LaTeX3: from local to global
A brief history and recent developments

Will Robertson and Frank Mittelbach

This is all Will's fault

```
From: Frank Mittelbach
Date: 27 June 2012

Will wrote:
> I'm still marking exams today, but I can put time
> aside tomorrow to produce some solid material.
> Because this is largely my fault, and to take the
> worry off you here, would you like to assume that
> I'll produce the whole talk?

I sure would :-)
```

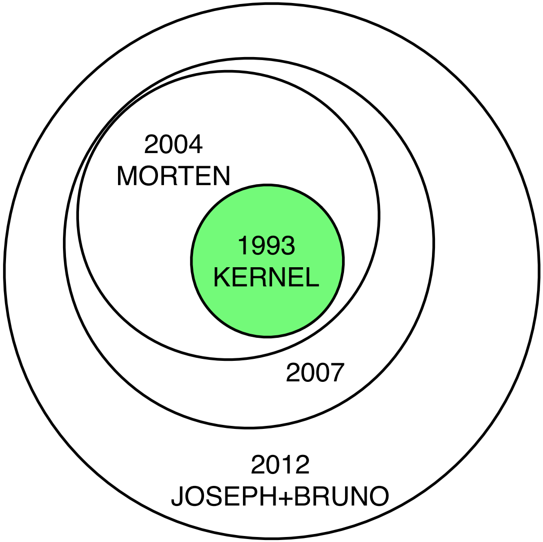
Outline

expl3 timeline

1991	Original kernel
1998	‘Modern’ beginning
2004	Morten
2008	Will
2009	Joseph
2011	Bruno

Paulo Roberto Massa Cereda

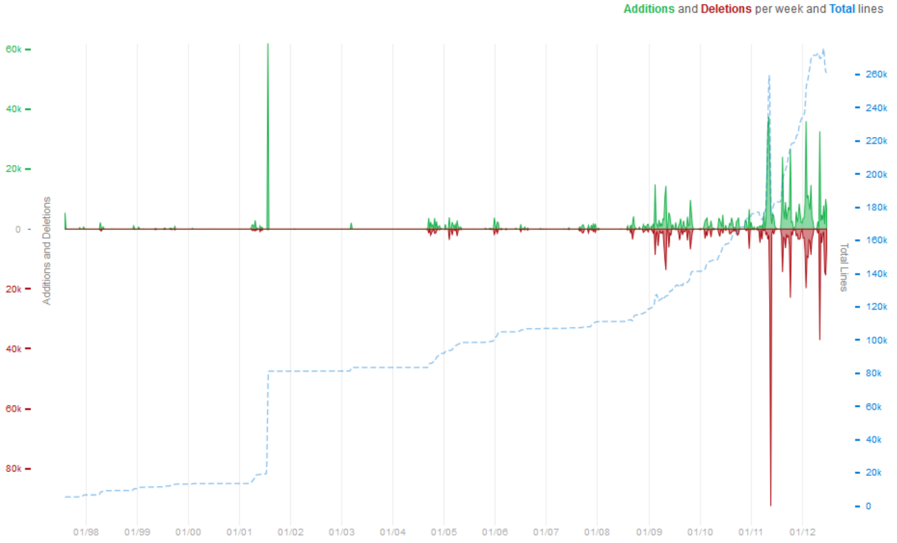
expl3 history



News items

2009/1	Test suites and reconsidering interface.
2009/2	Revamp naming; TL2009. Arg. spec. and l3msg.
2010/1	Rewrite of xparse and xtemplate.
2010/2	siunitx and fontspec. xhead; xcoffin; l3fp; I/O.
2011/1	LPPL OSI; Stack Exchange. l3fp; l3coffin.
2011/2	Big Bang; xgalley.
2012/1	Native drivers; l3regex. LDB.
2012/2	l3fp (exp.); @@.

Code frequency



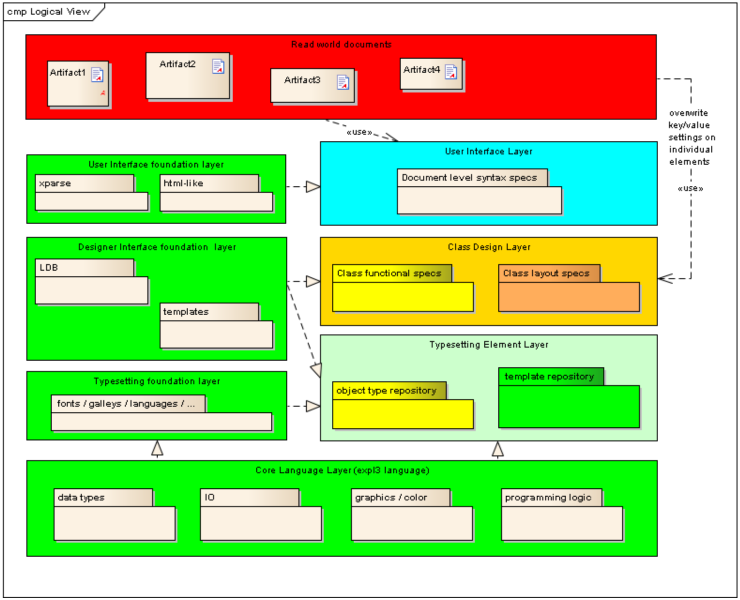
Aspects to LaTeX3

Multiple overlapping concepts.

- Programming layer expl3:
  fontspec/siunitx users (etc.) are ‘using’ LaTeX3
- l3in2e:
  xparse / xtemplate / xcoffins
- Typesetting research:
  xgalley / xor
- Kernel:
  All of the above

LaTeX3 is not monolithic.

Architecture



Outline

expl3 is the foundation

- Supports pdfTEX and XƎTEX and LuaTEX.
  All three are in active use.
- Abstraction for general programming concepts;
  avoid having to remember ‘special tricks’
  and reinvent the wheel.
- Conceived in the 90s; too slow then.
  Iterated and tested over the next decade;
  Consolidated in the last 5 years.
- People are using it!
  `github.com/jcsalomon/xpeek`

expl3 modules

Programming:

- basics / expan / quark / prg / token

Data types:

- int / dim / skip / box
- tl / seq / clist / prop

‘Complex’ data types/modules:

- msg / keys / file
- fp / coffins / (regex?)

Increasing complexity

```
\expandafter\ifx\csname foobar\endcsname\relax
  <not exist>
\else
  <exist>
\fi
```

This one…

```
\begingroup\expandafter\expandafter\expandafter\endgroup
\expandafter\ifx\csname foobar\endcsname\relax
  <not exist>
\else
  <exist>
\fi
```

And there is much worse! We want to mitigate complexity.

```
\cs_if_exist:NTF \foobar  {<exist>} {<not>}
\cs_if_exist:cTF {foobar} {<exist>} {<not>}
```

One of my favourite tricks

In plain:

```
\let\foo\bar

\expandafter\let\csname foo\expandafter
  \endcsname\csname bar\endcsname
```

In expl3:

```
\cs_set_eq:NN \foo  \bar
\cs_set_eq:cc {foo} {bar}
\cs_set_eq:Nc \foo  {bar}
\cs_set_eq:cN {foo} \bar
```

More expansion

A difficult case:

```
\foobar{abc}{\expandme}
```

How to expand `\expandme` once before this is seen by `\foobar`?

More expansion

A difficult case:

```
\foobar{abc}{\expandme}

\expandafter\foobar\expandafter
{\expandafter a\expandafter b\expandafter c\expandafter}%
\expandafter{\expandme}
```

More expansion

A difficult case:

```
\xfoobar{abc}{\expandme}
```

More expansion

A difficult case:

```
\xfoobar{abc}{\expandme}
```

```
\def\xfoobar#1#2{\@foobar{#2}{#1}}
```

More expansion

A difficult case:

```
\xfoobar{abc}{\expandme}

\def\xfoobar#1#2{\@foobar{#2}{#1}}
\def\@foobar#1#2{\expandafter\@@foobar\expandafter{#1}{#2}}
```

More expansion

A difficult case:

```
\xfoobar{abc}{\expandme}

\def\xfoobar#1#2{\@foobar{#2}{#1}}
\def\@foobar#1#2{\expandafter\@@foobar\expandafter{#1}{#2}}
\def\@@foobar#1#2{\foobar{#2}{#1}}
```

More expansion

A difficult case:

```
\xfoobar{abc}{\expandme}
```

```
\def\xfoobar#1#2{\@foobar{#2}{#1}}
\def\@foobar#1#2{\expandafter\@@foobar\expandafter{#1}{#2}}
\def\@@foobar#1#2{\foobar{#2}{#1}}
```

Now `\foobar` finally receives the arguments with correct prior expansion

More expansion

In expl3:

```
\foo_bar:no {abc} {\expandme}
```

```
\cs_new:Npn \foo_bar:nn #1#2 {...}
```

More expansion

In expl3:

```
\foo_bar:no {abc} {\expandme}

\cs_new:Npn \foo_bar:nn #1#2 {...}
\cs_generate_variant:Nn \foo_bar:nn {no}
```

Outline

A problem with $\LaTeX\,2_\varepsilon$

- 99% of the 2e kernel used by packages
- We cannot change the internals of the kernel!
- If only people didn't mess around with internals.
- Only *documented* interfaces should be used.

How?

- Can't (reasonably) enforce this in code.
- Can indicate this with code conventions.
- Can use `\tl_count:n` .
- *Cannot* use `\__tl_count:n` !

(Or if you do don't blame us!)

- TEX does not have name-spacing.
- Let's help with `docstrip`.
- Makes code easier to write.
- Helps enforce conventions.

$\LaTeX\,2_\varepsilon$ example

```
\begin{figure}...\end{figure}
```

All floats defined with `\@float...\end@float`.

Internally, uses `\@xfloat`.

Internal code in expl3 (.sty)

```
\cs_new_protected:Npn \seq_remove_duplicates:N #1
  {
    \__seq_remove_duplicates:NN \seq_set_eq:NN #1
  }

\cs_new_protected:Npn \__seq_remove_duplicates:NN #1#2
  {
    \seq_clear:N \l__seq_remove_seq
    \seq_map_inline:Nn #2
      {
        \seq_if_in:NnF \l__seq_remove_seq {##1}
          { \seq_put_right:Nn \l__seq_remove_seq {##1} }
      }
    #1 #2 \l__seq_remove_seq
  }
```

Internal code in expl3 (.dtx)

```
\cs_new_protected:Npn \seq_remove_duplicates:N #1
  {
    \@@_remove_duplicates:NN \seq_set_eq:NN #1
  }

\cs_new_protected:Npn \@@_remove_duplicates:NN #1#2
  {
    \seq_clear:N \l_@@_remove_seq
    \seq_map_inline:Nn #2
      {
        \seq_if_in:NnF \l_@@_remove_seq {##1}
          { \seq_put_right:Nn \l_@@_remove_seq {##1} }
      }
    #1 #2 \l_@@_remove_seq
  }
```

@@ summary

`\tl_count:n` is ‘public’.

`\@@_count:n` $\equiv$ `\__tl_count:n` is ‘internal’.

`\__tl_` to `\@@_` doesn’t save many letters here…

…but consider `\__fontspec_` !

Outline

‘Quick’ sorting in l3sort

```
\clist_set:Nn \l_foo_clist { 3 , 01 , -2 , 5 , +1 }

\clist_sort:Nn \l_foo_clist
  {
    \int_compare:nNnTF { #1 } > { #2 }
      { \sort_reversed: }
      { \sort_ordered: }
  }
```

Produces: ‘`-2,+1,01,3,5`’

TEX by Topic has an example of a lexicographic comparison.

Expandable floating point

- TeX uses integer arithmetic for everything
  even dimension calculation in multiples of `1sp`.
- Some have written maths modules for fixed point maths.
- Joseph wrote floating point maths.
- Bruno made it expandable!

Example

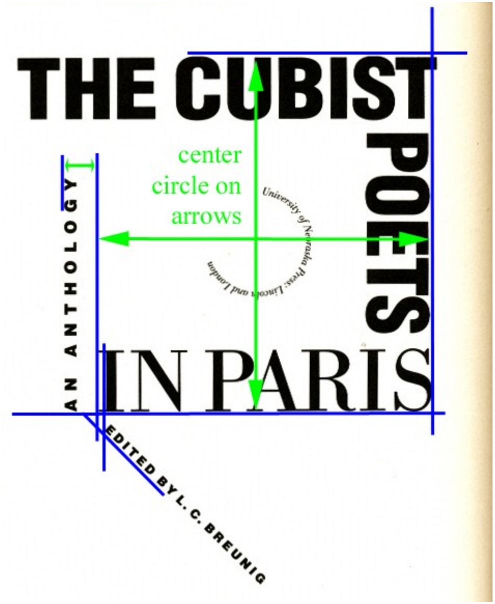
This code:

```
\usepackage{xparse, siunitx}
\ExplSyntaxOn
\NewDocumentCommand { \calcnum } { m }
  { \num { \fp_to_scientific:n {#1} } }
\ExplSyntaxOff

\calcnum {
  round ( 200 pi * sin ( 2.3 ^ 5 ) , 2 )
}
```

Produces: 6.2784×10^2

Coffins example



Regular expressions in l3regex

JWZ:

> *Some people, when confronted with a problem, think*
> *"I know, I'll use regular expressions."*
> *Now they have two problems.*

But regular expressions are useful!

What is regexp

Advanced pattern matching.

```
\tl_set:Nn \l_my_tl { That~cat. }
\regex_replace_once:nnN { at } { is } \l_my_tl
```

$\rightarrow$ ‘`This cat.`’

```
\tl_set:Nn \l_my_tl { This~cat~your~cat }
\regex_replace_all:nnN { \w+ } { \0 , } \l_my_tl
```

$\rightarrow$ ‘`This, cat, your, cat,`’

support tokens

- `‘\c{begin} \cB. (\c[^BE].*) \cE.’`
  matches: `\begin{`<anything without `{}`>`}`
- `[a-oq-z\cC.]` matches any lowercase latin letter except p, as well as control sequences.

Available functions:

- Match (TF)
- Count
- Extract
- Split
- Replace

Poor man's grep (for Windows users)

```
\ior_new:N \l_grep_stream

\cs_new:Npn \expl_grep:nn #1 #2
  {
    \ior_open:Nn \l_grep_stream {#2}
    \ior_str_map_inline:Nn \l_grep_stream
      {
        \regex_match:nnT {#1} {##1} { \texttt{##1}\\ }
      }
    \ior_close:N \l_grep_stream
  }

\expl_grep:nn {\\usepackage} {\jobname}
```

Poor man's grep (for Windows users)

This is the output:

```
\usepackage{expl3,l3regex,l3sort}
\usepackage{calc,graphicx}
\usepackage{metalogo,fancyvrb}
\usepackage{fontspec,siunitx}
\usepackage{biblatex}
\usepackage{xparse, siunitx}
\expl_grep:nn {\\usepackage} {\jobname}
\expl_grep:nn {\\usepackage} {\jobname}
```

Conclusion

The Hitch-Hiker's Guide to LaTeX3
by *Andrew Stacey* (TEX.sx Community Blog)

As with the original Hitch-Hiker's Guide, this blog post won't actually be all that useful to someone wanting to truly explore LaTeX3. It's more of a "What I did on my holidays" kind of guide. I've recently had my first go at doing some coding with LaTeX3 and I thought it might be interesting to record my experiences.

…

Will I use LaTeX3 again? Absolutely. I wouldn't choose it for a non-TeX situation, but if it's something to be done within TeX then LaTeX3 is definitely high up on the list of choices. Do I expect an easy ride? Not at all. But at the end, I expect a sense of accomplishment not quite like coding in any other language.

Conclusion

- LaTeX3 shouldn't be thought as monolithic
- Programming layer is solid and being used by others
- Document interface layer for LaTeX 2_ε available
- Current team focus is on the typesetting foundation layer
  - font selection
  - output routines
  - page layout