that this is an very specialized use case, mainly of interest to package authors.

Code improvements

Ensure that commands without arguments are not `\long`
In its original implementation `\newcommand` or `\renewcommand` always defined commands using `\long\def` even if the commands had no arguments, i.e., in situations where the concept of `\long` made no sense whatsoever.

The issue with that behavior is that commands differing only in their `\long` status are nevertheless considered different when compared with `\ifx` even if there are no arguments to which the `\long` would apply. Thus, after `\renewcommand\rmdefault{lmr}` and `\def\test{lmr}` the test `\ifx\test\rmdefault` would be *false*, but it would be *true* if `\rmdefault` had been defined using `\def` (as many class files do). This made comparing commands without argument rather difficult. We have therefore changed `\newcommand` and friends so that commands without arguments are always defined without using the unnecessary `\long` prefix.

Going forward, this will simplify package and kernel code as the code can reliably assume that such macros

are not \long regardless of whether they are defined by \renewcommand or \def.

There is a small chance that this is a breaking change for some package code (though we don't know of any case), i.e., if the code was deliberately checking against \long\def only—in that case the test now needs to be made against the definition without \long (or against both, which is what the NFSS implementation of the kernel did in the past). *(github issue 571)*

Avoid strange warnings about font substitutions

A font series value such as sbc contains both the weight (sb, i.e. "semibold") and the width (c, i.e. "condensed") of the font. If you want to reset only one of the two to "medium" and keep the other, you can use \fontseries{m?} or \fontseries{?m}: The former switches sbc to c, the latter switches sbc to sb. However, if the resulting series does not exist, you got strange warnings in the past, e.g.:

```
LaTeX Font Warning:
Font shape 'OT1/cmss/c/n' undefined
using 'OT1/cmss/m?/n' instead on input line 7.
LaTeX Font Warning:
Font shape 'OT1/cmss/m?/n' undefined
using 'OT1/cmss/m/n' instead on input line 7.
```

This has now been corrected so that you get a single, more meaningful warning:

```
LaTeX Font Warning:
Font shape 'OT1/cmss/c/n' undefined
using 'OT1/cmss/m/n' instead on input line 7.
```

If the m series does not exist either, you will still get strange warnings, but this should only affect very few fonts. The source file was also tidied up a little on this occasion. *(github issue 1727)*

Improved handling of infinite shrinkage errors

In the June 2024 release [5] we described the improved mark mechanism and the problems we had when working around TEX's "infinite shrinkage error". By now the engines got a new primitive \ignoreprimitiveerror which can be used to turn this error into a warning, when, for example, you do only a trial splitting of a box. This noticeably improves the output in the .log file from

```
! Infinite glue shrinkage found in box being split.
<argument> Infinite shrink error above ignored !
l. ...   }
The box you are \vsplitting contains some
infinitely shrinkable glue, e.g., '\vss' or
'\vskip 0pt minus 1fil'. Such glue doesn't belong
there; but you can safely proceed, since the
offensive shrinkability has been made finite.
```

to a simple

```
ignored error: Infinite glue shrinkage found in
box being split
```

As an important side effect, the return code from the TEX run stays at 0 (unless there are real errors); so in workflows that want to test whether a TEX run ended without errors, you don't get a bogus result because there is no longer an ignored error. *(github issue 1750)*

Allow multiple family names in \ProcessKeyOptions

The ability to process key–value options was introduced into the kernel in the June 2022 release [3], with the command \ProcessKeyOptions carrying out the option assignment. In the original version, this takes an optional argument which can select one key family (namespace) for options. We have now extended this to take a comma list of possible families. *(github issue 1756)*

Control of value expansion in keys

Normally, key–value input is treated "as is", with no expansion of either key names or values. However, there are occasions when the expansion of selected values is useful. We have now extended the key handling for templates (\DeclareInstance, etc.) and for keys created using the L3 programming layer to allow selective expansion. In both cases, the syntax uses a trailing colon and a single letter specifier: these letters are those used in \ExpandArgs or the L3 programming layer. For example, to use the values of the LATEX 2_ε variable \@itemdepth, one could have settings

```
key-a:c =  @itemdepth ,
key-b:v =  @itemdepth
```

This facility will *automatically* be available in any package setup macro using the L3 programming layer, for example siunitx. *(github issue 1801)*

Support word exclusion in case changing

Work on improving automatic case changing over previous releases has continued. We have now added the ability to 'register' words for exclusion from case changing, using \DeclareLowercaseExclusions, \DeclareTitlecaseExclusions and \DeclareUppercaseExclusions.

Automatic insertion of \par tokens

Since 2022 the major TEX engines have a parameter, \partokencontext, that controls whether a \par token is added when TEX is in horizontal mode at the end of \vbox and in similar contexts. This gives more control than the classical behavior where the internal *end paragraph* routine is invoked with no explicit token being added.

This allows the paragraph hooks to detect the end of paragraph even in contexts such as at the end of a \vbox, where traditionally package code has had to be modified to add an explicit \par. This is expected

to improve compatibility of existing packages with the tagging code.

LaTeX now sets this parameter to 2 by default to enable automatic insertion of `\par`. *(github issue 1864)*

Improved access to Generic Hooks

The code to add generic hooks such as `\AddToHook{cmd/somecmd/before}{...}` has been improved so that it is more likely to succeed in cases where the command has been defined using *expl3* syntax. Previously attempts to add hooks to commands would fail if the original definition had used `~` in an `\ExplSyntaxOn` context. *(github issue 1099)*

Bug fixes

Support active characters correctly with `\DeclareRobustCommand`

The mechanism used by `\DeclareRobustCommand` creates an internal command which has a space added to the name of the document one: so `\foo␣` for a command `\foo`. That fails if applied to an active character: unlike normal commands, these have to be exactly one character long. Due to the way the implementation works, to date this would result in redefining `\␣` every time `\DeclareRobustCommand` was used with an active character. This has now been corrected: robust active characters are now created using the engine `\protected` mechanism and do not use an internal auxiliary. They still work in file names and labels to give the character itself. *(github issue 345)*

Avoid a "Corrupted NFSS tables" error

When a character with an accent is typeset, say "ä" or "é", it might be the case that it doesn't exist in the font but has to be constructed from the base character and a standalone accent. If that accent is also not available in the font then LaTeX attempts to find it in a different font, typically one in a different encoding, e.g., `OT1`. Unfortunately, when that involved font substitutions it resulted in a loop generating the mentioned error. This has now been corrected by adding necessary `\DeclareFontSubstitution` statements. *(github issue 1709)*

Changes to packages in the tools category

Updating the status of some components

The tools bundle contains a range of packages with different usage profiles. Some of these were necessary in the transition from LaTeX 2.09 to LaTeX 2ε, while others are very widely used in current documents (for example array). We have therefore marked a small number of packages in tools as *retained only for historical and stability reasons*, and where relevant pointed to more up-to-date alternatives; the list is:

- enumerate: use enumitem instead
- rawfonts: retained as part of LaTeX 2.09 support
- somedefs: retained as part of LaTeX 2.09 support
- theorem: use amsthm instead
- verbatim: use fancyvrb instead

Update to handling page marks in longtable

The longtable package has been updated to correctly adjust the new LaTeX mark structures as each page is output. *(github issue 1814)*

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 1–42*. November 2025. `https://latex-project.org/news/latex2e-news/ltnews.pdf`

[3] LaTeX Project Team. *LaTeX 2ε News 35*. June 2022. `https://latex-project.org/news/latex2e-news/ltnews35.pdf`

[4] LaTeX Project Team. *LaTeX 2ε News 38*. November 2023. `https://latex-project.org/news/latex2e-news/ltnews38.pdf`

[5] LaTeX Project Team. *LaTeX 2ε News 39*. June 2024. `https://latex-project.org/news/latex2e-news/ltnews39.pdf`

[6] LaTeX Project Team. *LaTeX 2ε News 41*. June 2025. `https://latex-project.org/news/latex2e-news/ltnews41.pdf`

[7] LaTeX Project Team. *Tagging Status of LaTeX Packages and Classes*. November 2025. `https://latex3.github.io/tagging-project/tagging-status`

[8] LaTeX Project Team. *The LaTeX Tagged PDF repository*. November 2025. `https://github.com/latex3/tagging-project/issues`