Beyond \newcommand with xparse

Joseph Wright

1 Introduction

The LaTeX 2ε \newcommand macro is most LaTeX users' first choice for creating macros. As well as the 'sanity checks' it carries out, the ability to define macros with an optional argument is very useful. However, to go beyond using a single optional argument, or to create more complex input syntaxes, LaTeX 2ε users have to do things 'by hand' using \def or load one of the packages which extend \newcommand (for example twoopt (Oberdiek, 2008)).

As part of the wider efforts to develop LaTeX3, the xparse package (LaTeX3 Project, 2009) aims to replace \newcommand with a much more flexible set of tools. This means looking again at the way that commands are defined, and so xparse uses different syntax conventions to \newcommand. In this article, I will be looking at some of the highlights of xparse.

2 Creating document commands

LaTeX 2ε provides not only \newcommand, but also \renewcommand and \providecommand, all sharing a common syntax. xparse also provides a family of related commands following the same pattern:

- \NewDocumentCommand For defining a macro not already defined, giving an error message if it is.
- \RenewDocumentCommand For changing a definition, issuing an error message if the macro does not already exist.
- \ProvideDocumentCommand Creates a macro if it does not exist, and otherwise does nothing: i.e., will not change an existing definition.
- \DeclareDocumentCommand Does no checks for an existing definition: simply defines the macro using the expansion given.

As \DeclareDocumentCommand always creates an updated definition, it is most convenient for the examples in the rest of this article.

The \DeclareDocumentCommand function takes three mandatory arguments:

1. The name of the function to define;
2. An 'argument specification';
3. The code which the function expands to.

```
\DeclareDocumentCommand \foo { m } {%
  % Code here
}
```

The first and third arguments are essentially the same as the equivalents for \newcommand: it is the argument specification that marks out an xparse definition. As you might guess from the above example, it is enclosed in braces, and spaces are ignored.

3 Argument specifications

The basic idea of an argument specification ('arg spec') is that each argument is listed as a single letter. This means that the number of letters tells you how many arguments a function takes, while the letters themselves determine the type of argument. As the argument specification is a mandatory argument, a function with no arguments still needs an arg spec.

```
\DeclareDocumentCommand \foo { } {%
  % Code with no arguments
}
```

xparse provides a range of argument specifier letters, some of which are somewhat specialised. The following is therefore only covers the most generally useful variants in detail.

Mandatory arguments are created using the letter m. So

```
\DeclareDocumentCommand \foo { m m } {%
  % Code with 2 arguments
}
```

is nearly equivalent to

```
\newcommand*\foo[2]{%
  % Code with 2 arguments
}
```

The 'nearly' is an important point: in contrast to \newcommand, xparse functions are not \long by default. In xparse, we can decide for each argument whether to allow paragraph tokens or not. This is done by preceding the arg spec letter by +:

```
\DeclareDocumentCommand \foo { m +m } {%
  % #1 No \par tokens allowed
  % #2 \par tokens permitted
}
\DeclareDocumentCommand \foo { +m +m } {%
  % Both arguments allow \par
}
```

LaTeX optional arguments with no default value are given the letter o, while those with a default value are given the letter O. The latter also requires the default itself, of course!

```
\DeclareDocumentCommand \foo { o m } {%
  % First argument optional, no default
  % Second argument mandatory
}
\DeclareDocumentCommand \foo
  { O{bar} m } {%
  % First argument optional, default "bar"
  % Second argument mandatory
}
```

The use of two separate letters here illustrates another LaTeX3 concept: functions used for setting up a document should have a fixed number of mandatory arguments. So while `o` is given with no additional information, `O` must always be given along with the default (as shown).

Thus far, the xparse method does not go significantly beyond what is possible using `\newcommand`. However, as well as recognising more types of argument, xparse also allows free mixing of optional and mandatory arguments. For example, it is easy to create a function with two optional and two long mandatory arguments in one step.

```
\DeclareDocumentCommand \foo
  { o +m o +m } {%
  % Four args, #1, #2, #3 and #4
  % Only #2 and #4 can include \par tokens
}
```

Creating this type of behaviour is far from trivial without xparse.

Generalising the idea of a LaTeX 2_ε optional argument, which is always enclosed in square brackets, xparse can create optional arguments delimited by any pair of tokens. This is done using the letters `d` (no default value) and `D` (with default value): 'd' stands for 'delimited'. So we can easily add an argument in parentheses or angle brackets, for example.

```
\DeclareDocumentCommand \foo
  { d() D<>{text} m } {%
  % Optional #1 inside ( ... )
  % Optional #2 inside < ... >
  %   with default "text"
  % Mandatory #3
}
```

A standard LaTeX 2_ε method to indicate a special variant of a macro is to add a star to its name. xparse uses the letter `s` to indicate this type of argument. There is then a need to indicate if a star has been seen. This done by returned one of two special values (`\BooleanTrue` or `\BooleanFalse`), which can be checked using the function `\IfBooleanTF`:

```
\DeclareDocumentCommand \foo { s m } {%
  \IfBooleanTF #1 {%
    % Starred stuff using #2
  }{%
    % Non-starred stuff using #2
  }%
}
```

A generalised version of the `s` specifier, with the letter `t` for 'token'. This works in exactly the same way, but for an arbitrary token, which is given following the 't'.

```
\DeclareDocumentCommand \foo { t/ m } {%
  \IfBooleanTF #1 {%
    % Code if a slash was seen
  }{%
    % Code if no slash was seen
  }%
}
```

Of the more specialised specifier letters, perhaps the most interesting is `u`, to read 'up to' some specified value.

```
\DeclareDocumentCommand \foo
  { u{stop} } {%
  % Code here
}
\foo text stop here
```

Here, the code will parse '`text␣`' as `#1`. Following standard TeX behaviour, the space between 'text' and 'stop' will be picked up as part of the argument.

4 Optional arguments

`\newcommand` does not differentiate between an optional argument which has not been given and one which is empty:

```
\newcommand\foo[2][]{%
  % Code
}
\foo{bar}
\foo[]{bar}
```

In both cases, `#1` is empty: not entirely helpful. It is possible to get around this using a suitable default value, but xparse aims to solve this problem in a general fashion.

When no default is available for an optional argument, xparse will return the special marker `\NoValue` if the argument is not given. It is then possible to check for this marker using the `\IfNoValue` test:

```
\DeclareDocumentCommand \foo { o m } {%
  \IfNoValueTF{#1}{%
    % Stuff just with #2
  }{%
    % Stuff with #1 and #2
  }%
}
```

Following the standard LaTeX3 approach, this test is available with versions which only have a true or false branch:

```
\DeclareDocumentCommand \foo { o m } {%
  \IfNoValueF{#1}{%
    % Stuff with #1
  }%
  % Stuff with #2
}
```

5 Robustness

xparse creates functions which are naturally 'robust'. This means that they can be used in section names and so on without needing to be protected using `\protect`. This makes using functions created using xparse much more reliable than using those created using `\newcommand`, particularly when there are optional arguments.

xparse is also designed so that optional arguments can themselves contain optional material. For example, if you try

```
\newcommand*\foo[2][]{%
  % Code
}
\foo[\baz[arg1]{arg2}]{arg3}
```

you will find that `\foo` will pick up '`\baz[arg1`' as `#1` and '`arg2`' as `#2`: not what is intended. However, the same code with xparse

```
\DeclareDocumentCommand \foo { o m } {%
  % Code
}
\foo[\baz[arg1]{arg2}]{arg3}
```

will parse '`\baz[arg1]{}`' as `#1` and '`arg`' as `#2`, as anticipated.

6 Fully expandable commands

There are a small number of circumstances under which fully expandable detection of optional arguments is desirable. For example, the etextools package (Chervet, 2009) provides a number of utility macros to produce this type of macro.

Rather than require the learning of an entirely new method for creating purely expandable commands, xparse can generate them in an analogous manner to normal (robust) commands.

```
\DeclareExpandableDocumentCommand \foo
  { o m } {%
  % Expandable code
}
```

This process has some limitations, some of which can be detected by xparse at definition time. It is therefore intended for exceptional use when a robust command will not behave suitably.

7 Environments

In analogy to the relationship between `\newcommand` and `\newenvironment`, xparse provides the function `\DeclareDocumentEnvironment` (and variants) for creating environments. The same argument specifications are used for declaring the arguments to `\begin{...}`. The crucial difference to standard LaTeX 2_ε environments is that the arguments are also available in the `\end{...}` code.

```
\DeclareDocumentEnvironment {foo} { o m } {%
  % Begin code using #1 and #2
}{%
  % End code using #1 and #2
}
```

8 Conclusions

By providing a single interface for defining both simple and complex user functions, xparse frees us from needing to worry about the detail of parsing input. Almost all cases can be covered without the need to use low level methods to process input.

Final note: you can use xparse in LaTeX 2_ε. Just:

```
\usepackage{xparse}
```

References

Chervet, Florian. "The etextools package: An ε-TeX package providing useful (purely expandable) tools for LaTeX users and package writers". Available from CTAN, `macros/latex/contrib/etextools`, 2009.

LaTeX3 Project. "The xparse package: Generic document command processor". Available from CTAN, `macros/latex/contrib/xpackages/xparse`, 2009.

Oberdiek, Heiko. "The twoopt package". Part of the oberdiek bundle, available from CTAN, `macros/latex/contrib/oberdiek`, 2008.

◇ Joseph Wright
Morning Star
2, Dowthorpe End
Earls Barton
Northampton NN6 0NH
United Kingdom
joseph dot wright (at)
morningstar2 dot co dot uk