
L^AT_EX3: An outsider's overview

Joseph Wright

Abstract

The current experimental L^AT_EX3 packages provide a new, documented programming interface for T_EX. The key ideas implemented in this new interface are highlighted in this article.

1 Introduction

Modifying the behaviour of L^AT_EX 2_ε often requires a combination of user macros, internal L^AT_EX macro and T_EX primitives. This makes even trial modifications of document layout potentially difficult, even for the experienced L^AT_EX user. The differing syntax used by T_EX primitives and the L^AT_EX kernel only add to the confusion here.

The first step to develop a new L^AT_EX kernel is therefore to address how the underlying system is programmed. Rather than the current mix of L^AT_EX and T_EX macros, the experimental L^AT_EX3 system provides its own consistent interface to all of the functions needed to control T_EX. A key part of this work is to ensure that everything is documented, so that L^AT_EX users can work efficiently without needing to be familiar with the internal nature of the kernel or with plain T_EX.

The current kernel also suffers from the mixing of design commands with structural code. Thus changing a layout element often requires modifying a kernel code block (or loading a package which provides an interface to achieve this). The second challenge for L^AT_EX3 is therefore separation of the basic tools of the kernel from the design of documents.

This short overview article highlights the key developments to date in L^AT_EX3. It is based on my own experience working with the new tools for writing packages, and a talk given recently to the UK T_EX Users Group.

2 The components of L^AT_EX3

Currently, the experimental L^AT_EX3 packages are designed to be used “on top of” L^AT_EX 2_ε. This avoids needing to wait for the entire kernel to be finished before testing what is written.

The most developed part of the code is the `expl3` bundle, the core of the new kernel providing the new programming interface. The new language is fully documented in the file `source3.pdf`, which contains some notes for the experienced (L^A)T_EX programmer.

Built on top of `expl3` is the `xparse` package. This is meant to be a “bridge” between the internal and user parts of the new kernel. The `xparse` package

is used to create new user macros, in a much more controlled way than is possible using `\newcommand`.

More experimental than `xparse` are various other “`xpackages`”. These are designed to explore new approaches to layout and document design for L^AT_EX3.

The most complete part of L^AT_EX3 is the `expl3` bundle. The rest of this article is focussed mainly on the new internal syntax introduced in `expl3`.

3 A new internal syntax

L^AT_EX3 does not use `@` as a “letter” for defining internal macros. Instead, the symbols `_` and `:` are used in internal macro names to provide structure. In contrast to the plain T_EX format and the L^AT_EX 2_ε kernel, these extra letters are used only between parts of a macro name (no strange vowel replacement).

L^AT_EX3 separates macros which do something (functions) from ones which only store data. The general form of an internal function in L^AT_EX3 is `\<module>_<function>:<arg-spec>`. The `<module>` prefix is applied to almost all macros. For a package, it will typically be the package name; the kernel is split into a number of modules, each with its own name. The name of the `<function>` should give a good description of what it does: this may contain one or more `_` characters to divide the name into logical units.

The concept of the `<arg-spec>` is potentially confusing to existing (L^A)T_EX programmers. This *argument specifier* describes the arguments expected by the function. In most cases, each argument is represented by a single letter. The letter and its case then conveys information about the type of argument required. The use of the `<arg-spec>` is illustrated later in this article.

3.1 Primitives renamed

All of the T_EX primitives are given new names by `expl3`. Many are also given new L^AT_EX-like wrappers, so that the argument syntax is consistent. Many basic primitives have names which are little altered from the T_EX original.

At the most basic level, the `\fi` primitive becomes `\fi:`, indicating that no arguments are required. A more complex example is `\ifx`, which becomes `\if_meaning:NN`.

```
\if_meaning:NN \Macro_One \Macro_Two
% Do Stuff
\fi
```

Here, the `<arg-spec>` contains two letters, showing that two arguments are required. Both arguments are shown to be of type `N`, meaning that they should be single tokens *not* surrounded by braces.

3.2 Example kernel functions

Renaming primitives helps to keep the new syntax consistent, but does not show why the argument specifier is useful. This is perhaps best seen by looking at some of the functions provided by `expl3`.

By using the argument specifier, the new kernel provides families of related functions which avoid the need for complex `\expandafter` runs. For example, the \TeX primitive `\let` can only be used with two macro names. In \LaTeX 3, the family of `\let` macros contains:

```
\let:NN \Macro_One \Macro_Two
\let:Nc \Macro_One {Macro_Two}
\let:cN {Macro_One} \Macro_Two
\let:cc {Macro_One} {Macro_Two}
```

where the argument specified as `c` be given in braces and should expand to a `csname`. This is much clearer than the equivalent plain \TeX constructions; taking `\let:Nc` as an example:

```
\expandafter\let\expandafter\Macro_One
  \csname Macro_Two\endcsname.
```

The specifiers `n` (no expansion), `o` (expand once) and `x` (`\edef`-like expansion) allow large families of related functions to be created easily, so that using the results is easy. Thus we can create a macro `\Macro_One:nn`, then create `\Macro_One:no`, `\Macro_One:xn` and so on very rapidly.

The argument specifier concept also makes testing much easier. As an example, the new kernels provides three tests related to the `\ifundefined` macro:

```
\cs_if_free:cT {csname} {true}
\cs_if_free:cF {csname} {false}
\cs_if_free:cTF {csname} {true} {false}
```

In all three cases, the first argument will be converted to a `csname` (the `c` specifier). The first two functions then require one more argument, either `T` or `F`. As might be expected, these are executed if the test is true or false, respectively. The third function (ending `:cTF`) has both a true and false branch. By providing tests with the choice of `T`, `F` and `TF` arguments, empty groups in code can be avoided and meaning is much more obvious.

4 Data storage

In \LaTeX 3, macros which carry out some process are called functions, and all contain an argument specifier. Macros used for storage are handled separately, to help to make code cleaner and easier to read. To further aid the programmer, `expl3` defines several new data types:

- Token list pointers (`tlp`);

- Comma lists (`clist`);
- Property lists (`plist`);
- Sequences (`seq`).

in addition to the existing types, which are renamed:

- Boolean switches (`bool`);
- Counters (`int`);
- Skips (`skip`);

and so on.

The name “token list pointer” may cause confusion, and so some background is useful. \TeX works with tokens and lists of tokens, rather than characters. It provides two ways to store these token lists, within macros and as token registers (`toks`). \LaTeX 3 retains the name “`toks`” for the later, and adopts the name token list pointer for macros used to store tokens. In most circumstances, the `tlp` data type is more convenient for storing token lists.

The other new variable types are all essentially lists of items separated by a special token. The nature of the separator determines the type of variable and what functions apply. For example, a comma list is, rather obviously, a set of tokens separated by commas.

These are all created explicitly as either local or global. For example, a `tlp` may be named

```
\l_<module>_<name>_tlp
```

(local) or

```
\g_<module>_<name>_tlp
```

(global). The other variable types follow the same pattern, with the appropriate type identified in the variable name.

As well as the new data types, `expl3` provides a range of functions for manipulating data. Often, these have to have been coded by hand when using \LaTeX 2 ϵ . For example, `\tlp_elt_count:N` is available, to count the number of elements (usually letters) in a `tlp`.

5 Other key features

The new kernel will require the ϵ - \TeX extensions. This means that the new primitives are definitely available when working with \LaTeX 3. For example, `\unexpanded` is part of the expansion module, as `\exp_not:n`.

Boolean switches in \TeX and \LaTeX 2 ϵ use the `\iftrue` and `\iffalse` primitives. This can lead to problems with nesting (`Incomplete \if...`). To avoid this, \LaTeX 3 does not create switches in the same way. This means that all of the switches use exclusively \LaTeX syntax, and require an “access” function.

```
\bool_if:NT \l_example_bool {true code}  
\bool_if:NF \l_example_bool {false code}  
\bool_if:NTF \l_example_bool {true code}  
  {false code}
```

One of the most useful features of the new coding syntax is the treatment of white space. The literal space character () is ignored inside code block, meaning that the text can be laid out to aid ease of reading. When a space is required in the output, the hard space (~) is used. The ability to finish lines without needing % is highly welcome!

6 Conclusions

The current L^AT_EX₃ modules provide a new and powerful programming language for T_EX. The full details of the language are collected in one place, and the language is much more logical than the current mix of T_EX and L^AT_EX_{2 ϵ} . L^AT_EX₃ is therefore ready for serious use by (L^A)T_EX programmers.

At this stage, the document level of L^AT_EX₃ is much less defined. It seems likely that good separation of programming and document design will be made available. The new code syntax means that a number of ideas currently implemented as independent packages will need to be re-implemented either in the new kernel or as supported tools.

My own experience with L^AT_EX₃ convinces me that the kernel team need outsiders to use the code. The team have done a very good job so far, but everyone will bring new approaches using to the code. With the involvement of the wider T_EX community, L^AT_EX₃ has the potential to be a major step forward for L^AT_EX.

◇ Joseph Wright
Morning Star
2, Dowthorpe End
Earls Barton
Northampton NN6 0NH
United Kingdom
joseph.wright@morningstar2.co.
uk