The xifthen package

Josselin Noirel*

http://www.jnoirel.fr/

17th April 2009

Abstract

This package implements new commands that can be used within the first argument of ifthen’s \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.

Contents

What’s new 1

ifthen’s interface 1

Declaring and setting booleans . . . . . . . . . . . . . . . . . . . . . . . . . . . 1
Executing conditional code . . . . . . . . . . . . . . . . . . . . . . . . . . . 2

New tests 3

Examples 4

What’s new

1.1 Now \cnttest and \dimtest accept <= and >=.

• I renamed \terminateswith in \endswith.

1.2 Corrected a bug related to a bad interaction between new tests and ifthen’s replacement macro (credits go to MPG & P. Albarède).

1.3 Removed a spurious space (thanks to Ulrike Fisher).

ifthen’s interface

Declaring and setting booleans

You can declare boolean (presumably in the preamble of your document) with

\newboolean{\langle boolean\rangle}

\*This document corresponds to version v1.3 (2009/04/17) of xifthen.sty.
where \(\textit{boolean}\) is a name made up of alphanumeric characters. For instance,

\[
\text{\texttt{\textbackslash newboolean\{appendix\}}} \\
\text{\texttt{\textbackslash newboolean\{first\}}}
\]

Then your boolean is ready to be set with

\[
\text{\texttt{\textbackslash setboolean\{\textit{boolean}\}\{\textit{truth value}\}}}
\]

where \(\textit{truth value}\) can be \texttt{true} or \texttt{false}.

### Executing conditional code

The general syntax is inherited of that of the package \texttt{ifthen}:

\[
\text{\texttt{\textbackslash ifthenelse\{\textit{test expression}\}\{\textit{true code}\}\{\textit{false code}\}}}
\]

Evaluates the \(\textit{test expression}\) and executes \(\textit{true code}\) if the test turns out to be true and \(\textit{false code}\) otherwise. \texttt{ifthen} provides the tests explained in the next paragraphs.

### Value of a boolean

You can use the value of a boolean you declared, or the value of a primitive boolean of \TeX

\[
\text{\texttt{\textbackslash boolean\{\textit{boolean}\}}}
\]

### Tests on integers

To test whether an integer is equal to, strictly less than, or strictly greater than, you write the expression straightforwardly.

\[
\begin{align*}
\text{\texttt{\{value\}_1} &= \text{\texttt{\{value\}_2}} \\
\text{\texttt{\{value\}_1} &< \text{\texttt{\{value\}_2}} \\
\text{\texttt{\{value\}_1} &> \text{\texttt{\{value\}_2}} \\
\text{\texttt{\textbackslash isodd\{\{number\}\}}} \\
\end{align*}
\]

### Tests on lengths

There exist similar tests for the lengths, but you need in this case to surround the whole expression with \texttt{\textbackslash lengthtest}.

\[
\begin{align*}
\text{\texttt{\textbackslash lengthtest\{\texttt{\{dimen\}_1} = \texttt{\{dimen\}_2}\}}} \\
\text{\texttt{\textbackslash lengthtest\{\texttt{\{dimen\}_1} < \texttt{\{dimen\}_2}\}}} \\
\text{\texttt{\textbackslash lengthtest\{\texttt{\{dimen\}_1} > \texttt{\{dimen\}_2}\}}} \\
\end{align*}
\]

### Tests on commands

You can test if a command is undefined.

\[
\text{\texttt{\textbackslash isundefined \{command\}}} \\
\]

### Tests on character strings

You want to know whether two character strings are equal? Use:

\[
\text{\texttt{\textbackslash equal\{\{string\}_1}\{\{string\}_2\}}} \\
\]

Remark that the two arguments are fully expanded. In other words, it is the result of the expansion of the macros that is compared. This behaviour also entails a moving argument and you should protect fragile command to avoid bizarre errors.

---

1. The primitive booleans include: \texttt{\textbackslash mmode} (Are we in math mode?), \texttt{\textbackslash hmode} (Are we in horizontal mode?), \texttt{\textbackslash vmode} (Are we in vertical mode?), etc.
2. This test differs from \texttt{\textbackslash ifundefined} in that it takes a real command—and not a command name—as argument, and also in that command which is let equal to \texttt{\textbackslash relax} is not considered undefined.
3. Practically, the fact that the content is expanded, means that if the macro \texttt{\textbackslash bar} is defined as \texttt{\textbackslash baz\{\texttt{\textbackslash bar}\}} and the command \texttt{\textbackslash baz} is defined as \texttt{\textbackslash f\textbackslash s\textbackslash h\textbackslash s\textbackslash i\textbackslash d\textbackslash a\textbackslash n}, then \texttt{\textbackslash equal\{\texttt{\textbackslash bar}\}\{\texttt{foo}\}} turns out to be true, because \texttt{\textbackslash bar} eventually leads to \texttt{\textbackslash foo}. This is usually the wanted behaviour.
Building more elaborated expressions  You can build more sophisticated expressions using the `\AND` (conjunction), `\OR` (disjunction), and `\NOT` (negation) operators.

\( \langle \text{expression}_1 \rangle \ \AND \ \langle \text{expression}_2 \rangle \)
\( \langle \text{expression}_1 \rangle \ \OR \ \langle \text{expression}_2 \rangle \)
\( \\NOT \ \langle \text{expression} \rangle \)

The evaluation is lazy, meaning that if you write
\( \langle \text{expression}_1 \rangle \ \AND \ \langle \text{expression}_2 \rangle \)
then \( \langle \text{expression}_2 \rangle \) won’t be evaluated if \( \langle \text{expression}_1 \rangle \) is true.

There is not precedence rules: the argument is read from left to right and \( \\NOT \) applies to the very next test. When the precedence must be changed you can use the parentheses:
\( \\NOT \ \langle \text{expression} \rangle \)

New tests

After this brief review of ifthen’s principles, we introduce the new tests provided by xifthen.

Tests on integers  One of the drawback of \TeX’s tests and of ifthen’s as well, is the impossibility to use calc’s syntax in it. The \numexpr primitive of \e-\TeX somehow allows to overcome this difficulty but it is not well documented and normal users are certainly more familiar with the capabilities offered by calc. The xifthen package allows to use calc-valid expressions via the new test \cnttest. The syntax is as follows:
\[
\cnttest\{\langle \text{counter expression}_1 \rangle\}\{\langle \text{comparison} \rangle\}\{\langle \text{counter expression}_2 \rangle\}
\]
It evaluates the two counter expressions, compares them, and returns the value of the test. The comparison can be one of the following sequences \(<\>, \>=\), or \(\leq\).

Tests on lengths  The similar test has been designed for the lengths and dimensions:
\[
\dimtest\{\langle \text{dimen expression}_1 \rangle\}\{\langle \text{comparison} \rangle\}\{\langle \text{dimen expression}_2 \rangle\}
\]
It evaluates the two dimension expressions, compares them, and returns the value of the test. The comparison can be one of the following sequences \(<\>, \>=\), or \(\leq\).

Tests on commands  We define a companion of \isundefined that uses a command name rather than a command
\[
\isnamedefined\{\langle \text{command name} \rangle\}
\]

4 Lowercase versions of these commands also exist but we advise the user to stick to the uppercase ones because \or is part of \TeX’s syntax.

5 The evil is in the details: ifthen works by reading its argument twice. The tests are evaluated on the second pass, but and expansion is performed on the first one, regardless of the truth value. So that, quite surprisingly, the following test fails:
\[
\\NOT \ \isundefined\{\foo\} \ \AND \ \equal\{\foo\}\{\langle \text{string} \rangle\}
\]

6 If you are stuck with the distinction between ‘command’ and ‘command name’, let me explain it further with an example: the command name of the command \foo is foo. This is sometimes more convenient to use the command name than the name. Still, this functionality is probably intended more for experienced programmers who want to use the niceties of ifthen and xifthen.
Returns true if the command \(\text{(command name)}\) is defined\(^7\).

Sometimes, you need to compare two macros \texttt{\textbackslash foo} and \texttt{\textbackslash bar} and test whether they are actually the same macro.

\texttt{\textbackslash isequivalentto\{\text{(command\_1)}\}\{\text{(command\_2)}\}}

Corresponds to the \texttt{\textbackslash if x test} test: it returns true when the two commands are exactly equivalent (same definition, same number of arguments, same prefixes, etc., otherwise \texttt{false} is returned).

**Tests on character strings**  Very often, we see people using \texttt{\textbackslash equal\{#1\}{} in their command definitions (for instance, to test whether an optional argument had been passed to their macro). A more efficient test can be used:

\texttt{\textbackslash isempty\{}\text{(content)}\}}

Returns true if \texttt{\text{(content)}} is empty. It is essentially equivalent to \texttt{\textbackslash equal\{}\text{(content)}\}{} except that the argument of \texttt{\textbackslash isempty} isn’t expanded and therefore isn’t affected by fragile commands\(^8\).

It is possible to test whether a substring appears within another string\(^9\).

\texttt{\textbackslash isin\{}\text{(substring)}\}\{\text{(string)}\}

Sometimes, you need to check whether a string ends with a particular substring. This can be achieved using\(^9\)

\texttt{\textbackslash endswith\{}\text{(string)}\}\{\text{(substring)}\}

**Building more elaborated expressions**  It is then possible to create new tests with:

\texttt{\textbackslash newtest\{}\text{(command)}\}\{\text{(n)}\}\{\text{(test expression)}\}

Surprisingly, a simple \texttt{\textbackslash newcommand} would not work. The \texttt{\textbackslash newtest} macro defines a command \texttt{\text{(command)}} taking \(n\) arguments (no optional argument is allowed consisting of the test as specified by \texttt{\text{(test expression)}} that can be used in the argument of \texttt{\textbackslash ifthenelse}.

**Examples**

Let’s illustrate the most important features of \texttt{\textbackslash ifthen} with the following problem: if we want to test whether a rectangle having dimensions \(l\) and \(L\) meets the two following conditions: \(S = l \times L > 100\) and \(P = 2(l + L) < 60\):

\begin{verbatim}
\newtest\{}\text{(condition)}\}\{2\}\{}\{\text{(test expression)}\}
\\\\cnttest\{}\{(#1)\}*(#2)\}\{100\}\%\ AND\ \cnttest\{}\{(#1)\}+(#2)\}*2\}\{60\}\%
\end{verbatim}

Then \texttt{\textbackslash ifthenelse\{}\text{(condition\{14\}\{7\})\}\{TRUE\}\{FALSE\}} returns \texttt{FALSE} because \(14 \times 7 = 98\) and \(2 \times (14 + 7) = 42\), while \texttt{\textbackslash ifthenelse\{}\text{(condition\{11\}\{11\})\}\{TRUE\}\{FALSE\}} returns \texttt{TRUE} because \(11 \times 11 = 121\) and \(2 \times (11 + 11) = 44\).

\(^7\) Uses \texttt{\textbackslash ifcsname...\textbackslash endcsname} internally and not \texttt{\textbackslash ifundefined}.

\(^8\) Internally, it uses \texttt{\textbackslash unexpanded} and \texttt{\textbackslash ifmtarg}.

\(^9\) Uses \texttt{\textbackslash in@} and \texttt{\textbackslash ifin@} internally.

\(^{10}\) For compatibility reasons, there exist a command unfortunately called \texttt{\textbackslash terminateswith} that performs the same test but it is deprecated.

\(^{11}\) No optional argument is allowed because the macro needs to be expanded in the first pass and that optional arguments avoid that.

\(^{12}\) Note that, because within the arguments of \texttt{\textbackslash cnttest} the \texttt{\textbackslash calc} is used, you must use real parentheses (and, and, \texttt{\textbackslash and}) and not \texttt{\textbackslash ( and \textbackslash )}.
Now a list of typical uses of xifthen’s capabilities:

\begin{verbatim}
4 - 1 < 4: true  4 < 4: false  4 + 1 < 4: false 
4 - 1 ≤ 4: true  4 ≤ 4: true  4 + 1 ≤ 4: false 
4 - 1 = 4: false  4 = 4: true  4 + 1 = 4: false 
4 - 1 ≥ 4: false  4 ≥ 4: true  4 + 1 ≥ 4: true 
4 - 1 > 4: false  4 > 4: false  4 + 1 > 4: true


\ifthenelse{\isempty{}}{true}{false} true
\ifthenelse{\isempty{ }}{true}{false} true
\ifthenelse{\isempty{ foo }}{true}{false} false
\ifthenelse{\endswith{foo.}{.}}{true}{false} true
\ifthenelse{\endswith{foo!}{.}}{true}{false} false
\ifthenelse{\isin{foo}{foobar}}{true}{false} true
\ifthenelse{\isin{Foo}{foobar}}{true}{false} false
\ifthenelse{\cnttest{10 * 10 + 1}>{100}}{true}{false} true
\ifthenelse{\cnttest{10 * 10 + 1}>{100 * 100}}{true}{false} false
\ifthenelse{\isequivalentto{\usepackage}{\RequirePackage}}{true}{false} true
\ifthenelse{\isequivalentto{\usepackage}{\textit}}{true}{false} false
\ifthenelse{\isnamedefined{@foo}}{true}{false} false
\ifthenelse{\isnamedefined{@for}}{true}{false} true
\end{verbatim}