Abstract

The programming side of LaTeX is often overlooked. In many cases, the use of programming capabilities of LaTeX can facilitate document editing and improve the quality of the compiled document. LaTeX3 is a set of modern programming interfaces provided by the LaTeX kernel. Compared to traditional LaTeX programming, LaTeX3 yields more standardized, readable, and robust code. In this tutorial, we present a series of examples that help readers understand the commonly used modules of LaTeX3.

Modern LaTeX programming: an example based LaTeX3 tutorial

Ziyue Xiang

Contents

1 Introduction ? 1
1.1 Motivations of LaTeX3 ? 1
1.2 Compiling Examples ? 2
2 LaTeX3 Naming Conventions (I.1.1) ? 2
2.1 Category Codes and Command Names ? 2
2.2 Names of Variables ? 3
2.3 Names of Functions ? 3
2.4 The Use of LaTeX3 Naming Conventions ? 3
3 Understanding LaTeX3 Documentation (I.1.2) ? 4
4 Variables and Functions ? 4
A Scope of Variables ? 4

List of Examples

A.1 Scope of value assignment ? 4

1 Introduction

There is no doubt that LaTeX is viewed as a typesetting language by most of the users. The programming aspect of LaTeX is often overlooked by many. In practice, many large and structured documents can benefit from the programming capabilities provided by LaTeX. Even understanding the most basic programming principles in LaTeX can greatly facilitate the efficiency of generating figures or tables that are made up of similar and repeated sub-structures. The infrastructure provided by LaTeX 2_ε [2] is already Turing complete, which means its programming capabilities are identical to those of Python [12] and C [9]. However, the syntax and conventions of LaTeX 2_ε are nonstandardized and obsolete compared to the mainstream programming languages now. This makes learning LaTeX 2_ε programming more difficult for today's LaTeX users.

In order to modernize the programming interfaces in LaTeX, the LaTeX3 programming interfaces are introduced [5]. Unfortunately, the learning materials for this tool chain is scarce. One of the few resources available for new learners is *The LaTeX3 Interfaces* [11], which is a technical documentation that is difficult to for one to start with. Therefore, in this material we intend to provide an example based LaTeX3 tutorial for new LaTeX3 users with sufficient background in computer programming. That is, the reader is expected to understand basic structures (e.g., loops, conditional branches) as well as data types (e.g., integers, floating point numbers, strings) in computer programs.

Since the LaTeX3 project has accumulated a huge code base over the years, it is infeasible to cover all of its functionalities in one tutorial. In this tutorial, we focus on the most frequently used components in LaTeX3. The complete documentation of LaTeX3 can be found in [11]. In the upcoming section titles, if parentheses are present, then the content in the parentheses is the corresponding section number in the 2022-01-12 release of [11].

1.1 Motivations of LaTeX3

As mentioned above, LaTeX 2_ε is already Turing complete and serves as the building blocks of many existing packages. In this section, we describe the problems of traditional LaTeX programming, which justifies the reason why LaTeX3 is developed despite already having the powerful LaTeX 2_ε and many other existing packages.

Nonuniform interface Outside LaTeX3, the interfaces provided by traditional LaTeX are not standardized. They suffer from the following disadvantages.

Firstly, the mechanism of LaTeX can affect the readability of traditional LaTeX code. In LaTeX, functions and variables are all declared using control sequences (see Chapter 3 of [4]). When a function is invoked, it is expected to execute a series of predefined procedures. Variables are used to store values only. In LaTeX, we can declare functions that absorb one or multiple arguments. Sometimes arguments are stored in variables. Since both functions and its arguments can be both control sequences, it is difficult to distinguish between them. This is likely to make reading traditional LaTeX source code difficult.

Secondly, in traditional LaTeX the implementation of many fundamental programming capabilities are provided by external packages. For example, to compare the equality of two strings, we can use `\ifthenelse` and `\equal` from `ifthen` package [3]; we can use `\pdfstrcmp` from `pdftexcmds` package [7]; we can also use `\IfStrEq` from `xstring` package [10]. The use of multiple similar packages is likely to cause redundancy and compatibility issues. The lack of a centralized documentation and comparison for these similar packages also increases the learning cost of traditional LaTeX programming.

Expansion control Unlike generic programming languages, LaTeX does not have support for types. Programming components such as variables, functions and function arguments are all treated in the same way. Therefore, LaTeX programmers need to more precisely control how variables are defined and how functions are called. These techniques are known as expansion control. Expansion control in traditional LaTeX is achieved by using the `\expandafter` command, which is difficult to use because the number of `\expandafter` command calls required may scale exponentially [1].

LaTeX3 aims at mitigating these inconveniences in traditional LaTeX programming. It provides a uniform interface for LaTeX programmers, where functions and variables are separated from each other. It defines standardized infrastructure for many programming tasks such as string processing, numerical calculation, regular expression matching, etc. The new expansion control mechanism of LaTeX3 is easier and more straightforward to use.

1.2 Compiling Examples

This tutorial is based on examples. To compile the examples, the minimum preamble required is:

```
1 \documentclass{article}
2 \usepackage{tikz} % load TikZ for some TikZ
  ↪ examples
3 \usepackage{expl3} % load latex3 packages
```
1

The example code should be placed between `\begin{document}` and `\end{document}`. All examples were tested with TeXLive 2020 on Ubuntu 20.04. For newer versions of LaTeX compilers, there is no need to load the `expl3` package explicitly (i.e., Line 1:3[1] is optional).

The source code of this tutorial can be obtained from `https://github.com/xziyue/latex3-tutorial-latex` `source`.

2 LaTeX3 Naming Conventions (I.1.1)

Unlike many programming languages that enclose the function arguments with parentheses, LaTeX does not require delimiters between the function and its arguments. In the example below, we define 6 control sequences, where `\ta` and `\td` are functions, and the rest are variables.

```
1 \newcommand{\ta}[2]{[arg1={#1}, arg2={#2}]}
2 \newcommand{\tb}{$\alpha$}
3 \newcommand{\tc}{$\beta$}
4 \newcommand{\td}[1]{[arg3={#1}]}
5 \newcommand{\te}{$\gamma$}
6 \newcommand{\tf}{$\delta$}
7 \ta\tb\tc\td\te\tf
```

[arg1=α, arg2=β][arg3=γ]δ

2

On Line 2:7, we call functions `\ta` and `\td` with their respective arguments (stored in variables). We output the value of `\tf` next. In appearance, Line 2:7 is six control sequences placed next to each other. It is difficult to understand the code unless the programmer finds out which control sequences are functions and how many arguments each function absorbs. To improve readability, LaTeX3 introduces a special naming convention where functions and variables are clearly distinguishable. In addition, programmers can gather more information from function and variable names such as the number of function arguments, the type of variables, and the scope of variables.

2.1 Category Codes and Command Names

In LaTeX, every input character can be classified into 16 categories. Each category can be identified by an integer ranging from 0 to 15, which is known as the *category code*. More details about category codes can be obtained from [8]. We focus on one of the sixteen categories, which is known as "letter". In most cases, command names in LaTeX can only be made up of characters from the "letter" category. Because the "letter" category only contains the lowercase and uppercase versions of the 26 English alphabets by default, command names are comprised of these 52 characters exclusively under the initial LaTeX setup. By extending the "letter" category, it is possible to add more permissible characters to command names. For example, the `\makeatletter` command changes the category of `@` to "letter", which allows the character to be used in command names [6]. Many packages use the `@` character in their internal command names to maintain the integrity of defined commands. Since typical users can only access commands whose names are made up of English alphabets, this technique can prevent internal commands from being overwritten accidentally. The technique is also used in LaTeX3 to separate LaTeX3 from other LaTeX programming conventions. In LaTeX3, two new characters are introduced into

[1] Every listing has a unique index, which is shown at the bottom right corner. A line in the listing is referenced by <listing index>:<line number>. A range of lines in the listing are referenced by <listing index>:<line number 1>-<line number 2>.

the command names, namely _ (underscore) and : (semicolon).

The command `\ExplSyntaxOn` allows one to enter LaTeX3 programming mode. It changes the category code of underscore and semicolon to "letter". It also changes the category code associated with white space and line break characters so that they are ignored in LaTeX3 mode. Therefore, a white space should be explicitly entered with ~ or \␣ in LaTeX3 mode. The command `\ExplSyntaxOff` is used to exit LaTeX3 mode.

2.2 Names of Variables

In LaTeX3, the naming of variables and functions are different, which allows one to distinguish the two more easily. A LaTeX3 variable can be public or private depending on the semantics. The name of each LaTeX3 variable is made up of four parts: `scope`, `module`, `description`, and `type`. We will describe the meaning of each part later.

The name of a public variable should follow:

```
\<scope>_<module>_<description>_<type>
```

The name of a private variable should follow:

```
\<scope>__<module>_<description>_<type>
```

Private variables have an extra underscore between the `scope` part and the `module` part.

The meaning of each part is shown as follows.

- `scope`: a single letter identifying the scope of the variable; see Appendix A for more details
  - `l`: local variable
  - `g`: global variable
  - `c`: constant
- `module`: the name of the module where the variable is defined
- `description`: the description of the variable
- `type`: the variable type; some common types are shown in Table 1.

The `description` part can contain multiple segments separated by underscore(s). Some valid variables names are shown below.

- `\g_doc_variable_int`
- `\l_doc_bg_color_r_fp`
- `\l_doc_bg_color_g_fp`
- `\l__mypkg_tmpa_seq`
- `\c_left_brace_str`

Since the category code of underscore has been changed to "letter" under LaTeX3 mode, it can no longer be used to denoted subscript in math equations. One can use the predefined LaTeX3 constant `\c_math_subscript_token` to represent subscripts when programming in LaTeX3.

Type	Description
clist	comma separated list
dim	dimension
fp	floating point number
int	integer
seq	sequence
str	string
tl	token list
bool	boolean
regex	regular expression
prop	property list
ior	IO read
iow	IO write

Table 1: Commonly used variable types and their descriptions. See ?? for more information about variable types.

2.3 Names of Functions

To increase the readability of LaTeX3 code, a function name in LaTeX3 includes information about the arguments that it absorbs. The name of a public LaTeX3 function should follow:

```
\<module>_<description>:<arg-spec>
```

The name of a private LaTeX3 function should follow:

```
\__<module>_<description>:<arg-spec>
```

The semicolon is only used in LaTeX3 function names. The meaning of each part is described as follows.

- `module`: the name of the module where the function is defined
- `description`: the description of the function
- `arg-spec`: the argument specifications–it is made up of zero or more English letters describing the type of every function argument, where each letter specifies the type of an argument; some common argument specification letters are shown in Table 2.

Some valid function names are shown below.

- `\group_begin:`
- `\tl_put_right:Nn`
- `\tl_gput_right:Nx`
- `\__doc_do_something:Nnxx`

2.4 The Use of LaTeX3 Naming Conventions

The LaTeX3 naming conventions are essentially suggestions. In reality, the LaTeX compiler does not check if the code is written according to LaTeX3 naming conventions. However, using the naming conventions can greatly improve the readability of one's code. In Section ???, we can also see that LaTeX3's expansion control mechanism sometimes requires functions to be defined using LaTeX3 naming conventions. Almost all LaTeX3 packages are written

`arg-spec` Letter	Description
`n`	A token list argument
`N`	A single token argument
`V`	An argument passed by value
`o`	A token list argument expanded once
`x`	A token list argument expanded recursively
`T`	A true branch in conditional statements
`F`	A false branch in conditional statements
`p`	A parameter list
`c`	A token list argument that is fully expanded and treated as a command name

Table 2: Commonly used argument specification letters and their descriptions. More details are described in Section ???.

using the naming conventions. Therefore, it is recommended that all LaTeX3 users follow the LaTeX3 naming conventions strictly.

3 Understanding LaTeX3 Documentation (I.1.2)

Currently, most information about LaTeX3 is compiled in [11]. It is mainly a technical documentation on LaTeX3 functions, scratch variables, and constants. We describe briefly how [11] documents these concepts.

- LaTeX3 functions: the majority of the contents in [11] are about LaTeX3 functions. In [11], each function is documented using a block similar to the one shown in Fig. 1. The color background is added in the figure to help understand the block, it does not exist in the original document.

  The yellow part in Fig. 1 indicates the basic forms and the variants of the function. The first and third line in the yellow part are the basic forms for local and global assignment (see Appendix A), respectively. The second line shows the function variants for local assignment; the fourth line shows the function variants for global assignment. More details about function variants will be introduced in Section ???. The cyan part in Fig. 1 shows the basic usage of the function, where the meaning of each function argument is described. The magenta part in Fig. 1 shows the detailed description of the function.
- Scratch variables: similar to the C programming language [9], variables in LaTeX3 need to be defined before use. LaTeX3 has predefined a set of empty variables for convenience. These variables are known as scratch variables. In LaTeX3, each package will usually define several scratch variables. They are documented in specific sections, similar to the one demonstrated in Fig. 2. The left hand side of Fig. 2 shows the predefined scratch variables; the right hand side shows the description of the variables. It is recommended not to use scratch variables, especially in large projects or generic packages. This reduces the chances of collision.
- Constants: LaTeX3 defines a large number of constants such as the value of π, the value of e, special characters, etc. They are documented in specific sections, similar to the one demonstrated in Fig. 3. The left hand side of Fig. 3 shows the command name of the constant; the right hand side shows the description of the constant.

4 Variables and Functions

A Scope of Variables

In LaTeX3, the scope of value assignment is usually local. Example A.1 (Listing 3) shows the difference between local and global assignment. In this example, Lines 3:4-9 are enclosed in a group. Inside the group, the value of `\l_tmpa_tl` is locally assigned with `\tl_set:Nn`, while the value of `\g_tmpa_tl` is globally assigned with `\tl_gset:Nn`. Since the value assignment of `\l_tmpa_tl` is local, the scope of the assignment is constrained within the group. Therefore, when we use its value outside the group, we get the old value. In contrast, the value assignment of `\g_tmpa_tl` is able to go beyond group boundaries.

Example A.1: Scope of value assignment

```
1  \ExplSyntaxOn
2  \tl_set:Nn \l_tmpa_tl {old}
3  \tl_gset:Nn \g_tmpa_tl {old}
4  \group_begin:
5  \tl_set:Nn \l_tmpa_tl {new} % set value locally
6  \tl_gset:Nn \g_tmpa_tl {new} % set value globally
7  \par In~group:~\tl_use:N \l_tmpa_tl
8  \par In~group:~\tl_use:N \g_tmpa_tl
9  \group_end:
10 \par Outside~group:~\tl_use:N \l_tmpa_tl
11 \par Outside~group:~\tl_use:N \g_tmpa_tl
12 \ExplSyntaxOff
```

```
In group: new
In group: new
Outside group: old
Outside group: new
```

3

In Section 2.2, it is described that the scope of variables should be encoded as the first letter (prefix) of variable names. If a variable is to be assigned using `set` functions, we can consider it to be local and use `l` as the prefix. If a variable is to be assigned using `gset` functions, we can consider it to be global and use `g` as the prefix.

```
\tl_set:Nn
\tl_set:(NV|Nv|No|Nf|Nx|cn|cV|cv|co|cf|cx)
\tl_gset:Nn
\tl_gset:(NV|Nv|No|Nf|Nx|cn|cV|cv|co|cf|cx)
```

```
\tl_set:Nn <tl var> {<tokens>}
```

Sets <*tl var*> to contain <*tokens*>, removing any previous content from the variable.

Figure 1: An example of function documentation excerpted from Section IV.15.2 of [11]. Color background is added in this figure to help understand the documentation.

```
\l_tmpa_tl
\l_tmpb_tl
```

Scratch token lists for local assignment. These are never used by the kernel code, and so are safe for use with any LaTeX3–defined function. However, they may be overwritten by other non-kernel code and so should only be used for short-term storage.

Figure 2: An example of scratch variables excerpted from Section IV.15.8 of [11].

References

[1] S.V. Bechtolsheim. A tutorial on `\expandafter`. *TUGboat* 9(1):57–61, 1988.

[2] K. Berry, S. Gilmore, T. Martinsen. *LaTeX 2ε: An Unofficial Reference Manual*. 12th Media Services, 2017.

[3] D. Carlisle, The LaTeX Project Team, L. Lamport. *The `ifthen` package*. `https://ctan.org/pkg/ifthen?lang=en`

[4] D.E. Knuth, D. Bibby. *The TeXbook*. Addison-Wesley Reading, 1984.

[5] F. Mittelbach, the LaTeX Project Team. Quo vadis LaTeX(3) team — a look back and at the upcoming years. *TUGboat* 41(2):201–207, 2020.

[6] A. Munn. What do `\makeatletter` and `\makeatother` do?, January 2011. `https://tex.stackexchange.com/questions/8351/what-do-makeatletter-and-makeatother-do`

[7] H. Oberdiek. *The `pdftexcmds` package*. `https://ctan.org/pkg/pdftexcmds?lang=en`

[8] Overleaf. Table of TeX category codes. `https://www.overleaf.com/learn/latex/Table_of_TeX_category_codes`

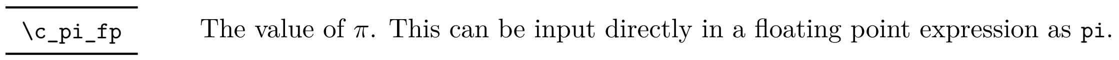
[9] D.M. Ritchie, B.W. Kernighan, M.E. Lesk. *The C programming language*. Prentice Hall Englewood Cliffs, 1988.

[10] C. Tellechea. *The `xstring` package*. `https://www.ctan.org/pkg/xstring`

[11] The LaTeX Project Team. The LaTeX3 Interfaces, January 2020. Release 2022-01-12. `https://tug.org/svn/texlive/trunk/Master/texmf-dist/doc/latex/l3kernel/interface3.pdf?revision=61588&view=co`

[12] G. VanRossum, F.L. Drake. *The Python language reference*. Python Software Foundation Amsterdam, Netherlands, 2010.

◇ Ziyue Xiang
Purdue University
`ziyue.alan.xiang (at) gmail (dot) com`
`https://www.alanshawn.com`
ORCID 0000-0001-6054-5801



`\c_pi_fp` The value of π. This can be input directly in a floating point expression as `pi`.

Figure 3: An example of scratch variables excerpted from Section IV.28.6 of [11].