LaTeX3: Using the layers

It's alright ma, it's only witchcraft

Joseph Wright

1st June 2013

Outline

Outline

Layers of abstraction

Code	Data structures and commands to build higher-level typesetting elements
Functionality	Typesetting elements that can be customized to show varying behaviours
Design	Specific elements and formatting from the functionality layer
User	User level syntax to call instances

The LaTeX2e situation

User	`\section` * [⟨*TOC*⟩] {⟨*heading*⟩}
Design	Use of `\@startsection`: sets appearance of sections
Functionality	Arguments of `\@startsection`: *e.g.* vertical space above below, *etc.*
Code	`\if@noskipsec \leavevmode \fi \addpenalty` `...`

The design layer

Describing design

Ideally, document design could be written independent of code

> *Section headers will be set in 16 point sanserif, with space before the section of 6 points and after of 6 points unless immediately followed by a subsection in which case . . .*

Some ideas on this problem in xtemplate plus the 'LaTeX data base': these are difficult problems!

The functionality layer

```
\def\@startsection#1#2#3#4#5#6{%
  \if@noskipsec \leavevmode \fi
  \par
  \@tempskipa #4\relax
  \@afterindenttrue
  \ifdim \@tempskipa <\z@
    \@tempskipa -\@tempskipa \@afterindentfalse
  \fi
  \if@nobreak
    \everypar{}%
  \else
    \addpenalty\@secpenalty\addvspace\@tempskipa
  \fi
  \@ifstar
    {\@ssect{#3}{#4}{#5}{#6}}%
    {\@dblarg{\@sect{#1}{#2}{#3}{#4}{#5}{#6}}}}
```

The code layer

- Where do you find documentation for each of the following?
- Which can you use in your own code?

```
\def\@float#1{%
  \@ifnextchar[%
    {\@xfloat{#1}}%
    {\edef\reserved@a
      {\noexpand\@xfloat{#1}[\csname fps@#1\endcsname]}%
     \reserved@a}}
\def\@dblfloat{%
  \if@twocolumn\let\reserved@a\@dbflt\else
    \let\reserved@a\@float\fi
  \reserved@a}
\def\@xfloat #1[#2]{%
  \@nodocument
  \def \@captype {#1}%
```

Outline

\newcommand

With a star . . .

```
\newcommand*{\foo}{Code with no arguments}

\newcommand*{\foo}[2]{Code using #1 and #2}

\newcommand*{\foo}[2][]{Code using #1 and #2}

\newcommand*{\foo}[2][default]
  {Code using #1 and #2}
```

\newcommand

. . . or without

```
\newcommand {\foo}{Code with no arguments}

\newcommand {\foo}[2]{Code using #1 and #2}

\newcommand {\foo}[2][]{Code using #1 and #2}

\newcommand {\foo}[2][default]
  {Code using #1 and #2}
```

The aims of xparse

- Separate syntax from functionality
- Define all arguments in one place
- Intersperse mandatory and optional arguments
- More types of argument without code
- Mix long and short arguments
- Create engine robust commands
- Informative error messages

\...DocumentCommand

- \NewDocumentCommand
- \RenewDocumentCommand
- \ProvideDocumentCommand
- \DeclareDocumentCommand

\...DocumentCommand

- \NewDocumentCommand
- \RenewDocumentCommand
- \ProvideDocumentCommand
- \DeclareDocumentCommand

Syntax

\DeclareDocumentCommand {\⟨*command*⟩}
{⟨*arg. spec.*⟩} {⟨*code*⟩}

Mandatory arguments

```
\DeclareDocumentCommand{\foo}{ }
  {Code using no arguments}

\DeclareDocumentCommand{\foo}{ m }
  {Code using #1}

\DeclareDocumentCommand{\foo}{ m m }
  {Code using #1 and #2}

\DeclareDocumentCommand{\foo}{ m m m }
  {Code using #1, #2 and #3}
```

Mandatory arguments

Mixing short and long

```
\DeclareDocumentCommand{\foo}{ m m m }
  {Three short arguments}

\DeclareDocumentCommand{\foo}{ +m +m +m }
  {Three long arguments}

\DeclareDocumentCommand{\foo}{ m +m m }
  {Only #2 is long}
```

Optional arguments

(Almost) like LaTeX2e

```
\DeclareDocumentCommand{\foo}{ O{} }
  {One optional argument}

\DeclareDocumentCommand{\foo}{ O{default} m }
  {First argument optional with default value}
```

Optional arguments

Beyond LaTeX2e

```
\DeclareDocumentCommand{\foo}{ O{} O{} m }
  {Two optionals then a mandatory}

\DeclareDocumentCommand{\foo}{ o m }
  {%
    \IfNoValueTF{#1}
      {Code for just #2}
      {Code for #1 and #2}%
  }
```

Optional arguments

Beyond LaTeX2e

```
\newcommand*{\foo}[2][]
  {Code here}

\foo[\baz[arg1]]{arg2}

#1 = \baz[arg1
#2 = ]
```

Optional arguments

Beyond LaTeX2e

```
\DeclareDocumentCommand{\foo}{ O{} m }
  {Code here}

\foo[\baz[arg1]]{arg2}

#1 = \baz[arg1]
#2 = arg2
```

Stars

```
\DeclareDocumentCommand{\foo}{ s m }
  {%
    \IfBooleanTF #1
      {Process #2 with a star}
      {Process #2 without a star}%
  }
```

Re-implementing `\chapter`

The original

```
\def\chapter{...\secdef\@chapter\@schapter}

% syntax support code:
\def\secdef#1#2{\@ifstar{#2}{\@dblarg{#1}}}
\def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
\long\def\@dblarg#1%
  {\kernel@ifnextchar[{#1}{\@xdblarg{#1}}}
\long\def\@xdblarg#1#2{#1[{#2}]{#2}}
```

Re-implementing `\chapter`

In xparse

```
\ExplSyntaxOn % So we don't worry about spaces below!
\DeclareDocumentCommand { \chapter } { s o m }
   {
     \IfBooleanTF {#1}
        { \@schapter {#3} } % ignore [...] if given
        { \IfNoValueTF {#2}
            { \@chapter [#3] {#3} } % use title twice
            { \@chapter [#2] {#3} } % use optional arg
        }
    }
```

Other argument types

- `d` An optional argument delimited by two tokens
- `g` An optional argument in braces (‘group’)
- `r` A mandatory argument delimited by two tokens (‘required’)
- `t` A single optional token
- `v` An argument read verbatim

Outline

Aims

- A complete coding environment
- Fully documented
- Rigorously tested
- Clear guidance on what is usable
- ‘Best practice’ promoted by team
- . . .

Not much like TeX, is it?

```
\cs_new_protected:Npn \malmedal_output_reverse:
  {
    \seq_set_eq:NN \l__malmedal_temp_seq
      \g_malmedal_input_seq
    \seq_reverse:N \l__malmedal_temp_seq
    \int_set:Nn \l__malmedal_count_int
      { \seq_count:N \l__malmedal_temp_seq }
    \seq_map_inline:Nn \l__malmedal_temp_seq
      {
        \malmedal_print:n {##1}
        \int_decr:N \l__malmedal_count_int
      }
  }
```

A crash course in expl3

- Coding is done inside `\ExplSyntaxOn` ... `\ExplSyntaxOff`
- `:` and `_` are used *systematically* in names
- White space is ignored
- Separation of commands for internal use from those publicly-available (*documented*)

Not much like TeX, is it?

```
\cs_new_protected:Npn \malmedal_output_reverse:
  {
    \seq_set_eq:NN \l__malmedal_temp_seq
      \g_malmedal_input_seq
    \seq_reverse:N \l__malmedal_temp_seq
    \int_set:Nn \l__malmedal_count_int
      { \seq_count:N \l__malmedal_temp_seq }
    \seq_map_inline:Nn \l__malmedal_temp_seq
      {
        \malmedal_print:n {##1}
        \int_decr:N \l__malmedal_count_int
      }
  }
```

So you want to learn expl3?

- Read the documentation:
  - expl3
  - interface3
- Blog posts on `texdev.net`:
  search for ‘Programming LaTeX3’
- Ask questions on LaTeX-L or TeX-sx!

A lot done . . .

- Code layer works and is ‘out there’
- xparse is much more powerful than `\newcommand`
- Using these allows implementation of new ideas, such as xcoffins

. . . a lot left to do!

- Design layer is still to be solved
- The output routine is a major challenge
- There is always more to add to expl3
- Choosing what *not* to add to expl3 is also hard!