Experiences with the LaTeX -dev formats
Concerning this release ... (LuaLaTeX engine)
Improved load-times for expl3

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

Other changes to the LaTeX kernel
UTF-8 characters in package descriptions
Fix inconsistent hook setting when loading packages
Avoid spurious warning if LY1 is made the default encoding
Ensure that \textbackslash slash remains robust
Make math delimiters robust in a different way
Allow more write streams with filecontents in LuaTEX
Allow spaces in filecontents option list
New reverselist Lua callback type

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

Changes to packages in the tools category
Fixed column depth in boxed multicol
Ensure that multicol does not lose text
Allow spaces in \hhline arguments

LaTeX requirements on engine primitives

Experiences with the LaTeX -dev formats
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 LuaTEX 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 fonts. This
additional font renderer will greatly improve the shaping
of various scripts when using LuaTEX, many of which
are currently handled correctly only by Xe\TeX, 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 Xe\TeX
or LuaTEX load fonts, which is written using expl3.

The expl3 layer contains a non-trivial number of
macros, and when used with the Xe\TeX and Lua\TeX
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 LaTeX 2e 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

LaTeX News, and the LaTeX software, are brought to you by the LaTeX Project Team; Copyright 2020, all rights reserved.
the Unicode data files. Loading expl3 in documents and 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 (scsl) or swash (sw). By using `\fontshape` those shapes can be explicitly selected. For the swash shapes there is also `\textslshape` and `\textuclshape` 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 `\fontshape` 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 `\upshape` 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 `\textulshape` (or `\textuclshape`) which in turn will not change the font with respect to italics, slanted or swash. There is one exception: for compatibility reasons `\upshape` will change small capitals back to upright (n shape), if the current shape is sc. This is done so that something like `\textscshape . . . \upshape` 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 `\normalshape` which is a shorthand for `\upshape\textulshape`.

The way that shapes combine with each other is not hardwired; it is customizable and extensible if there is ever a need for this. The mappings are defined through `\DeclareFontShapeChangeRule` 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 engines, and in *fontaxes* by Andreas Bühmann and Michael Ummels for pdf\LaTeX{}; 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 Penninga, 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{bfseries}` or `\textit{itshape}` 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, `\fontseries{c}\selectfont` to get a condensed face (suitable for a marginal note) and that would still allow the use of `\textbf{bfseries}` 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 \textbf, \textmd, and of their declaration forms \bfseries and \mdseries, 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 \texttt{rm}, \textsf{sf} or \texttt{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 \texttt{md} or \texttt{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 \DeclareEmphSequence expects a comma separated list of font declarations corresponding to increasing levels of emphasis. For example,
\begin{verbatim}
  \DeclareEmphSequence{\itshape,\%
  \upshape\scshape,\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{\emreset} (by default \texttt{\ulcsshape\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 \texttt{\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 \DeclareFontFamilySubstitution, for example:
\begin{verbatim}
  \DeclareFontFamilySubstitution{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 substitutefont 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 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 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 \textsf and monospaced if you typeset using \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 \legacyoldstylenums; and to fully revert to the old behavior there is also \UseLegacyTextSymbols. The latter declaration reverts \oldstylenums and also changes the footnote symbols, such as \textdagger, \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 error or 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 quiet, then even the information in the transcript is suppressed (which is not really recommended).

**New alias size function for use in .tex 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., “regular”, “bold”, etc. These face names were then mapped by substitution to the standard NFSS series names, i.e., “m” or “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.

**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 \textshape or \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.

**Other changes to the \LaTeX{} kernel**

**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 \ProvidesPackage. They worked (sometimes) before, but the switch to UTF-8 made them always generate an error. This has been corrected.  

**Fix inconsistent hook setting when loading packages**

As part of loading a package, the command \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).

Avoid spurious warning if LY1 is made the default encoding

Making LY1 the default encoding, as is done by some font support packages, gave a spurious warning even if \rmdefault was changed first. This was corrected. 

**Ensure that \textbackslash slash remains robust**

In the last release we made most document-level commands robust, but \textbackslash slash became fragile again whenever \raggedright or similar typesetting was used. This has been fixed.

**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.

**Allow more write streams with filecontents in \LuaTeX**

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 \LuaTeX{} more write streams are available and those can also now be utilised.

**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.

**New reverselist Lua callback type**

A new callback type, reverselist, was added:

post_mlist_to_hlist_filter and post_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 graphicx, such as \textcolor or \includegraphics, were still fragile so didn’t work in moving arguments. All of these are now robust. *(github issue 208)*

Changes to packages in the tools category

Fixed column depth in boxed multicols

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

Ensure that multicols does not lose text

The 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. *(github issue 237)*

Allow spaces in \hhline arguments

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

\LaTeX{} requirements on engine primitives

Since the finalization of \vTeX{} in 1999, a number of additional ‘utility’ primitives have been added to pdf\TeX{}. Several of these are broadly useful and have been required by expl3 for some time, most notably pdfstrcmp. Over time, a common set of these ‘post-v\TeX{}’ primitives have been incorporated into X\TeX{} and (u)p\TeX{}; they were already available in Lua\TeX{}.

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

- \expanded
- ifincsname
- ifpdfprimitive
- pdfcreationdate
- pdfelapsedtime
- pdffiledump
- pdffilemoddate
- pdffilesize
- pdflastxpos
- pdflast ypos
- pdfmdfivesum
- pdfnormaldeviate
- pdfpageheight
- pdfpagewidth
- pdfprimitiv e
- pdfrandomseed
- pdfreadcrypt
- pdfshellescape
- pdfstrcmp
- pdfuniformdeviate

For ease of reference, these primitives will be referred to as the ‘pdf\TeX{} utilities’. With the exception of expanded, these have been present in pdf\TeX{} since the release of version 1.40.0 in 2007; expanded was added for \TeX{} Live 2019. Similarly, the full set of these utility primitives has been available in X\TeX{} from the 2019 \TeX{} Live release, and has always been available in Lua\TeX{} (some by Lua emulation). The Japanese p\TeX{} and up\TeX{} gained all of the above (except ifincsname) for \TeX{} 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:

- \Uchar
- \Ucharcat
- \Umathcode

Note that it has become standard practice to check for Unicode-aware engines by using the existence of the Umathcode primitive. As such, this is already a requirement: engines lacking these primitives cannot use Unicode features of the \LaTeX{} 2e kernel or expl3. Note also that up\TeX{} can handle Unicode but it is not classed as a Unicode engine by the base \LaTeX{} code.

References

[3] B\TeX{} documentation on the B\TeX{} Project Website. https://latex-project.org/help/documentation/