Towards \LaTeXe \ Coding Standards

Didier Verna

didier@lrde.epita.fr
http://www.lrde.epita.fr/~didier

TUG 2011 – Thursday, October 20
History


... *The Elements of whatnot Style*. 
Purpose

- help programmers to read and understand source code
- not only their own but that of others
- From the GNU Coding Standards:

  Their purpose is to make the GNU system clean, consistent, and easy to install. This document can also be read as a guide to writing portable, robust and reliable programs.
The coding standards many-festos

- **Readability**
- **Maintainability**

- **Robustness**
- **Reliability**

- **Portability**
- **Extensibility**
- **Intercession**

**Consistency:** the exact coding style is less important than actually sticking to it!
30 years and no style?  
... makes \LaTeX{} a dull toy...

- **Legacy**
  - Learning by example (learn the good *and* the bad)

- **Lack of help**
  - Liberal language (macro-expansion)
  - Editor support (complicated)

- **Lack of need**
  - A world of dwarfs  
    (\TeX{}Live 2009: average 327 LoC, median 134 LoC)
  - Antisocial development  
    (most packages single-authored)
30 years and *almost* no style?
...makes \LaTeX{} an *almost* dull toy...

- Tools
  - Blank lines, comment syntax
  - `calc`, `ifthen`, `doc`, `ltxdoc` *etc.*

- Conventions
  - `\usepackage` *vs.* `\RequirePackage`, @ character *etc.*
  - \LaTeX{} itself not even conformant (*e.g.* `\hbox`, `\m@ne`)

- Companion
  - Section 2.1 (Structuring of source files)
  - Section A.4 (Structuring of package files)
  - Less than 1% of the book...
The need for coding standards is real

- **Why?**
  - Learning by *good* example
  - Facilitate interaction
  - Clean up the current intercession mess

- **How?**
  - Level 1 (low): layout – formatting, indentation, naming schemes *etc.*
  - Level 2 (mid): design – modularity, encapsulation, other paradigms *etc.*
  - Level 3 (high): behavior – user interfaces, package interaction / conflict management *etc.*
  - Level 4 (meta): social
Blanks
Be spacey

1. Stay WYSIWYGly coherent
   ▶ \,, \par
   ▶ Tabular-like environments (&, \\)

2. Put only one “logical” instruction per line
   ▶ environment calls
     ▶ \expandafter\foo\bar
     ▶ \raggedleft\foo\bar baz

3. Be as spacey as you like in math mode
   ▶ blanks ignored

4. Grouping (*any* kind) \⇒ indentation
   ▶ {}, \begin{group} \end{group}, \makeatletter, \makeatother *etc.*

5. The % character is your friend!
Example
Choose your preferred style...

\def\@docinclude#1 {\clearpage
  \if@files \immediate\write@mainaux{\string\@input{#1.aux}}\fi
  \if@tempswatrue\if@partsw \@tempswafalse\edef@tempb{#1}\for
  \@tempa:=\@partlist\do{\ifx\@tempa\@tempb\@tempswatrue\fi}
  \fi
  \let\@auxout\@partaux \if@files \immediate\openout\@partaux #1.aux
  \immediate\write\@partaux{\relax}\fi
% ... \fi :-(
Braces for impact!
Where do you put them?

**Hmmm**

\newenvironment{env}[1] 
\{
\%
\opening\code
\opening\code
\}
\{
\%
\closing\code
\closing\code
\}

**Ok**

\newenvironment{env}[1] 
\{
\%
\opening\code
\opening\code
\}
\{
\%
\closing\code
\closing\code
\}

**Ouch!**

\newenvironment{env}[1] \%
\opening\code
\opening\code
\%
\closing\code
\closing\code

**Ok**

\newenvironment{env}[1]\%
\%
begin{env}{opt}
\opening\code
\opening\code
\%
end{env}
\%
\closing\code
\closing\code

- Note: brace position may require eol `% char

**LaTeX Style**

Didier Verna

**Introduction**

Coding Standards?
Not LaTeX ones?
What can we do?

**Layout**

Blanks
Names

**Design**

Duplication
Conditionals
Modularity

**Behavior**

Key/Value interfaces
Intercession

**Social**

Conclusion
Perspectives
Exceptional situations / oddities
No rule without exception

- Forced indentation

```latex
\@ifnextchar[\%] syntax screwup!
{\@fxbeginsenv\{#2\}}{\@@fxbeginsenv\{#2\}}
```

- Empty body-like macro arguments

```latex
\@ifundefined\{#1\note\}\{\%
\@fpkger\{command prefix \#1 already in use\}%
You have called \string\FXRegisterAuthor\space with a command prefix already in use.\MessageBreak
Please choose another one.\}%
```
How maniac can you be?

Inter-macro indentation

\newcommand\text{%
   @nextentry
   \noalign\bgroup
   \gdef\@beforespace { \subrubricbeforespace }%
   @ifstar { \@stext } { \@text } }

\newcommand\@text [1]{%    
   \gdef\@nextentry {}%    
   \egroup% end of \noalign opened in \text.    
   \multicolumn{3}{@{ }p{\linewidth}@{}}{\@rubrictextfont #1}\\
}

\newcommand\@stext{%
   \gdef\@nextentry {\egroup \par}%
   \egroup% end of \noalign opened in \text.    
   \multicolumn{3}{@{ }p{\linewidth}@{}}{\bgroun\@rubrictextfont}
Names
You get one for life, so beware...

1. Use prefixes
   - Avoid name clashes (e.g. `\text` in `CurVe` and `siunitx`)
   - Mandatory for styles, arguable for classes
   - Use one and stick to it!
     `\finkdir` vs. `\fnk@maindir`

2. Use postfixes (beware the `\new*` commands!)
   - `\newsavebox\myitemsBOX` vs. `\newcounter{myitems}`

3. From the Companion
   - Lowercase for API
   - Mixed case for extension API
   - @ character for internals (several levels)

4. But stop the madness!
   - `\@latexerr`, `\latex@error`
   - `\@input`, `@@input`, `\@input@`, `\filef@und`
   - `\sixt@@n`, `\g@addto@macro`
Examples

**API**

\fxnote
\fxuselayout

\FXLayoutInline
\FXRegisterAuthor

**Internals**

@fxnote
@fxuselayout

@FXLayoutInline
@FXRegisterAuthor

**Nesting levels**

\DeclareRobustCommand\fxnote {
  \%
  \@ifstar {
    \%
    \fxnote*
    \@ifnextchar [
      { \@@fxsnote { #2 } } { \@@@fxnote { #2 } } }
  { \@@fxsnote { #2 } } { \@@@fxnote { #2 } } } }%

\long\def\@fxsnote[#1][#2][#3][#4] {
  \%
  \for \ draft \ mode
  \% \ . \ . \ .
  \let \@@@fxnote@early\@@@fxnote@early@draft
  \long\def\@fxsnote[#1][#2][#3][#4] {
  \%
  \for \ final \ mode
  \% \ . \ . \ .
  \let \@@@fxnote@final\@@@fxnote@early@final

**Polymorphic macros**

\def\@@@fxnote@early@draft\for\draft\mode
\def\@@@fxnote@early@final\for\final\mode
\% \ . \ . \ .
\let \@@@fxnote@early\@@@fxnote@early@draft
Exceptional situations / oddities
No exception without rule

- Conforming to *de facto* standards
  - \ifmycondition
  - \listoffixmes, \listfixmename
  - But \fixmeindexname or \fxindexname?

- Forced exceptions
  - Manual: \l@fixme
  - Auto: \c@mycounter, \myenv, \endmyenv

- Commands vs. environments
  - \fxnote but \begin{anfxnote}\end{anfxnote}
General design rules

1. Don’t reinvent the wheel / Use existing tools
   - `calc`, `ifthen`, `record` (!) `etc.`
   - Higher abstraction $\rightarrow$ better readability

2. Duplication is evil / Copy-paste is evil
   - Use wrappers
   - Use abstractions

3. Conditionals are evil
   - Centralize the logic
   - Be polymorphic

4. Be modular
   - Use `docstrip`
   - Write small macros
Duplication is evil / Copy-paste is evil
Use wrappers and abstractions

**Bad**

\define@key [ fx ] { layout } { morelayout } {% ...}
\define@cmdkey [ fx ] { layout } { innerlayout } {% ...}
\define@key [ fx ] { envlayout } { envlayout } {% ...}

**Good**

\newcommand \@fxdefinekey {
\define@key [ fx ] }
\newcommand \@fxdefinecmdkey {
\define@cmdkey [ fx ] }

%%% ...

\@fxdefinekey { layout } { morelayout } {% ...}
\@fxdefinecmdkey { layout } { innerlayout } {% ...}
\@fxdefinekey { envlayout } { envlayout } {% ...}

**Bad**

\define@boolkey [ fx ] { lang } { langtrack } [ true ]{}
\@fxdefinevoidkey { lang } { nolangtrack } {}%
\@nameuse { fx@lang@langtrack } { false }{}

\define@boolkey [ fx ] { log } { silent } [ true ]{}
\@fxdefinevoidkey { log } { nosilent } {}%
\@nameuse { fx@log@silent } { false }{}

**Good**

\newcommand * \@fxdefineboolkey [ 3 ] [] {}%
\define@boolkey [ fx ] { #2 } { #3 } [ true ] [ #1 ]
\@fxdefinevoidkey { #2 } { no #3 } {}%
\@nameuse { fx@#2@#3 } { false }{}

%%% ...

\@fxdefineboolkey { lang } { langtrack }
\@fxdefineboolkey { log } { silent }
Conditionals are evil
Centralize the logic, be polymorphic

Bad
\newif\ifdraft
\def\do@everything{%
  \ifdraft
    \@dothis\this\way
  \else
    \@dothis\this\other\way
  \fi
%
%   ... 
  \ifdraft
    \@dothat\that\way
  \else
    \@dothat\that\other\way
  \fi
}
\DeclareOption{draft}{{
    \let\@dothis\@dothis@draft
    \let\@dothat\@dothat@draft
}\ExecuteOptions{final}{{
    \let\@dothis\@dothis@final
    \let\@dothat\@dothat@final
}\ExecuteOptions{final}{ProcessOptions

Good
\def\@dothis@draft{\this\way}
\def\@dothis@final{\this\other\way}
\def\@dothat@draft{\that\way}
\def\@dothat@final{\that\other\way}
\def\do@everything{%
  \@dothis
%
%   ... 
  \@dothat
\DeclareOption{draft}{{
    \let\@dothis\@dothis@draft
    \let\@dothat\@dothat@draft
}\DeclareOption{final}{{
    \let\@dothis\@dothis@final
    \let\@dothat\@dothat@final
}\ExecuteOptions{final}{ProcessOptions

---

**Latex Style**

---

Didier Verna

**Introduction**

Coding Standards?
Not Latex ones?
What can we do?

**Layout**

Blanks
Names

**Design**

Duplication
Conditionals
Modularity

**Behavior**

Key/Value interfaces
Intercession

**Social**

**Conclusion**

**Perspectives**
Only interested in the "green" lines...

Originally 203 LoC. 156 after dead branches removal.
Behavior

1. Be nice to your users (incl. yourself)
   - Document your packages *properly*
   - Be backward-compatible
   - Use key/value interfaces

2. Be nice to your hackers (incl. yourself)
   - Be bottom-up
   - Organize your code by feature

3. Intercession management
   - Localize behavior
   - `filehook` is crucial
Key/Value interfaces
How do I choose one? Yeah, I know...

- Package level
- `\mysetup` macro
- Macro level

**xkeyval example**

```latex
\ExecuteOptionsX [my]<fam1,fam2,\ldots>{opt1=def1,opt2=def2,\ldots}
\ProcessOptionsX *[my]<fam1,fam2,\ldots>

\newcommand*\mysetup [1] {\setkeys{my}{fam1,fam2,\ldots}{#1}}
\newcommand\mymacro [2] [] { %
    \setkeys{my}{fam1,fam2,\ldots}{#1}
    \ldots}
```
Behavior

1. Be nice to your users (incl. yourself)
   - Document your packages *properly*
   - Be backward-compatible
   - Use key/value interfaces

2. Be nice to your hackers (incl. yourself)
   - Be bottom-up
   - Organize your code by feature

3. Intercession management
   - Localize behavior
   - `filehook` is crucial
Standard interface too limited
The \LaTeX\ developer’s worst nightmare

- \@ifpackageloaded, \@ifclassloaded
  - Curative (*a posteriori*) code only
  - What about precautionary code?
- \AtBeginDocument
  - Massively defer code execution
  - What about the order?
- Example:
  - Style \textit{S} calls \AtBeginDocument{\texttt{\textbackslash things}}
  - Class \textit{C} loads style \textit{S}
  - How does \textit{C} intercede on \texttt{\textbackslash things}?
Start with your default behavior

Rewrite on demand and locally

Example: how $C_u V_e$ handles bibliography

filehook is crucial
Before and after hooks
Collaboration and Reactivity
One year and 38 weeks later…

1. Propose collaboration
   - Don’t keep it for yourself
   - Don’t reinvent the wheel

2. Accept collaboration
   - Be reactive
     - Review and accept patches
     - Examine and implement ideas
   - Open development
     - Use collaborative tools
     - Trust people
Conclusion
The golden rules of (ŁĄTĘX) programming

1. Coding style is important
2. Sticking to it is *more* important
3. Keep it in mind permanently
4. Let it evolve
5. No rule without exception