

L^AT_EX News

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

Contents

Introduction	1
News from the Tagged PDF project	1
Expanding the <code>\DocumentMetadata</code> command	1
Requiring or testing for the Tagging support code	1
Moving paragraph tagging into sockets	2
Hooks for <code>\includegraphics</code> keys	2
Symbolic structure names	2
Normalizing key names for block environments	2
Contexts in typesetting	2
MathML intent attributes	2
Correctly handle tagging of math in tabular cells	3
New or improved commands	3
Support separate font families for script fonts	3
Programming support for font meta families	3
Recovering the argument specifier for document commands	3
Code improvements	3
Ensure that commands without arguments are not <code>\long</code>	3
Avoid strange warnings about font substitutions	3
Improved handling of infinite shrinkage errors	4
Allow multiple family names in <code>\ProcessKeyOptions</code>	4
Control of value expansion in keys	4
Support word exclusion in case changing	4
Automatic insertion of <code>\par</code> tokens	4
Bug fixes	4
Support active characters correctly with <code>\DeclareRobustCommand</code>	4
Avoid a “Corrupted NFSS tables” error	5
Changes to packages in the tools category	5
Updating the status of some components	5

Introduction

to write

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 also 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 get also 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}
```

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 begin of the

class or package file. It will produce a suitable error message if the tagging support code has not been loaded.

Classes and package 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 L^AT_EX 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” L^AT_EX uses a structure named `text-unit`¹ and use `text` (role mapped to P) only for (portions of) the actual paragraph text L^AT_EX.

This is semantically clear and allows processors who care to identify the complete paragraphs by looking for `text-unit` tags. But we got also 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 keys 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 (with `\text_purify:n`) purified 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\myaltttext{#2}}
\includegraphics[alt=Hello World]
  {example-image}
The alt text of the graphic was \myaltttext.
```

¹The 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

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 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 like `Testament` or `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 to write

Contexts in typesetting

to write if latex-lab-context makes it to the next stage before 2025/11/01

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 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. L^AT_EX’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 L^AT_EX’s font metafamilies

L^AT_EX knows three main document font families: `\rmfamily` for the document’s serified 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 L^AT_EX 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 L^AT_EX 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 !
1. ... }
```

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\epsilon
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.

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

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 L^AT_EX 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 L^AT_EX 2.09 to L^AT_EX 2_ε, 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 L^AT_EX 2.09 support
- `somedefs`: retained as part of L^AT_EX 2.09 support
- `theorem`: use `amsthm` instead
- `verbatim`: use `fancyvrb` instead

References

- [1] Leslie Lamport. *L^AT_EX: 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] L^AT_EX Project Team. *L^AT_EX 2_ε News 1–42*. November 2025. <https://latex-project.org/news/latex2e-news/ltnews.pdf>
- [3] L^AT_EX Project Team. *L^AT_EX 2_ε News 35*. June 2022. <https://latex-project.org/news/latex2e-news/ltnews35.pdf>
- [4] L^AT_EX Project Team. *L^AT_EX 2_ε News 38*. November 2023. <https://latex-project.org/news/latex2e-news/ltnews38.pdf>

- [5] L^AT_EX Project Team. *L^AT_EX 2_ε News 39*. June 2024. <https://latex-project.org/news/latex2e-news/ltnews39.pdf>
- [6] L^AT_EX Project Team. *L^AT_EX 2_ε News 41*. June 2025. <https://latex-project.org/news/latex2e-news/ltnews41.pdf>