

Code Guidelines for L^AT_EX-Programming

Sebastian Mesow

February 8, 2020

Contents

1 General	1	1.4 Documentation of Macros and their Arguments	4
1.1 Macro Definitions	1	1.4.1 Normal Arguments	5
1.1.1 Multiple Similar Expressions with the same operator Multiple Times	2	1.4.2 Further Arguments	5
1.1.2 Avoid in-place-argument-calculation	2	1.4.3 Hidden Arguments	5
1.2 If Statements	3	2 Indention	5
1.3 Names of Macros	3	2.1 Source Code Groups	6
		3 Text in Source Code	6
		3.1 Info, Warning and Error Messages . . .	6

1 General

Every line with source code must end with a percent sign `%` or a comment. (Though in effect every non-blank line in the source file contains at least one percent sign.)

One command per line. Definitions normally are separated by an empty or blank line.

Macro definitions normally are separated by one blank line. An exception can be the definition of many similar macros that only hold values e.g. for localization or character things.

Every option in an option list is finished by a comma - also the last option. Many options in an options list should be spread over many lines. In this case the option list is another source code group.

Comment tricky stuff with line comments in the same line or before the tricky code.

1.1 Macro Definitions

Use `\newcommand*` (and **not** `\newcommand` or any of its derivatives for macro definitions that are functions. Prefer it over `\def` or any of its derivatives.

Example:

```
% syntax: \myMacro<code>\@nil
%
% #1 - the code to do something with (#1 is a delim arg)
\newcommand*{\myMacro}{}%
\def\myMacro #1\@nil{%
    % do something with all code until (not included) the first \@nil
}%
```

You can use `\def` or any of its derivatives for internal, “local” variables of functions.

1.1.1 Multiple Similar Expressions with the same operator Multiple Times

When concatenating two or more very similar expressions on the same level with always the same operator, then every line contains one such expression and starts with this operator (of course except the first line). Do not place the operator at the end of every line (then except the last).

Counter Example:

```
\ifthenelse{%  
  \equal{\@tempa}{option1}\or%  
  \equal{\@tempa}{option2}\or%  
  \equal{\@tempa}{option3}%  
}{%  
  % if known option  
}{%  
  % if unknown option  
}%
```

Example:

```
\ifthenelse{%  
  \equal{\@tempa}{option1}%  
  \or \equal{\@tempa}{option2}%  
  \or \equal{\@tempa}{option3}%  
}{%  
  % if known option  
}{%  
  % if unknown option  
}%
```

1.1.2 Avoid in-place-argument-calculation

Counter example:

```
\expandafter\ifx\expandafter\@tempa%  
  \ifx\@tempb\@tempc%  
    \@tempd% leads to \ifx\@tempa\@tempd  
  \else%  
    \@tempe% leads to \ifx\@tempa\@tempe  
  \fi%  
  % code executed%  
  % if \@tempa is equal to \@tempd and (precondition) \@tempb is equal to \@tempc  
  % or  
  % if \@tempa is equal to \@tempe and (precondition) \@tempb is not equal to \@tempc  
\else%  
  % other code  
\fi%
```

This can be simplified to with an extra value-holding macro

```
\ifx\@tempb\@tempc%  
  \let\compare@macro\@tempd% leads to \ifx\@tempa\@tempd  
\else%  
  \let\compare@macro\@tempe% leads to \ifx\@tempa\@tempe  
\fi%  
\ifx\@tempa\compare@macro%  
  % code executed%
```

```

    % if \@tempa is equal to \@tempd and (precondition) \@tempb is equal to \@tempc
    % or
    % if \@tempa is equal to \@tempe and (precondition) \@tempb is not equal to \@tempc
\else%
    % other code
\fi%

```

1.2 If Statements

When using classical \TeX if-else-statements, then the `\if condition`, `\else` and `\fi` statements are always on a separate line - even if the true or false code is empty. When using \LaTeX -style if-else-constructs like `\snrue_codeelse_code\@ifunde` then the complete, empty false code `{}` should be on the same line as the end of the true code. It can be use full to explicitly comment an empty true or false clause

Long and/or complicated expressions for the condition of an if statement should be an own source code group.

Examples for \TeX -Style if-else-statements:

```

\ifx\@tempa\@tempb%
    % true code
\else%
    % false code
\fi%

\ifx\@tempa\@tempb%
    % true code
\fi%

\ifx\@tempa\@tempb%
\else%
    % false code
\fi%

```

Examples for \LaTeX -style if-else-macros:

```

\@ifundefined{csn}{%
    % true code
}%
    % false code
}%

\@ifundefined{csn}{%
    % true code
}{}%

\@ifundefined{csn}{%
}%
    % false code
}%

```

1.3 Names of Macros

All internal macros of a package start with the same prefix and an at-sign `@`. This prefix is also an abbreviation for the package in general. Concepts or Features inside a package append another prefix and at-sign to the package prefix.

Multiple variants of “a” macro (that have something common, especially their arguments) have the same name, but are distinguished by an at-sign and a suffix. Same for variants of variants. Internal, local macros/variables also get an at-sign and a suffix. Empty prefixes and suffixes are forbidden (except for code of the LaTeX included format or other included formats and the LaTeX included classes or other included classes).

Use camel case. example: `camelCaseIsThatWeUse`

Examples:

```
% documentation of the macro
\newcommand*{\prefix@macroName}[1]{%
  % replacement text
}%

% documentation of the macro
\newcommand*{\prefix@feature@macroName}[2]{%
  % replacement text
}%

% documentation of the macro
\newcommand*{\prefix@macroName@variantI}[3]{%
  % replacement text
}%

% documentation of the macro
\newcommand*{\prefix@macroName@variantII}[3]{%
  % replacement text
}%
```

1.4 Documentation of Macros and their Arguments

Example:

```
% short description of what the macro does
%   - further informations about what it does (optional)
%
% syntax: \prefix@macro[first arg]{second arg}{first further arg} (optional)
%
% [#1] - short description of the first argument, default: "default val for first arg"
% #2   - short description of the second argument; must this argument be fully expandable?
%       - further informations about the second argument (optional)
% f1   - short description of the first further argument
\newcommand*{\prefix@macro}[2][default val for first arg]{%
  % replacement text using the arguments
  % when invoking an other macro at macro chaining:
  \prefix@macro@further% {first further arg} follows
}%
```

1.4.1 Normal Arguments

arg notation	explanation
<code>*</code>	star; Describe what it does when present and when not!
<code>#1</code>	mandatory argument
<code>[#1]</code>	optional argument
<code>[[#1]]</code>	mandatory argument that must be enclosed with brackets according the parameter text; optional argument of a higher macro that calls this macro; mostly only used at lower macros, that take these optional argument by call of the form <code>\@testopt{\lowermacro}{default arg}</code>

1.4.2 Further Arguments

Further arguments occur when chaining macros (see the pattern). You can say that they are arguments from the point of view of the API but you do not declare it in the definition of **this**, higher macro yet. Instead they are declared in the definition of a second, lower macro, that the higher macro calls at the end.

arg notation	explanation
<code>f*</code>	further star; Describe what it does when present and when not!
<code>f1</code>	mandatory further argument
<code>[f1]</code>	optional further argument
<code>[[f1]]</code>	mandatory further argument that must be enclosed with brackets

1.4.3 Hidden Arguments

Hidden Arguments are sensitive to the calling, outside context at the call of the macro. Although the other arguments are the same the concrete executed code of the macro can vary.

arg notation	explanation
<code>\macro</code>	macro from the calling context (hidden argument)
<code>cs macro@#1</code>	macro from the calling context (hidden argument) whose command sequence name is based by an other argument
<code>cnt counter</code>	integer value of a counter register (<code>L^AT_EXcounter</code>)
<code>cnt \counter</code>	integer value of a counter register (<code>T_EXcounter</code>)
<code>len \length</code>	length of a dimension register (<code>L^AT_EXlength</code>)
<code>dim \dimension</code>	length of a dimension register (<code>T_EXdimension</code>)
<code>skip \skip</code>	adjustable length of a skip register (<code>T_EX</code>)
<code>toks \tokens</code>	tokens of a token register (<code>T_EX</code>)

If an argument or macro takes discrete values - a so called enumeration - than also explain the possible values.

2 Indention

“Tabs” always consists of four spaces.

The start of macro definitions is normally not indented, but every source code line is intended by at least one tab. Every source code group increases the indention level by one.

2.1 Source Code Groups

- replacement text for macro definition
- replacement texts that consists of one simple command may not be intended
- arguments that are code, especially arbitrary long
- arguments that are long, e.g. quite long text
- long and/or complicated expressions for the condition of an if statement
- conditionally executed code in every kind of if-else constructs
- conditionally executed code in every kind of switch constructs
 - If the switch-construct takes "keys" and the conditionally executed code is the "value", then the "key" is normally with one extra tab intended and the "value code" is intended by two extra tabs.
- long options list

3 Text in Source Code

Always use `\par` instead of a blank line (two newlines). Such a `\par` is always on its own line.

Avoid using a normal space if the text is quite short (e.g. about 3 words or word equivalents); then instead use `\space`.

3.1 Info, Warning and Error Messages

Every line of info, warning and error messages ends with `\MessageBreak` or an equivalent (except of course the last line).