The xifthen package

Josselin Noirel

25th March 2006

Abstract

This package implements new commands to go within the first argument of `\ifthenelse` to test whether a string is void or not, if a command is defined or equivalent to another. It includes also the possibility to make use of the complex expressions introduced by the package calc, together with the ability of defining new commands to handle complex tests. This package requires the ε-TeX features.

1 General syntax

The general syntax is inherited of that of the package ifthen:

`\ifthenelse{`⟨*test expression*⟩`}{`⟨*true code*⟩`}{`⟨*false code*⟩`}`

Evaluates the ⟨*test expression*⟩ and executes ⟨*true code*⟩ if the test turns out to be true and ⟨*false code*⟩ otherwise. ifthen provides the following tests:

⟨*value* 1⟩ `=` ⟨*value* 2⟩

⟨*value* 1⟩ `<` ⟨*value* 2⟩

⟨*value* 1⟩ `>` ⟨*value* 2⟩ Simple tests on integer comparisons.

`\isodd{`⟨*number*⟩`}` Is ⟨*number*⟩ odd?

`\isundefined`⟨*command*⟩ Id ⟨*command*⟩ undefined?

`\equal{`⟨*string* 1⟩`}{`⟨*string* 2⟩`}` Are ⟨*string* 1⟩ and ⟨*string* 2⟩ equivalent (after expansion)?

`\boolean{`⟨*boolean*⟩`}` Does the boolean ⟨*boolean*⟩ hold the value *true* or *false*?

`\lengthtest{`⟨*dimen* 1⟩ `=` ⟨*dimen* 2⟩`}`

`\lengthtest{`⟨*dimen* 1⟩ `<` ⟨*dimen* 2⟩`}`

`\lengthtest{`⟨*dimen* 1⟩ `>` ⟨*dimen* 2⟩`}` Simple dimension comparisons.

`\(...\)` Parenthesis.

`\AND`

`\OR`

`\NOT` Conjunction, disjunction, negation.

2 New tests

`\isnamedefined{`⟨*command name*⟩`}`

Returns *true* if the command \⟨*command name*⟩ is defined.

`\isempty{`⟨*content*⟩`}`

Returns *true* is ⟨*content*⟩ is empty (in the sense used by ifmtarg which is used internally). It is essentially equivalent to `\equal{`⟨*content*⟩`}{}` except that the argument of `\isempty` isn't expanded and therefore isn't affected by fragile commands.

`\isequivalentto{`⟨*command* 1⟩`}{`⟨*command* 2⟩`}`

Corresponds to the `\ifx` test: it returns *true* when the two commands are exactly equivalent (same definition, same number of arguments, etc., otherwise *false* is returned.

`\cnttest{`⟨*counter expression* 1⟩`}`⟨*comparison*⟩`{`⟨*counter expression* 2⟩`}`

Compares the two counter expressions (having the usual syntax of the package calc) and returns the value of the test. The comparison can be one of the following characters <, >, and =.

`\dimtest{`⟨*dimen expression* 1⟩`}`⟨*comparison*⟩`{`⟨*dimen expression* 2⟩`}`

Compares the two dimension expressions (having the usual syntax of the package calc) and returns the value of the test. The comparison can be one of the following characters <, >, and =.

3 Defining new complex test commands

`\newtest{`⟨*command*⟩`}[`⟨*n*⟩`]{`⟨*test expression*⟩`}`

Defines a command named ⟨*command*⟩ taking *n* arguments (no optional argument is allowed) consisting of the test as specified by ⟨*test expression*⟩ that can be used in the argument of `\ifthenelse`. For instance, if we want to test whether a rectangle having dimensions l and L meets the two following conditions: $S = l \times L > 100$ and $P = 2(l + L) < 60$:

```
\newtest{\sillytest}[2]{%
  \cnttest{(#1)*(#2)}>{100}%
  \AND
  \cnttest{((#1)+(#2))*2}<{60}%
}
```

Then `\ifthenelse{\sillytest{14}{7}}{TRUE}{FALSE}` returns FALSE because $14 \times 7 = 98$ and $2 \times (14 + 7) = 42$, while `\ifthenelse{\sillytest{11}{11}}{TRUE}{FALSE}` returns TRUE because $11 \times 11 = 121$ and $2 \times (11 + 11) = 44$.