The xoptarg Package

Josselin Noirel
http://www.jnoirel.fr/

22nd April 2007 (v1.0)

Abstract

Because they rely on the `\futurelet` primitive, the macros with optional arguments cannot be expandable. However, it is possible to make them expandable if there is at least one mandatory argument (see 'Limitations').

Contents

Introduction

Normal macros are expandable:

```
\newcommand*{\cmda}[1]{1. #1 2. #1 3. #1 4. #1}
\newcommand*{\cmdb}[3]{[#1#2#3 #1#3#2 #2#1#3 #2#3#1 #3#1#2 #3#2#1]}
```

'Expandable' means that if you print them on the terminal with `\typeout`

```
\typeout{\cmda{Test}...and \cmdb{T}{h}{e} rest}
```

it will result in the processed message which is printed during the compilation:

```
1. Test 2. Test 3. Test 4. Test...and [The Teh hTe heT eTh ehT] rest
```

But, commands defined to take an optional argument behave differently. The code:

```
\newcommand*{\cmdc}[2][X](#1,#2)
\typeout{\cmdc[A]{B} \cmdc{C}}
```

prints the unprocessed

```
\cmdc[A]{B} \cmdc{C}
```

and not `(A,B) (X,C)`! (Of course, in typesetting context, it doesn't make any difference... the interest of the package lies in the expandability in moving-argument contexts.) The xoptarg helps you to make your macros expandable even when they take an optional argumen (see 'Limitations' though).

Usage

Defining commands

Your macros, including those that take an optional argument can be expandable provided they also take at least one mandatory argument[1]. Practically spoken, your command, when it takes an optional argument, must take at least two arguments in total[2].

Load the package with

```
\usepackage{xoptarg}
```

then, instead of using `\newcommand`, use `\xnewcommand`

`\xnewcommand*{`⟨*command*⟩`}[`*n*`][`⟨*default*⟩`]{`⟨*definition*⟩`}`

This is the same syntax as `\newcommand`'s and it works even if the command does take any optional argument. A normal, non-expandable, macro is defined when you define a macro that takes one optional argument and no mandatory one. A warning is issued to inform you that it won't be possible to make it expandable[3].

Changing the prefix

This package uses newcommand that allows a more sophisticated syntax:

`\xnewcommand[`⟨*prefix*⟩`]{`⟨*command*⟩`}[`*n*`][`⟨*default*⟩`]{`⟨*definition*⟩`}`

For instance, to define a macro globally, use

`\xnewcommand[\global]{`⟨*command*⟩`}{`⟨*definition*⟩`}`

(But `\newcommand` would do the job as well, as newcommand improves it also.)

Limitations

The limitations of the package are the following:

- To be expandable, the macros with an optional argument must have at least one mandatory argument.
- The mandatory arguments must be surrounded by braces in this case.
- There is no `\rexnewcommand` command.

[1] If your macros don't take any optional argument, the standard behaviour prevails so that you don't to have to worry about them. The 'at least one mandatory argument' condition applies only when the macro takes an optional argument.

[2] A second requirement is that the mandatory arguments be surrounded by explicit braces `{...}`.

[3] The command will be made robust using `\protected` if ε-TEX is available.