Abstract

LaTeX3: Programming in LaTeX with Ease

Ziyue “Alan” Xiang

Contents

List of Examples

1 Introduction

Many people view LaTeX as a typesetting language and overlook the importance of programming in document generation process. In reality, many large and structural documents can benefit from a programming backend, which enhances layout standardization, symbol coherence, editing speed and many other aspects. Despite the fact the standard LaTeX (LaTeX 2_ε) is already Turing complete, which means it is capable of solving any programming task, the design of numerous programming interfaces is highly inconsistent due to the long history of LaTeX. This makes programming with LaTeX 2_ε extremely daunting, even for seasoned computer programmers.

To make programming in LaTeX easier, the LaTeX3 programming interface is introduced, which aims to provide modern-programming-language-like syntax and library for LaTeX programmers. Unfortunately, learning materials for this wonderful language is scarce. One of the few resource available for new users is *The LaTeX3 Interfaces* [5], which is essentially an API documentation that is not designed for introductory purposes. This situation may have barred many LaTeX users from utilizing the generic programming capabilities of LaTeX. Therefore, this article intends to provide an easy-to-understand tutorial for LaTeX users with computer programming knowledge. Hopefully, readers can improve their LaTeX editing efficiency and document quality after understanding LaTeX3.

This article is largely based on *examples*, which demonstrate different aspects of LaTeX3 programming. The minimum preamble for all examples is

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

Examples should be placed between `\begin{document}` and `\end{document}`. All examples are tested with TeXLive 2020 on Ubuntu 20.04.

Since the scale of LaTeX3 is huge, it would be infeasible to cover all modules and functionalities in one tutorial. As a result, this article only focuses on most frequently used components in LaTeX3. The complete API documentation can be found in *The LaTeX3 Interfaces* [5]. The numbers surrounded by parentheses in some section titles of this tutorial indicate the corresponding documentation location in *The LaTeX3 Interfaces*.

1.1 Why LaTeX3?

As mentioned above, LaTeX 2_ε is already Turing complete and serves as the building block of many important packages. What are the reasons for switching to LaTeX3?

Better expansion control Fundamentally, LaTeX works by doing *substitution*: commands are substituted by their definition, which is subsequently replaced by definition’s definition, until something irreplaceable is reached (e.g. text or TeX primitives). This process is called *expansion*. The mechanism of expansion may sound simple and straightforward. However, it usually requires a lot of manual fine-tuning in practice.

Consider the example below. We know that the `\uppercase` macro capitalize English letters, which renders the first output line in all caps. But if we store some text in `\myname` and then apply `\uppercase` to the command, we can see that the output is *not* turned into uppercase letters.

```
\par\uppercase{Alan Xiang}
\newcommand*{\myname}{Alan Xiang}
\par\uppercase{\myname}
```

ALAN XIANG
Alan Xiang

Why would this happen? Let us dig into how `\uppercase` works. The `\uppercase` macro scans each token[1]inside its argument group one by one. If an English letter is encountered, its uppercase form is left in the output stream. If a command is encountered, it will not try to apply `\uppercase` to the content of the command. Instead, the command itself will be placed into the output stream. In this case, `\myname` will be left untouched in the output, which is subsequently expanded to its original definition.

[1] Tokens are smallest units that TeX compilers work with. For now, we can consider a token to be either a character or command. For more about TeX tokens, see [3].

What if we also want to capitalize the content of \myname as well? To achieve this, we need to fine-tune the expansion process by changing the *order* of expansion. That is, to expand \myname before \uppercase. In this way, the \uppercase command will receive the content of \myname in the form of English letters, which allows capitalization to function correctly.

In LaTeX, the classic way of controlling the order of expansion is via the \expandafter macro, which it is notoriously difficult to use. According to *A Tutorial on* \expandafter [1], to reverse the expansion of a series of n tokens, the ith token has to be preceded by $2^{n-i} - 1$ \expandafters. The exponential growth of the number of \expandafters greatly reduces the readability of source code and increases the chances of mistakes. For example, in Joseph Wright's answer to an expansion-related question on TeX StackExchange [4], a total of 26 \expandafters are used to reorder the expansion of merely 4 arguments. To avoid this annoyance, one of the key features of LaTeX3 is to provide simple and reliable expansion control.

Standardized interface Just like any other generic programming languages, LaTeX provides integer, floating point, string and container variables. However, tradition LaTeX interface for these types is very messy. For example, to compare the equality of two strings, we can use \ifthenelse and \equal from ifthen package; we can use \pdfstrcmp from pdftexcmds package; we can also use \IfStrEq from xstring package. The fact that so many heterogeneous packages provide similar functionalities induces redundancy and potential compatibility issues. Therefore, LaTeX3 is to provide a set of unified and standardized interfaces for all possible LaTeX variable types.

Modernized experience TeX was first designed in the late 1970s, when computer hardware and programming languages were prototypes compared to their contemporary counterparts. As a result, TeX and LaTeX contain quirky usages that may seem odd for programmers today. For example, to multiply a counter variable by 3, one writes \multiply\counter by 3; to invoke the date command via the terminal, one writes \immediate\write18{date}. It can be seen that these syntaxes are either outdated or perplexing. In a fairly popular language nowadays (e.g. Python), these two tasks can be done by counter*=3 and os.system('date'), whose code possesses superior simplicity and interpretability. LaTeX3 attempts to modernize the LaTeX language by adapting to modern-language-like syntaxes and introducing a naming system that makes LaTeX code more readable.

2 LaTeX3 Naming Conventions (I-1)

In Python or C++, if we see a(b);, we can tell a is a function and b is its argument. However, in LaTeX, if we see \a\b, there are be two possibilities:

- \a is a function and \b is its argument
- Both \a and \b are variables

The syntactic design of LaTeX makes it difficult to distinguish between functions and variables, for each control sequence can either be a function that receives arguments or a variable that absorbs nothing. It can lead to confusion when one is trying to understand others' source code. Therefore, LaTeX3 introduces a set of naming rules that encode important information into the name of control sequences as a way to improve readability.

Before discussing LaTeX3 naming conventions, let us take a diversion to look at the low-level design of LaTeX and find out how we can use non-English characters in command names.

2.1 Category Code & Command Name

When the LaTeX compiler reads a source file, it will read and process each character one by one. For each character in the file, in addition to its character code, LaTeX compiler will also assign a *category code* based on current category code table. The default LaTeX category code table is shown in Table 1.

Category Code	Description	Character(s)
0	Escape character: tells LaTeX to start looking for a command	\
1	Start of group	{
2	End of group	}
3	Toggle math mode	$
4	Alignment tab	&
5	End of line	'\r'
6	Macro parameter	#
7	Superscript	^
8	Subscript	_
9	Ignored character	'\0'
10	Spacer	'\32', '\t'
11	Letter	A–Z, a–z, ...
12	Other	0–9, +, @...
13	Active character: used for single character commands	~...

Category Code	Description	Character(s)
14	Comment character: ignore everything that follows until end of line	`%`
15	Invalid character: not allowed in `.tex` files	`'\127'...`

Table 1: Default LaTeX category code table [2]. Characters surround by single quotes indicate their C-style representation.

LaTeX reacts to each character according to its category code instead of character code. If we change the category code associated with a character, we can completely change the *meaning* of that character. For example, if we assign category code 7 to `_` and category code 8 to `^`, we can use `_` to denote superscript and `^` to denote subscript.

Example 1: Doing 123

```
\ExplSyntaxOn
\tl_set:Nn \l_tmpa_tl {A}
\group_begin:
\tl_set:Nn \l_tmpa_tl {B}
\par value~inside~group:~\tl_use:N \l_tmpa_tl
\group_end:
\par value~outside~group:~\tl_use:N \l_tmpa_tl

\tl_set:Nn \l_tmpb_tl {A}
\group_begin:
\tl_gset:Nn \l_tmpb_tl {B}
\par value~inside~group:~\tl_use:N \l_tmpb_tl
\group_end:
\par value~outside~group:~\tl_use:N \l_tmpb_tl
\ExplSyntaxOff
```

value inside group: B
value outside group: A
value inside group: B
value outside group: B

References

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

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

[3] Overleaf. What is a "TeX token"? `https://www.overleaf.com/learn/latex/Articles/What_is_a_%22TeX_token%22%3F`.

[4] PLK. Expanding arguments before macro call. `https://tex.stackexchange.com/questions/104506/expanding-arguments-before-macro-call`, March 2013.

[5] The LaTeX3 Project. The LaTeX3 Interfaces. `http://ctan.math.washington.edu/tex-archive/macros/latex/contrib/l3kernel/interface3.pdf`, October 2020.

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