LATEX News
Issue 31, February 2020

Contents

Experiences with the LATEX -dev formats 1
Concerning this release . . . (LuaLATEX engine) 1
Improved load-times for expl3 1

Improvements to LATEX font selection: NFSS 2
Extending the shape management in NFSS . . . 2
Extending the font series management in NFSS 2
Font series defaults per document family . . . 3
Handling of nested emphasis . . . . . . . . . 3
Providing font family substitutions . . . . 3
Providing all text companion symbols by default 3
New alias size function for use in .fd files . . 4
Suppress unnecessary font substitution warnings 4

Other changes to the LATEX kernel 4
UTF-8 characters in package descriptions . . . 4
Fix inconsistent hook setting when loading packages . . . . . . . . 4
Avoid spurious warning if LY1 is made the default encoding . . . . 4
Ensure that \textbackslash\ remains robust . . . . . . . . . . . . . . . 4
Make math delimiters robust in a different way 4
Allow more write streams with filecontents in LuaLATEX . . . . . . 4

Changes to packages in the graphics category 4
Make color/graphics user-level commands robust 4

Changes to packages in the tools category 5
Fixed column depth in boxed multicols . . . 5
Ensure that multicols does not lose text . . . 5
Allow spaces in \hline arguments . . . . . . . . . . 5

LATEX requirements on engine primitives 5

Experiences with the LATEX -dev formats 5
As reported in the previous LATEX News, we have made a pre-release version of the LATEX kernel available as LATEX-dev. Overall, the approach of having an explicit testing release has been positive: it is now readily available in TeX systems and is getting real use beyond the team.

The current release has been tested by a number of people, and we have had valuable feedback on a range of the new ideas. This has allowed us to fix issues in several of the new features, as described below.

We wish to thank all the dedicated users who have been trying out the development formats, and we encourage others to do so. Pre-testing in this way does mean that, for the vast majority of users, problems are solved before they even appear!

Concerning this release . . . (LuaLATEX engine)

The new LuaHBTeX engine is LuaLATEX with an embedded HarfBuzz library. HarfBuzz can be used by setting a suitable renderer in the font declaration. A basic interface for that is provided by fontspec. This additional font renderer will greatly improve the shaping of various scripts when using LuaLATEX, many of which are currently handled correctly only by XeLATEX, which always uses HarfBuzz.

To simplify testing of the new engine, binaries have already been added to MiKTeX and TeX Live 2019 and both distributions have already now changed the LuaLATEX-dev format to use it.

Going forward, LuaLATEX (and LuaLATEX-dev) will both use the LuaHBTeX engine. The timing of the switch to the LuaHBTeX engine depends on the distribution you use (for TeX Live this will be with TeX Live 2020).

Improved load-times for expl3

The LATEX3 programming layer, expl3, has over the past decade moved from being largely experimental to broadly stable. It is now used in a significant number of third-party packages, most notably xparse, for defining interfaces in cases where no expl3 code is “visible.” In addition, most LATEX documents compiled using XeLATEX or LuaLATEX load fontspec, which is written using expl3.

The expl3 layer contains a non-trivial number of macros, and when used with the XeLATEX and LuaLATEX engines, it also loads a large body of Unicode data. This means that even on a fast computer, there is a relatively large load time when using expl3.

For this release, the team have made adjustments in the LATEX2ε kernel to pre-load a significant portion of expl3 when the format is built. This is transparent to the user, other than the significant decrease in document processing time: there will be no “pause” whilst loading the Unicode data files. Loading expl3 in documents and LATEX News, and the LATEX software, are brought to you by the LATEX3 Project Team; Copyright 2020, all rights reserved.
packages can continue to be done as usual; eventually, it
will be possible to omit
\RequirePackage{expl3}
entirely but, to support older formats, this is still
recommended at present.

Improvements to \LaTeX’s font selection
mechanism (NFSS)

Extending the shape management in NFSS

Over time, more and more fonts have become available
for use with \LaTeX. Many such font families offer
additional shapes such as small caps italic (scit), small
caps slanted (acs1) or swash (sw). By using \textfontshape
those shapes can be explicitly selected. For the swash
shapes there is also \textswashshape and \texttextshape available.

In the original font selection implementation a request
to select a new shape always overrode the current shape.
With the 2020 release of \LaTeX this has changed and
\textfontshape can now be used to combine small capitals
with italic, slanted or swash letters, either by explicitly
asking for scit, etc., or by asking for italics when
typesetting already in small caps, and so forth.

Using \textupshape will still change italics or slanted
back to an upright shape but will not any longer alter
the small caps setting. To change small capitals back
to upper/lower case you can now use \textulcshape (or
\textulcsfshape) which in turn will not change the font
with respect to italics, slanted or swash. There is one
exception: for compatibility reasons \textupshape will
change small capitals back to upright (n shape), if the
current shape is sc. This is done so that something like
acshape... \textupshape continues to work as before, but
we suggest that you don’t use that deprecated method
in new documents.

Finally, if you want to reset the shape back to normal
you can use \textnormalshape which is a shorthand for
\textupshape\textulcshape.

The way that shapes combine with each other is not
hardwired; it is customizabe and extensible if there is
ever a need for this. The mappings are defined through
\textDeclarenumFontShapeChangeRule and the details for
developers are documented in source2e.pdf.

The ideas for this interface extension have been
pioneered in fontspec by Will Robertson for Unicode
eengines, and in fontaxes by Andreas Bühmann and
Michael Ummels for pdflatex; they are by now used in
many font support packages.

Extending the font series management in NFSS

Many of the newer font families also come provided
with additional weights (thin, semi-bold, ultra-bold,
etc.) or several running widths, such as condensed or
extra-condensed. In some cases the number of different
values for series (weight plus width) is really impressive:
for example, Noto Sans offers 36 fonts, from ultra-light
extra condensed to ultra-bold medium width.

Already in its original design, NFSS supported 9
weight levels, from ultra-light (ul) to ultra-bold (ub),
and also 9 width levels, from ultra-condensed (uc) to
ultra-expanded (ux): more than enough, even for a font
family like Noto Sans. Unfortunately, some font support
packages nevertheless invented their own names, so in
recent years you have been able to find all kinds of
non-standard series names (k, i, j and others), making
it impossible to combine different fonts successfully
using the standard NFSS mechanisms.

Over the course of the last year a small number
of individuals, notably, Bob Tennent, Michael Sharpe
and Marc Fennina, have worked hard to bring this
unsatisfactory situation back under control; so today
we are happy to report that the internal font support
files for more than a hundred font families are all
back to following the standard NFSS conventions.
Combining them is now again rather nice and easy, and
from a technical perspective they can now be easily
matched; but, of course, there is still the task of choosing
combinations that visually work well together.

In the original font selection implementation, a request
to select a new series always overrode the current one.
This was reasonable because there were nearly no fonts
available that offered anything other than a medium or
a bold series. Now that this has changed and families
such as Noto Sans are available, combining weight and
width into a single attribute is no longer appropriate.
With the 2020 release of \LaTeX, the management of
series therefore changed to allow independent settings of
the weight and the width attributes of the series.

For most users this change will be largely transparent
as \LaTeX offers only \textbf{or} \textbf{series to select a
bolder face (and \textmd and \textmdseries to return
to a medium series): there is no high-level command
for selecting a condensed face, etc. However, using the
NFSS low-level interface it is now possible to ask for, say,
\textfontseries{c}\textselectfont to get a condensed face
(suitable for a marginal note) and that would still allow
the use of \textbf inside the note, which would select
a bold-condensed face (and not a rather odd-looking
bold-extended face in the middle of condensed type).

The expectation is that this functionality will be
used largely by class and package designers but, given
that the low-level NFSS commands are usable on the
document level and that they are not really difficult to
apply, there are probably also a number of users who
will enjoy using these new possibilities that bring \LaTeX
back into the premier league for font usage.

The ways in which the different series values
combine with each other is not hardwired but is again
customizable and extensible. The mappings are defined
through \DeclareFontSeriesChangeRule and the details for developers are documented in source2e.pdf.

Font series defaults per document family
With additional weights and widths now being available in many font families, it is more likely that somebody will want to match, say, a medium weight serif family with a semi-light sans serif family, or that with one family one wants to use the bold-extended face when \textbf is used, while with another it should be bold (not extended) or semibold, etc.

In the past this kind of extension was provided by Bob Tennent’s mweights package, which has been used in many font support packages. With the 2020 release of \LaTeX{} this feature is now available out of the box. In addition we also offer a document-level interface to adjust the behavior of the high-level series commands \textit, \textmd, and of their declaration forms \textbfseries and \textmdseries, so that they can have different effects for the serif, sans serif and typewriter families used in a document.

For example, specifying
\begin{verbatim}
\DeclareFontSeriesDefault[rm]{bf}{sb}
\DeclareFontSeriesDefault[tt]{md}{lc}
\end{verbatim}
in the document preamble would result in \textbf producing semi-bold (sb) when typesetting in a roman typeface. The second line says that the typewriter default face (i.e., the medium series md) should be a light-condensed face. The optional argument here can be either rm, sf or tt to indicate one of the three main font families in a document; if omitted you will change the overall document default instead. In the first mandatory argument you specify either md or bf and the second mandatory argument then gives the desired series value in NFSS nomenclature.

Handling of nested emphasis
In previous releases of \LaTeX{}, nested \textit commands automatically alternated between italics and upright. This mechanism has now been generalised so that you can now specify for arbitrary nesting levels how emphasis should be handled.

The declaration \Decla
tEmphSequence expects a comma separated list of font declarations corresponding to increasing levels of emphasis. For example,
\begin{verbatim}
\Decla
tEmphSequence{\itshape,%
\upshape\ucshape,\itshape}
\end{verbatim}
uses italics for the first, small capitals for the second, and italic small capitals for the third level (provided you use a font that supports these shapes). If there are more nesting levels than provided, \LaTeX{} uses the declarations stored in \texttt{\reset} (by default \ucshape\upshape) for the next level and then restarts the list.

The mechanism tries to be “smart” by verifying that the given declarations actually alter the current font. If not, it continues and tries the next level—the assumption being that there was already a manual font change in the document to the font that is now supposed to be used for emphasis. Of course, this only works if the declarations in the list’s entries actually change the font and not, for example, just the color. In such a scenario one has to add \emforce to the entry, which directs the mechanism to use the entry, even if the font attributes appear to be unchanged.

Providing font family substitutions
Given that \pdftex can only handle fonts with up to 256 glyphs, a single font encoding can only support a few languages. The T1 encoding, for example, does support many Latin-based scripts, but if you want to write in Greek or Cyrillic then you will need to switch encodings to LGR or T2A. Given that not every font family offers glyphs in such encodings, you may end up with some default family (e.g., Computer Modern) that doesn’t blend in well with the chosen document font. For such cases NFSS now offers \Decla
reFontFamilySubstitutio
, for example:
\begin{verbatim}
\Decla
reFontFamilySubstitutio
{LGR}
{Montserrat-LF}{IBMPlexSans-TLF}
\end{verbatim}
tells \LaTeX{} that if you are typesetting in the sans serif font Montserrat-LF and the Greek encoding LGR is asked for, then \LaTeX{} should use IBMPlexSans-TLF to fulfill the encoding request.

The code is based on ideas from the \texttt{substitutfont} package by Günter Milde, but the implementation is different.

Providing all text companion symbols by default
The text companion encoding TS1 was originally not available by default, but only when the \texttt{textcomp} package was loaded. The main reason for this was limited availability of fonts with this encoding other than Computer Modern; another was the memory restrictions back in the nineties. These days neither limitation remains, so with the 2020 release all the symbols provided with the \texttt{textcomp} package are available out of the box.

Furthermore, an intelligent substitution mechanism has been implemented so that glyphs missing in some fonts are automatically substituted with default glyphs that are sans serif if you typeset in \texttt{textsf} and monospaced if you typeset using \texttt{texttt}. In the past they were always taken from Computer Modern Roman if substitution was necessary.

This is most noticeable with \texttt{oldstylenums} which are now taken from TS1 so that you no longer get 1234 but 1234 when typesetting in sans serif fonts and 1234 when using typewriter fonts.
If there ever is a need to use the original (inferior) definition, then that remains available as \textbackslash{legacyoldstylelnums} and to fully revert to the old behavior there is also \textbackslash{UseLegacyTextSymbols}. The latter declaration reverts \textbackslash{oldstylelnums} and also changes the footnote symbols, such as \textbackslash{textdagger}, \textbackslash{textparagraph}, etc., to pick up their glyphs from the math fonts instead of the current text font (this means they always keep the same shape and do not nicely blend in with the text font).

With the text companion symbols as part of the kernel, it is normally no longer necessary to load the textcomp package, but for backwards compatibility this package will remain available. There is, however, one use case where it remains useful: if you load the package with the option \textbackslash{error} or \textbackslash{warn} then substitutions will change their behavior and result in a \LaTeX error or a \LaTeX warning (on the terminal), respectively. Without the package the substitution information only appears in the .log file. If you use the option \textbackslash{quiet}, then even the information in the transcript is suppressed (which is not really recommended).

\textbf{New(alias) size function for use in .fd files}

Most of the newer fonts supported in \TeX have been set up with the autoinst tool by Marc Penninga. In the past, this program set up each font using the face name chosen by that font’s designer, e.g., “\textit{regular}”, “\textit{bold}”, etc. These face names were then mapped by substitution to the standard NFSS series names, i.e., “\textit{m}” or “\textit{b}”. As a result one got unnecessary substitution warnings such as “Font T1/abc/bold/n not found, using T1/abc/b/n instead”.

We now provide a new NFSS size function, alias, that can and will be used by autoinst in the future. It provides the same functionality as the subst function but is less vocal about its actions, so that only significant font substitutions show up as warnings.

\textbf{Suppress unnecessary font substitution warnings}

Many sans serif fonts do not have real italics but usually only oblique/slanted shapes, so the substitution of slanted for italics is natural and in fact many designers talk about italic sans serif faces even if in reality they are oblique.

With nearly all sans serif font families, the \LaTeX support files therefore silently substitute slanted if you ask for \textit{\textit{shape}} or \textit{\textit{textit}}. This is also true for Computer Modern in T1 encoding but in OT1 you got a warning on the terminal even though there is nothing you can do about it. This has now been changed to an information message only, written to the .log file. (github issue 172)

\textbf{Other changes to the \LaTeX kernel}

\textbf{UTF-8 characters in package descriptions}

In 2018 we made UTF-8 the default input encoding for \LaTeX but we overlooked the case of non-ASCII characters in the short package descriptions used in declarations, e.g., in the optional argument to \textbackslash{ProvidesPackage}. They worked (sometimes) before, but the switch to UTF-8 made them always generate an error. This has been corrected. (github issue 52)

\textbf{Fix inconsistent hook setting when loading packages}

As part of loading a package, the command \textbackslash{package.sty-h@@k} gets defined. However, attempting to load a package a second time resulted in this hook becoming undefined again. Now the hook remains defined so that extra loading attempts do not change the state of \LaTeX (relevant only to package developers). (github issue 198)

\textbf{Avoid spurious warning if \textbackslash{LY1} is made the default encoding}

Making \textbackslash{LY1} the default encoding, as is done by some font support packages, gave a spurious warning even if \textbackslash{\textbackslash{rmdefault}} was changed first. This was corrected. (github issue 199)

\textbf{Ensure that \textbackslash{\textbackslash{}remains robust}}

In the last release we made most document-level commands robust, but \textbackslash{\textbackslash{}} became fragile again whenever \textbackslash{\textbackslash{raggedright}} or similar typesetting was used. This has been fixed. (github issue 203)

\textbf{Make math delimiters robust in a different way}

Making math delimiters robust caused an issue in some situations. This has been corrected. This also involved a correction to amsmath. (github issue 251)

\textbf{Allow more write streams with filecontents in Lua\TeX}

Most \TeX engines only support a maximum of sixteen concurrently open write streams, and when those have been used up, then filecontents or any other code trying to open one will fail. In Lua\TeX more write streams are available and those can also now be utilised. (github issue 238)

\textbf{Allow spaces in filecontents option list}

Leaving spaces or newlines in the option list prevented the options from being correctly recognized. This has been corrected. (github issue 256)

\textbf{New reverse\textbackslash{\textbackslash{list}} Lua callback type}

A new callback type, reverse\textbackslash{\textbackslash{list}}, was added: post\textbackslash{\textbackslash{mlist_to_hlist_filter}} and post\textbackslash{\textbackslash{linebreak_filter}} are now of this type.
Changes to packages in the graphics category

Make color & graphics user-level commands robust
Some of the user-level commands in color, graphics and
graphics, such as \textcolor or \includegraphics,
were still fragile so didn’t work in moving arguments.
All of these are now robust. \texttt{github issue 208)

Changes to packages in the tools category

Fixed column depth in boxed multicols
The \texttt{multicols} environment was setting \texttt{\maxdepth}
when splitting boxes; but, due to the way the internal
interfaces of \LaTeX{} are designed, it should have
used \texttt{\maxdepth} instead. As a result, balanced
boxed multicols sometimes ended up having different
heights even if they had exactly the same content.
\texttt{github issue 190)

Ensure that multicols does not lose text
The \texttt{multicols} environment needs a set of consecutively
numbered boxes to collect column material. The way
those got allocated could result in disaster if other
packages allocated most boxes below box 255 (which
\TeX{} always uses for the output page). In the original
implementation that problem was avoided because
one could only allocate box numbers below 255, but
nowadays the \LaTeX{} allocation routine allows allocating
box numbers both below and above 255. So the
assumption that when asking for, say, 20 boxes you
always get a consecutive sequence of 20 box register
numbers became no longer true: some of the column
material could end up in box 255, where it would get
overwritten. This has now been corrected by allocating
all necessary boxes with numbers above 255 whenever
there aren’t enough lower-numbered registers available.
\texttt{github issue 237)

Allow spaces in \texttt{\hhline} arguments
The \texttt{\hhline} command, which allows the specification
of rule segments in \texttt{\tabular} environments, now
allows (but ignores) spaces between its tokens: so
\texttt{\hhline{= : =}} is now allowed and is equivalent to
\texttt{\hhline{:::=}}. This matches similar token arguments
in \LaTeX{} such as the \texttt{\[h\;t\;p\]} argument on floats. A
similar change has been made to the extended \texttt{\hhline}
command in the \texttt{colorstbl} package. \texttt{github issue 242)

\LaTeX{} requirements on engine primitives
Since the finalization of \texttt{\epsilon-T\TeX{}} in 1999, a number of
additional ‘utility’ primitives have been added to
\texttt{pdftex}. Several of these are broadly useful and have
been required by \texttt{exp3} for some time, most notably
\texttt{\pdfstrcmp}. Over time, a common set of these ‘post-\texttt{\epsilon-}\texttt{T\TeX{}}’
 primitives have been incorporated into \texttt{X}\texttt{T}\texttt{E}\texttt{X} and
\texttt{(u)p-T\TeX{}}; they were already available in \texttt{Lu\TeX{}}.

A number of these additional primitives are needed to
support new or improved functionality in \texttt{\LaTeX{}}. This is
seen for example in the improved UTF-8 handling, which
uses \texttt{\if\ifcsname}. The following primitive functionality
(\texttt{which in Lu\TeX{}} may be achieved using Lua code) will
therefore be \texttt{required} by the \texttt{\LaTeX{}} kernel and core
packages from the start of 2021:

\begin{itemize}
  \item \texttt{\expanded}
  \item \texttt{\if\ifcsname}
  \item \texttt{\ifpdfprimitive}
  \item \texttt{\pdfcreationdate}
  \item \texttt{\pdffilemoddate}
  \item \texttt{\pdffilesize}
  \item \texttt{\pdffilemoddate}
  \item \texttt{\pdffilesize}
  \item \texttt{\pdffivedigit}
  \item \texttt{\pdffivedigit}
\end{itemize}

For ease of reference, these primitives will be referred to
as the ‘\texttt{pdff} \texttt{T} \texttt{E} \texttt{X} utilities’. With the exception of
\texttt{\expanded}, these have been present in \texttt{pdff} \texttt{T} \texttt{E} \texttt{X} since
the release of version 1.40.0 in 2007; \texttt{\expanded} was
added for \texttt{T} \texttt{E} \texttt{X} Live 2019. Similarly, the full set of these
utility primitives has been available in \texttt{X} \texttt{T} \texttt{E} \texttt{X}
from the 2019 \texttt{T} \texttt{E} \texttt{X} Live release, and has always been available in
\texttt{Lu\TeX{}} (some by Lua emulation). The Japanese \texttt{pT} \texttt{E} \texttt{X}
and \texttt{upT} \texttt{E} \texttt{X} gained all of the above (except \texttt{\if\ifcsname})
for \texttt{T} \texttt{E} \texttt{X} Live 2019 and will both have that primitive also
from the 2020 release onward.

At the same time, engines which are fully Unicode-
capable must provide the following three primitives:

\begin{itemize}
  \item \texttt{\Uchar}
  \item \texttt{\Ucharcat}
  \item \texttt{\Umathcode}
\end{itemize}

Note that it has become standard practice to check for
Unicode-aware engines by using the existence of the
\texttt{\Umathcode} primitive. As such, this is already a
requirement: engines lacking these primitives cannot use
Unicode features of the \texttt{\LaTeX{}} 2\epsilon kernel or \texttt{exp3}. Note
also that up\texttt{T} \texttt{E} \texttt{X} can handle Unicode but it is not classed as
a Unicode engine by the base \texttt{\LaTeX{}} code.

References

[1] Frank Mittelbach: The \texttt{\B} \texttt{T}\texttt{E}\texttt{X} release workflow and the
\texttt{\B} \texttt{T}\texttt{E}\texttt{X} \texttt{dev formats}. In: TUGboat, 40\#2, 2019.
\url{https://latex-project.org/publications/}

[2] \texttt{\B} \texttt{T}\texttt{E}\texttt{X} Project Team: \texttt{\B} \texttt{T}\texttt{E}\texttt{X} 2\epsilon font selection.
\url{https://latex-project.org/help/documentation/famguide.pdf}

[3] \texttt{\B} \texttt{T}\texttt{E}\texttt{X} documentation on the \texttt{\B} \texttt{T}\texttt{E}\texttt{X} Project Website,
\url{https://latex-project.org/help/documentation/}