Creating document commands: The good, the bad and the ugly

Joseph Wright

Creating document commands in LaTeX has traditionally involved a mix of `\newcommand`, semi-internal kernel commands (like `\@ifnextchar` and `\@ifstar`) and low-level TeX programming using `\def`. As part of wider efforts to improve LaTeX, the team have over the past few years developed ideas for creating document commands in the package xparse. In a parallel article (on `\NewDocumentCommand`, on the following pages), I've looked at how the xparse ideas compare to the abilities of other packages.

The aims of xparse have always been two-fold: to provide a clear way to create *new* commands, and to provide a language to describe *existing* ones. It is also intended to be as flexible as possible, so it doesn't impose artificial restrictions on syntax. That comes at a cost, however: it can be (ab)used to create commands that do not fit into the standard LaTeX pattern.

The LaTeX kernel now integrates most, though not all, of xparse into the kernel. That means the core ideas are available out-of-the-box. This seems like a good time, therefore, to look at the best ways to use the abilities of xparse in making document commands. I won't look at the full detail, but rather pick out how to, and how not to, create good document commands.

1 The Good

The LaTeX kernel is very careful to have consistent syntax for document commands. It uses only a small number of the possible argument types, which I'll describe in xparse terms:

- Mandatory (`m`) arguments in braces.
- Optional (`o`/`O{<default>}`) arguments in `[]`, which may have a default; in xparse terms we can tell the difference between a missing optional argument and one given with an empty `[]` pair.
- An optional star (`s`).
- Picture co-ordinates (`r()`), which are split into x and y, so in xparse terms subject to `\SplitArgument`.

Most of the time, the LaTeX kernel makes arguments *long*, which is shown as `+` in xparse syntax.

A star is *always* used as the first argument after a command, so in some ways it looks like part of the command name itself. Optional arguments are *almost* always given before mandatory ones, and most of the time there is *only one*. Where two are used, for example with `\makebox`, it's because the second is *strictly dependent* on the presence of the first.

Following the kernel, signatures (argument descriptions) such as:

```
s o m
s O{<default>} m m
o m m
```

are 'good'. You *can* use something like

```
s +m O{0} +o +m
```

(with optional arguments after a mandatory one; this is the syntax of `\newcommand`!) if you are careful, but *think very carefully.*

There's one syntax that's not from the kernel but is recommended where it applies: the beamer overlay syntax, which is `d<>` in xparse terms. This always comes first (other than a star), and is best reserved for the 'on X slides of Y' idea in presentations (doesn't have to be using beamer).

xparse lets us create arguments using `_` and `^`, similar to TeX's core math mode syntax. Most of the time, this should be reserved for math mode where you need to emulate the TeX syntax but for some reason need to grab the arguments yourself. This is done using `e{^_}`.

2 The Bad

The above already shows we have quite a few combinations available. Things go bad when too many combinations are used. For example:

- Multiple optional arguments where the second or subsequent ones don't strictly depend on the earlier ones.
- Optional arguments using tokens other than `[]` (or `<>` for overlays).
- Testing for tokens other than `*` as 'a special case' (think things like `+`).

Almost always, complex setups using these types of combination mean *you need to rethink the syntax.* In particular, multiple optional arguments tend to be much better replaced by using a keyval approach.

3 The Ugly

Some ideas in xparse won't be making it to the kernel: these are definitely the Ugly. They'll stay in a stub xparse for historical reasons, and as they do describe some syntax choices people have made, but in truth, they should be avoided:

- *Optional* groups (`g`) in braces; breaks the LaTeX conventions badly.
- Arguments *up to* a left brace (`l`); useful at a low level, but not in a document command.

- Arguments *up to* a token (`u`); widely used in programming, but again not in document commands.

You might wonder why they are all there in the first place: these were part of the more experimental work in xparse, and those particular experiments have shown we don't want to enable such syntaxes even for emulating existing commands.

4 A Fistful of Tokens

There are of course places where you need to go outside of the xparse structures, particularly when parsing specialist data. The popular Ti*k*Z graphics system is one example; linguistic glosses are another. But these are restricted contexts, normally used within a dedicated environment where it is clear that the 'usual' rules do not apply. Basically, if you do this, you are on your own, so be sure to check the balance of consistency *versus* compactness.

5 For a Few Tokens More

Using xparse syntax makes it *much* easier to have a clear break between interface and implementation. As such, the fact that it's got more going on 'beneath the hood' is worth it: it's a lot easier to track what's happening. The move into the kernel will make using xparse descriptions even easier to exploit, so it's important that users defining their own commands give a little thought to the syntax they choose.

◇ Joseph Wright
Northampton, United Kingdom
`joseph dot wright (at) morningstar2.co.uk`