The optparams package*

Jonathan Sauer
jonathan.sauer@gmx.de

2004/08/14

Abstract

This file describes the optparams package that provides a small macro for creating macros with multiple optional parameters.

Contents

1 Introduction

The LaTeX macro `\newcommand` is used to define macros that may have one optional parameter (the first one). However, there is no easy way of defining macros having several optional parameters. This package provides the macro `\optparams` to simplify the creation of macros with multiple optional parameters (or optional parameters only).

*This document corresponds to optparams.sty v0.9, dated 2004/08/14. The package is based on David Kastrup's macros for 'Around the Bend 21'.

2 Examples

2.1 Example one

```
\long\def\test@[#1][#2][#3][#4]{%
  (#1) (#2) (#3) (#4)
}

\newcommand{\test}{%
  \optparams{\test@}{[one][two][three][four]}%
}
```

You have now defined a macro `\test` that takes up to four optional parameters (`#1` to `#4`). You can call this macro as:

- `\test[this]`, resulting in `(this) (two) (three) (four)`.
- `\test[this][is]`, resulting in `(this) (is) (three) (four)`.
- `\test[this][is][a]`, resulting in `(this) (is) (a) (four)`.
- `\test[this][is][a][test]`, resulting in `(this) (is) (a) (test)`.

You see that the default parameters defined in `\test` are replaced one by one by the parameters given when `\test` is called.

2.2 Example two

```
\long\def\test@[#1][#2][#3][#4]#5{%
  (#1) (#2) (#3) (#4) !#5!
}

\newcommand{\test}{%
  \optparams{\test@}{[one][two][three][four]}%
}
```

You have now defined a macro `\test` that takes up to four optional parameters (`#1` to `#4`) and one mandatory parameter (`#5`). You can call this macro as:

- `\test[this]{foo}`, resulting in `(this) (two) (three) (four) !foo!`.
- `\test[this][is]{foo}`, resulting in `(this) (is) (three) (four) !foo!`.
- `\test[this][is][a]{foo}`, resulting in `(this) (is) (a) (four) !foo!`.
- `\test[this][is][a][test]{foo}`, resulting in `(this) (is) (a) (test) !foo!`.

3 Description of the macros

`\optparams` Usage: `\optparams {`⟨*macro*⟩`} {`⟨*default parameters*⟩`}`.

This macro reads any optional parameters from the input and finally calls your macro with the optional parameters as well as the remaining default parameters.

Each default parameter must have the form `[`⟨*value*⟩`]`. As all default parameters are passed as one parameter to `\optparams`, they must be enclosed in braces (`{`⟨*default parameters*⟩`}`).

The macro that is finally called must have as many optional parameters as have been defined in the call of `\optparams`; their form must be `[`⟨*parameter*⟩`]`, where ⟨*parameter*⟩ is a number beginning with one and ending with nine.[1] The parameter numbers must be numbered consecutively, i.e. `[#1][#2][#3]` instead of `[#1][#2][#4]`.

Because of this special way of declaring macro parameters, you cannot use `\newcommand` but have to use the TEX primitiv `\def`. As `\def`, contrary to `\newcommand`, does not warn you when you overwrite an already existing macro, you first should define the macro as a dummy using `\newcommand` and then correctly using `\def`, i.e.:

```
\newcommand{\test@}{}
\long\def\test@[#1][#2][#3][#4]{%
  <macro>
}
```

This way you do not accidentally overwrite an existing macro.

What does the `\long` do? In TEX, in order to quicker capture errors such as missing right braces `}`, parameters of macros defined using `\def` cannot contain a `\par`, or TEX will complain ('runaway argument?'). This makes it easier for the user to spot mistakes, as in that case TEX will stop processing immediately at the end of the current paragraph and not continue until i.e. the end of the file before realizing that a macro parameter has not been closed via `}`.

But there is a way to make the usage of `\par` as a macro parameter possible: Using `\long` in front of the `\def`. In LATEX, `\newcommand` defines macros as `\long` by default, and its variant `\newcommand*` defines macros restricted to 'short' parameters. So as demonstrated in the examples, you should either use `\newcommand` and `\long\def` together or `\newcommand*` and `\def` in order to make your macros behave correctly.

4 Notes

- If you use the notation used in the examples above, ⟨*macroname*⟩ for the main macro and ⟨*macroname*⟩`@` for the macro finally called by `\optparams`,

[1] This is a general restriction of TEX: A macro cannot have more than nine parameters.

and if you define these macros not in a package or class, but in your document preamble, you have to enclose the definitions in `\makeatletter` ... `\makeatother`, otherwise you cannot use `@` in macro names.

- If you define a macro having only optional parameters (as in the first example above), you have to make sure the macro is not called in a context where a `[` follows not as a parameter, but simply as an opening bracket, i.e. `\test[foo] [as a side note` ..., where the call of `\test` should have only `[foo]` as its only parameter. In this case `\optparams` will think that another optional parameter follows, resulting in chaos.

  To prevent this from happening, insert a `\relax` after the last optional parameter of the macro call, i.e. `\test[foo]\relax [as a side note` ... Then `\optparams` will stop looking for more optional parameters.

5 Implementation

5.1 Main macros

`\optparams` Calls `#1` using a variable number of parameters. Default parameters are provided in `#2` in the form `[`⟨*param one*⟩`] [`⟨*param two*⟩`]` ...

```
\newcommand{\optparams}[2]{%
  \optparams@{#1}{}{#2}%
}
```

5.2 Internal macros

`\optparams@` Checks if the next character from the input is a `[`. If true calls `\optparams@@`, as there are still optional parameters left. Otherwise calls ⟨*macro*⟩ with ⟨*parameters read*⟩ and ⟨*remaining default parameters*⟩.

Usage: `\optparams@ {`⟨*macro*⟩`} {`⟨*parameters read*⟩`} {`⟨*remaining default parameters*⟩`}`.

```
\def\optparams@#1#2#3{%
  \@ifnextchar[{%
    \optparams@@{#1}{#2}#3\@nil%
  }{%
    #1#2#3%
  }%
}
```

`\optparams@@` Reads the next optional parameter from the input (as `#5` or ⟨*new parameter*⟩). Then gobbles up the corresponding default parameter (as `#3` or ⟨*default for parameter read*⟩, appends the parameter read to ⟨*parameters read*⟩ and calls `\optparams@` again to check for more optional parameters.

Usage: `\optparams@@ {`⟨*macro*⟩`} {`⟨*parameters read*⟩`} {`⟨*default for parameter read*⟩`} {`⟨*remaining default parameters*⟩`} {`⟨*new parameter*⟩`}`.

```
\def\optparams@@#1#2[#3]#4\@nil[#5]{%
  \optparams@{#1}{#2[#5]}{#4}%
```

```
13 }
```

Now this is quite a coincidence: Thirteen lines of code, and this package was begun on Friday, 13th ...

Index

Numbers written in italic refer to the page where the corresponding entry is described, the ones underlined to the code line of the definition, the rest to the code lines where the entry is used.