Creating new document commands in \LaTeX has traditionally been the job of \newcommand. This lets you create a command with mandatory arguments, and also support a first optional argument. However, it can’t create more complex commands: \LaTeX uses stars, multiple optional arguments, and plenty more. To define commands using such syntaxes, the kernel itself uses lower-level \TeX programming. But this is opaque to many users, and a variety of packages have been created to ease the burden.

Over the last decade, the \LaTeX team have developed xparse, a generic document command parser, as a way to unify many ideas and provide a single consistent way to create document commands. The bulk of that code has now been moved to the \LaTeX kernel, and in a parallel article (starts on the preceding page) I’ve provided some ideas about how best to exploit that.

In this article, I want to look at a related issue: why to use this ‘xparse’ approach, and how it compares to existing solutions, both in the \LaTeX kernel and the wider package sphere. Here, I’m going to avoid talking about ‘simple’ shortcuts (things such as \newcommand\myname{Joseph}); these are best left to \newcommand. Instead, I want to deal with commands which take arguments and have some element of ‘programming’ to them.

What I’ll seek to highlight here is that using \NewDocumentCommand, we get a single consistent and reliable way to create a variety of commands. There’s no need to worry about clashes between approaches, and it all ‘just works’.

1 Preliminaries: Protected commands and optional arguments

Before we start, a couple of things are worth mentioning. First, there is the idea of ‘protected’ commands. In some places, we need commands not to ‘expand’ (turn into their definition). With a modern \TeX system, that can be arranged by the engine itself (pdf\TeX or similar), using \TeX’s \protected primitive (built-in). The \LaTeX kernel doesn’t use that mechanism in \newcommand, but lots of other tools do. I’m going to assume that we want to make protected commands unless I mention otherwise. Almost always, unless you are creating a ‘shortcut’ for some text, you want your commands to be protected.

The second thing to note is that \TeX itself has no concept of optional arguments, so they are always arranged using some clever look-ahead code. In xparse, nested optional arguments are handled automatically, but again, \newcommand and similar do not do that.

2 The kernel: versus \newcommand

The kernel’s \newcommand can, as I’ve said, create commands with multiple mandatory arguments but with only one optional one. A simple example:

\begin{verbatim}
\newcommand\foo[3][default]{% 
  Code perhaps using #1 and 
  definitely using #2 and #3%
}
\end{verbatim}

We can of course create an equivalent command using \NewDocumentCommand:

\begin{verbatim}
\NewDocumentCommand\foo{+O{default} +m +m}{% 
  Code perhaps using #1 and 
  definitely using #2 and #3%
}
\end{verbatim}
You may notice that I’ve used \texttt{+m} for both of the mandatory arguments, as that matches \texttt{\textbackslash newcommand}: the arguments can accept paragraphs (is \texttt{\textbackslash long}, in \TeX{} terms). With \texttt{\textbackslash newcommand}, all arguments either accept \texttt{\textbackslash par} or do not: with \texttt{\textbackslash NewDocumentCommand} we can select on a per-argument level what happens.

The optional argument with a default works using \texttt{O{default}}, and the result will be the same functionality as \texttt{\textbackslash newcommand}. We gain the idea that nested optional arguments are parsed properly, some better error messages if we use \texttt{\textbackslash foo} incorrectly, and an engine-robust definition of \texttt{\textbackslash foo}.

We can’t do a lot more with \texttt{\textbackslash newcommand}, so rather than try to show other \texttt{\textbackslash NewDocumentCommand} features here, we’ll first consider how we might make more complex syntaxes using just the classical \TeX{} kernel.

3 ... \textit{versus} \texttt{\textbackslash def}: The primitive

Using the \TeX{} primitive \texttt{\textbackslash def}, plus the kernel internal commands \texttt{\textbackslash@ifstar} and \texttt{\textbackslash@ifnextchar}, we can construct more complex syntaxes. For example, let’s create the syntax for \texttt{\textbackslash section}: a star, an optional argument and a mandatory one. I’ll assume we have \texttt{\textbackslash@} defined as a letter here. I’m also going to pass the presence of a star as the text \texttt{true} or \texttt{false}, as it makes things clearer.

\texttt{\textbackslash newcommand\textbackslash section\{\%
  \textbackslash@ifstar
    {\textbackslash sectionauxi\{true\}}
    {\textbackslash sectionauxi\{false\}}\%
  \}
\textbackslash def\textbackslash section@starred\#1\{\%
  \textbackslash@ifnextchar[\%
    {\textbackslash sectionauxi\{\#1\}}
    {\textbackslash sectionauxi\{\#1\}\[\]}\%
  \}
\textbackslash long\texttt{\textbackslash def\textbackslash sectionauxi\#1[\#2]\#3\%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
6 ... versus \texttt{\textbackslash withsuffix: suffix}

The \texttt{suffix} package allows one to extend an existing command to look for an optional token ("suffix") immediately after the command name. Taking a simple example from StackExchange (https://tex.stackexchange.com/a/4388), we start with

\begin{verbatim}
\newcommand\foo{blah}
\WithSuffix\newcommand\foo*[{blahblah}]
\end{verbatim}

which translates to

\begin{verbatim}
NewDocumentCommand\foo{s}{%\IFBooleanTF{#1}{blah}{blahblah}}
\end{verbatim}

This means we only need one line for the interface set up, and don’t need, for example, to split up grabbing optional arguments into two different places (as in the previous example with \textbackslash section).

7 ... versus \texttt{\newcommandx: xargs}

The \texttt{xargs} package is perhaps the most complete approach to extending \texttt{\newcommand} as far as optional arguments are concerned. It provides \texttt{\newcommandx}, which has the same syntax as \texttt{\newcommand} but where the second optional argument is a key–value list, which then describes which arguments are optional, and what their defaults are. Taking an example from the documentation:

\begin{verbatim}
\newcommandx*\coord[3][2=1,3=n]{% 
  (#2_{#1}, \ldots, #2_{#3})% 
}
\end{verbatim}

would create a command with two optional arguments, \texttt{#2} and \texttt{#3} (each with defaults), leaving \texttt{#1} mandatory. Translating into \texttt{\NewDocumentCommand} syntax might make that clearer!

\begin{verbatim}
NewDocumentCommand\coord[m 0{1} 0{n}]{% 
  (#2_{#1}, \ldots, #2_{#3})% 
}
\end{verbatim}

The \texttt{xargs} package has the idea of \texttt{\textbackslash usedefault}, which allows \texttt{[]} to be the same as \texttt{[default]}. That’s not something \texttt{xparse} does, as it is pretty confusing: what happens when you want an empty optional argument? This links to something I’ve said before: avoid consecutive optional arguments unless the second is dependent on the first.

8 ... versus \texttt{\texttt{\newcommand.py: newcommand}}

Stepping outside of \texttt{\LaTeX} itself, Scott Pakin’s Python script \texttt{\texttt{\newcommand.py}} provides a description language somewhat like \texttt{xparse}, and converts this into a ‘template’ of \texttt{\LaTeX} code, allowing a ‘fill in the blanks’ approach to creating commands. It can cover several of the ideas that \texttt{xparse} can, including a few that will not be migrated to the \texttt{\LaTeX} kernel. It can also set up a command taking more than 9 arguments, but that’s always going to be tricky as a user.

What is important is that using a script means we have to work in two steps, and it’s hard to see what’s happening from the \texttt{\LaTeX} source. It also doesn’t offer anything that the kernel doesn’t already do: no protected commands, no nested optional arguments, no improved error messages. So in many ways this is using techniques we’ve already seen, just made a little more accessible, at least if you have Python installed.

9 ... versus \texttt{\texttt{\newenvironment: environ}}

As well as document commands, the \texttt{xparse} syntax can be used to create document \texttt{environments}: the same relationship we have between \texttt{\newcommand} and \texttt{\newenvironment}. What people sometimes want to do is grab an entire document environment body and use it like a command argument. Classically, one does that using the \texttt{environ} package. Again, taking an example from the documentation:

\begin{verbatim}
\newenvironment{test}{% 
  \fbox{\parbox{1.5cm}{\BODY}}\color{red} 
  \fbox{\parbox{1.5cm}{\BODY}}% 
}\end{verbatim}

would grab all of the body of the environment \texttt{test} and typeset it twice, the first time in red. That is, the environment body is saved as \texttt{\BODY}.

Using \texttt{\NewDocumentEnvironment}, we have a syntax similar to \texttt{\newenvironment}

\begin{verbatim}
\NewDocumentEnvironment{test}{+b}{% 
  \fbox{\parbox{1.5cm}{\BODY}}\color{red} 
  \fbox{\parbox{1.5cm}{\BODY}}% 
}\end{verbatim}

with the argument grabbed in the normal way as (here) \texttt{#1}. We can therefore have ‘real’ arguments first, then grab the body.

10 Summary

Using the tools set up in \texttt{\NewDocumentCommand}, we can have a consistent way of creating a wide range of document commands. Rather than use a mixture of tools, from the kernel, the \texttt{\LaTeX} engine, and the package sphere, it is far preferable to use the single interface of \texttt{\NewDocumentCommand} for defining new commands today.

\begin{verbatim}
\section*{Acknowledgments}

\noindent \textit{Thank you to}\ \textit{advocates of the \texttt{\LaTeX} newcommand interface}\ and all who have contributed to the \texttt{\LaTeX}   \texttt{\newcommand} package.
\noindent \textit{Your work is appreciated.}
\end{verbatim}

\begin{verbatim}
\section*{Bibliography}

\begin{thebibliography}{9}
\bibitem{1} Joseph Wright
\textit{Northampton, United Kingdom}
\texttt{joseph dot wright (at) morningstar2.co.uk}
\end{thebibliography}
\end{verbatim}

\begin{verbatim}
\begin{itemize}
\item \texttt{\newenvironment} \texttt{\textbackslash withsuffix: suffix}
\end{verbatim}

\begin{verbatim}
\newcommand\foo{blah}
\WithSuffix\newcommand\foo*[{blahblah}]
\end{verbatim}

\begin{verbatim}
NewDocumentCommand\foo{s}{%\IFBooleanTF{#1}{blah}{blahblah}}
\end{verbatim}

\begin{verbatim}
\newcommandx*\coord[3][2=1,3=n]{% 
  (#2_{#1}, \ldots, #2_{#3})% 
}
\end{verbatim}

\begin{verbatim}
NewDocumentCommand\coord[m 0{1} 0{n}]{% 
  (#2_{#1}, \ldots, #2_{#3})% 
}
\end{verbatim}

\begin{verbatim}
\newenvironment{test}{% 
  \fbox{\parbox{1.5cm}{\BODY}}\color{red} 
  \fbox{\parbox{1.5cm}{\BODY}}% 
}\end{verbatim}

\begin{verbatim}
\NewDocumentEnvironment{test}{+b}{% 
  \fbox{\parbox{1.5cm}{\BODY}}\color{red} 
  \fbox{\parbox{1.5cm}{\BODY}}% 
}\end{verbatim}

\begin{verbatim}
\section*{Acknowledgments}

\noindent \textit{Thank you to}\ \textit{advocates of the \texttt{\LaTeX} newcommand interface}\ and all who have contributed to the \texttt{\LaTeX}   \texttt{\newcommand} package.
\noindent \textit{Your work is appreciated.}
\end{verbatim}

\begin{verbatim}
\section*{Bibliography}

\begin{thebibliography}{9}
\bibitem{1} Joseph Wright
\textit{Northampton, United Kingdom}
\texttt{joseph dot wright (at) morningstar2.co.uk}
\end{thebibliography}
\end{verbatim}

\begin{verbatim}
\begin{itemize}
\item \texttt{\newenvironment} \texttt{\textbackslash withsuffix: suffix}
\end{verbatim}