The **boolexpr*** package

Purely expandable boolean expressions and switch (ε-TEX).

Florent Chervet <florent.chervet@free.fr>

July 22, 2009

Abstract

boolexpr provides a purely expandable way to evaluate boolean expressions of the form:

α \AND β \OR γ ...

where α, β and γ are *atomic expressions* of one of those 8 valid forms:

$x = y$ | $x <> y$[1] | $x < y$ | $x <= y$ | $x > y$ | $x >= y$

\if⟨*test*⟩ 0\else 1\fi | another \boolexpr evaluation

where x and y are either numeric expressions (or dimensions, glue, muglue to test using \dimexpr, \glueexpr or \muexpr – please refer to the **\boolexpr** examples) and ⟨*test*⟩ may be a switch (\iftrue / \iffalse or a conditional[2]). boolexpr abide by the precedence of \AND on \OR, and the whole expression is evaluated until the result is known (in other words, \AND and \OR are *shortcut* boolean operators).

\boolexpr will expand to **0** if the expression is **true**, making it proper to work with \ifcase.

Furthermore, boolexpr defines a \switch syntax which remains purely expandable.

The boolexpr package is designed to work with an ε-TEX distribution of LATEX: it is based on the ε-TEX \numexpr primitive and requires no other package.

Contents

---

* boolexpr: CTAN:macros/latex/contrib/boolexpr

This documentation is produced with the DocStrip utility.

⟶ To get the documentation, run (thrice): `pdflatex boolexpr.dtx`
for the index: `makeindex -s gind.ist boolexpr.idx`
⟶ To get the package, run: `etex boolexpr.dtx`

The .dtx file is embedded into this pdf file thank to embedfile by H. Oberdiek.

1. The choice of <> rather than != is due to Category codes considerations.
2. \if, \ifcase, \ifcat, \ifcsname, \ifdefined, \ifdim, \ifeof, \iffontchar, \ifhmode, \ifinner, \ifmmode, \ifnum, \ifodd, \ifvmode, \ifvoid, \ifx

1 Introduction – Using boolexpr : \boolexpr and \switch

\boolexpr{⟨*boolean expression*⟩}

\boolexpr is a macro that takes for unique argument a series of *atomic expressions* of the form:

numeric expr.	=	numeric expr.
numeric expr.	<>	numeric expr.
numeric expr.	<	numeric expr
numeric expr.	<=	numeric expr
numeric expr.	>	numeric expr.
numeric expr.	>=	numeric expr.
\if	⟨*test*⟩	0 \else 1\fi

\boolexpr{⟨*boolean expression*⟩}

related by **\AND** or **\OR** (with the usual logical precedence).

\boolexpr expands to 0 if the whole expression is true and to a non nul number if the whole expression is false.

\boolexpr is **purely expandable**.

Therefore, testing may be used as follow:

```
\ifcase\boolexpr{ boolean expression }
     what to do if true
\else
     what to do if false
\fi
```

It is possible to use switches as boolean quantities into a \boolexpr expression with the syntax:
\ifswitch 0\else 1\fi

It is also possible to use \ifdim, \ifnum etc. (although it is not necessary because other forms of atomic expression can perform those tests more easily) and \ifdefined, \ifcsname etc. with the same syntax, f.ex.:
\ifcsname ⟨*cs-name*⟩ \endcsname 0\else 1\fi

It means that if the conditional is true then the *atomic expression* is true (expands to 0), otherwise the *atomic expression* is false (expands to non 0).

It is possible to test dimensions (or glue or muglue) by writing \dimexpr (or \glueexpr or \muexpr) in front of the *atomic expression*; therefore, the following are valid atomic expressions:

\dimexpr	dimen expr.	<	dimen expr
\glueexpr	glue expr.	<>	glue expr.
\muexpr	mu expr.	=	mu expr.

It is allowed to group expressions inside the argument of \boolexpr by inserting another \boolexpr evaluation, f.ex.:
\boolexpr{ \boolexpr{ α \OR β } \AND γ }

The logical **NOT** operator can be achieved by writing for example:
\ifcase\boolexpr{⟨*boolean expression*⟩} 1\else 0\fi

Finally, if the ⟨*boolean expression*⟩ is missing:
\boolexpr{ } expands to 1 (*ie.* false).

\ifboolexpr{⟨*boolean expression*⟩}{⟨*true part*⟩}{⟨*false part*⟩}

\ifboolexpr is the LaTeX form of a \boolexpr test.

\ifboolexpr is purely expandable (provided ⟨*true part*⟩ and ⟨*false part*⟩ are so).

\boolexpr examples

The part of the expression that is evaluated is in blue (the remainder is not evaluated).

```
\ifcase\boolexpr{ 45 > 80 \OR 5<>5 \AND  5<4 }
      boolexpr is true
\else boolexpr is false                          ⟶ boolexpr is false
\fi
```

```
\ifcase\boolexpr{ 45 < 80 \OR 5 = 5 \AND  0<>0 }
      boolexpr is true
\else boolexpr is false                          ⟶ boolexpr is true
\fi
```

```
\ifcase\boolexpr{ \boolexpr{ 45 < 80 \OR 5 = 5 } \AND  0<>0 }
      boolexpr is true
\else boolexpr is false                          ⟶ boolexpr is false
\fi
```

```
\ifcase\boolexpr{ 12>0 \AND (4+3)*5 > 20 }
     boolexpr is true
\else boolexpr is false                          ⟶ boolexpr is true
\fi
```

```
\makeatletter
\number\boolexpr{ \the\catcode`\@=11 }           ⟶ 0
                                                 (catcode of character @ is 11)
```

```
\makeatother
\number\boolexpr{ \the\catcode`\@=11 \AND \ifdefined\@undefined 0\else 1\fi }
                                                 ⟶ 1
                                                 (catcode of character @ is 12)
```

```
\makeatletter
\number\boolexpr{ 3<4 \AND \@ifundefined{iftest}{1}{\iftest 0\else 1\fi} }
                                                 ⟶ 1: \iftest not defined
```

```
\makeatletter   \newif\iftest   \testtrue
\number\boolexpr{ 3<4 \AND \@ifundefined{iftest}{1}{\iftest 0\else 1\fi} }
                                                 ⟶ 0: \iftest is true
```

```
\ifcase\boolexpr{ \dimexpr 12pt + 1in > 8mm * 2 \AND \iftest 0\else 1\fi }
      boolexpr is true
\else  boolexpr is false                         ⟶ boolexpr is true
\fi
```

```
\ifcase\boolexpr{ 0=0 \AND \ifcase\boolexpr{1=1 \AND 5<=5} 1\else 0\fi }
      boolexpr is true                           ⟶ boolexpr is false
\else  boolexpr is false                         α \AND NOT( β \AND γ )
\fi                                              = α \AND NOT β \OR α \AND NOT γ
```

Results in green were evaluated by boolexpr.sty at compilation time.

The \switch syntax

```
\switch
\case{⟨boolean expression⟩}  ...
\case{⟨boolean expression⟩}  ...
\otherwise  ...
\endswitch
```

boolexpr defines a syntax for \switch conditionals which remains purely expandable:

Each part of the switch is optional. That means:

```
\switch
\case{ ⟨bool expr⟩ } ...
\case{ ⟨beel expr⟩ } ...
\case{ ⟨bool expr⟩ } ...
\endswitch
```

```
\switch
\otherwise ...
\endswitch
```

```
\switch
\endswitch
```

are allowed by boolexpr.

\switch examples

The part of the expression that is evaluated is in blue (the remainder is not evaluated).

```
\switch
\case{6>1 \AND 6<=5}$\geq 1$ and $\leq 5$%
\case{3<10}$> 5$ and $< 10$%
\case{3>10}$\geq 10$%
\endswitch
```

⟶ > 5 and < 10

```
\edef\result{%
\switch
\case{6>1 \AND 6<=5}$\geq 1$ and $\leq 5$%
\case{3<10}$> 5$ and $< 10$%
\case{3>10}$\geq 10$%
\endswitch}
```

⟶ result:
>5 and < 10

```
\def\result#1{\textbf{%
\switch
\case{\pdfstrcmp{#1}{one}}one%
\case{\pdfstrcmp{#1}{two}}two%
\otherwise something else%
\endswitch}}

\protected@edef\result{\result{12}}
```

⟶ result:\protect
\textbf { something else }

Results in green were evaluated by boolexpr.sty at compilation time.

2 Implementation

2.1 The algorithm

The *string* is the suite of *atomic expressions* connected by \AND or \OR.
The *result* must be 0 if the *string* is true, and non zero if the *string* is false.
"*go to* some macro" means: "*now expand* some macro".

A \bex@OR

1) Split the *string* into two parts:
   #1 = before the first \OR (#1 does not contain any \OR)
   #2 = after the first \OR

2) If #2 is blank: the *string* contains no \OR
   then go to \bex@AND to test \AND relations in #1
   Otherwise: test the \AND relations in #1 and keep #2 in a so called "OR-buffer" for further testing.

B \bex@AND
#1 = OR-buffer for further testing if needed

1) Split the string "before the first \OR" (*ie.* the #1 of A.1) into two parts:
   #2 = before the first \AND (#2 is an *atomic expression*)
   #3 = after the first \AND (#3 does not contain any \OR)

2) Then test #2 (the *atomic expression*):

TRUE: If #3 is blank then #2 is either:

- an atomic expression alone
- the last atomic expression in *string*, preceded by \OR
- an atomic expression preceded by \OR and followed by \OR

In each of these 3 cases, the whole expression (*ie.* the *string*) is true because #2 is true (otherwise, we had known the result of the whole *string* earlier, and were not into testing #2)
Now if #3 is not blank then #2 is followed by \AND:
go to \bex@ANDAND to test the series of \AND

FALSE: if the OR-buffer #1 is blank then #2 is either:

- an atomic expression alone
- an atomic expression followed a series of \AND (and no \OR)
- the last atomic expression of the *string*

In each or these 3 cases, the whole expression (*ie.* the *string*) is false because #2 is false (otherwise, the result would have been known earlier)
Now if the OR-buffer #1 is not blank, then we have to do more tests to get the result:
go to \bex@OR to split the OR-buffer (#1 here) and continue testing...

C \bex@ANDAND
#1 is the OR-buffer for further testing if needed

1) Split the string (*ie.* #3 in B.2.TRUE) into two parts:
   #2 : before the first \AND (#2 is an *atomic expression*)
   #3 : after the first \AND

2) Test the *atomic expression* #2:

TRUE: If #3 is blank then #2 is the last atomic expression of a series of \AND (possibly followed by \OR).
Conclusion: the whole *string* is true (otherwise, we would have concluded earlier that it was false and were not into testing #2... think about it)
Now if #3 is not blank then #2 is followed by \AND and we have to test further:
go to \bex@ANDAND to test #3.

FALSE: we do not have to test the following \AND: the \AND-connected series is false.
If the OR-buffer #1 is blank then the whole *string* is false.
Now if the OR-buffer #1 is not blank: continue testing into this OR-buffer :
go to \bex@OR.

2.2 Category codes considerations

At first glance, the author of this package wanted to test inequality with the operator !=. A problem arose because some languages make the character ! active (f.ex. french). As far as babel changes the catcodes \AtBeginDocument, the category code of ! is different in the preamble (12) than in the document (13).

After all, it was possible to change the definitions after begin document but... if you try to make the = character active, you will (surprisingly) observe that a test like:

```
\ifnum 4=4 ok\fi
```

leads you to one of the following error messages:

```
undefined control sequence =      if = is undefined
missing = inserted for \ifnum     if = is defined.
```

The same apply for < or >. Therefore: such conditionals are possible for TEX only if =, < and > have a category code of 12 (11 is forbidden too).

Thus the choice of <> is far easier and more reliable than the c-like !=.

2.3 Identification

This package is intended to use with a LATEX distribution of ε-TEX.

```
1 ⟨*package⟩
2 \ProvidesPackage{boolexpr}
3   [2009/08/12 v2.1 Purely expandable boolean expressions and switch (eTeX)]
```

2.4 Special catcode

The colon (:) will be used as a delimiter. We give it a category code of 4.

```
4 \let\bex@AtEnd\@empty
5 \def\TMP@EnsureCode#1#2{%
6   \edef\bex@AtEnd{%
7     \bex@AtEnd
8     \catcode#1 \the\catcode#1\relax
9   }%
10  \catcode#1 #2\relax
11 }
12 \TMP@EnsureCode{42}{4}% :
```

2.5 Helper macros

While reading the log file it is preferable to read \@firstoftwo/\@secondoftwo when the algorithm is making a choice (\ifblank) and \bex@truepart/bex@falsepart when the algorithm has just determined the result of an atomic expression.

```
\let\bex@truepart\@firstoftwo
\let\bex@falsepart\@secondoftwo

\def\bex@afterelse#1\else#2\fi{\fi#1}
```

\bex@ifblank The following macro is purely expandable and its code is most probably due to D. Arseneau (url.sty)

```
\begingroup\catcode`\|=3
\long\gdef\bex@ifblank#1{\bex@ifbl@nk#1||\@secondoftwo\@firstoftwo:}
\long\gdef\bex@ifbl@nk#1#2|#3#4#5:{#4}
\endgroup
```

\bex@ifblankdef It's just the same but expand once the control sequence given as an argument.

```
\long\def\bex@ifblankdef#1{\expandafter\bex@ifblank\expandafter{#1}}
```

2.6 Atomic expression evaluation

The six possible numeric atomic expressions $x < y$, $x <= y$, $x > y$, $x >= y$, $x <> y$ and $x = y$ are first transformed to their zero-form:
\numexpr $x - y < 0$, \numexpr $x - y > 0$, \numexpr $x - y <> 0$, \numexpr $x - y = 0$ etc.

Before all, we need to know which relation is used in the atomic expression:

\bex@rel \bex@rel tests an *atomic expression*: first determine its type (inferior to, superior to, equality, inequality, other \boolexpr) and then use the appropriate evaluation macro:

```
\def\bex@rel#1{%
  \bex@test@eval#1:
    {\bex@eval{#1}}
    {\bex@test@neq#1<>:
      {\bex@test@infeq#1<=:
        {\bex@test@inf#1<:
          {\bex@test@supeq#1>=:
            {\bex@test@sup#1>:
              {\bex@test@eq#1=:
                {\@latex@error{Unknown relation found while scanning
                  \noexpand\boolexpr!}\@ehd}
                {\bex@eq#1:}}
              {\bex@sup#1:}}
            {\bex@supeq#1:}}
          {\bex@inf#1:}}
        {\bex@infeq#1:}}
      {\bex@neq#1:}}}
```

The test macros They test each *atomic expression* in order to determine its type:

```
\def\bex@test@neq#1<>#2:{\bex@ifblank{#2}}
\def\bex@test@eq#1=#2:{\bex@ifblank{#2}}
\def\bex@test@infeq#1<=#2:{\bex@ifblank{#2}}
\def\bex@test@inf#1<#2:{\bex@ifblank{#2}}
\def\bex@test@supeq#1>=#2:{\bex@ifblank{#2}}
\def\bex@test@sup#1>#2:{\bex@ifblank{#2}}
```

```
\def\bex@test@eval#1#2:{%
   \ifcat\noexpand#1\relax% #1 is a control sequence
         \bex@afterelse\bex@test@Eval{#1}
   \else \expandafter\@secondoftwo
   \fi}
\def\bex@test@Eval#1{\csname @%
   \ifx#1\the second%
   \else\ifx#1\numexpr second%
   \else\ifx #1\number second%
   \else\ifx #1\dimexpr second%
   \else\ifx #1\glueexpr second%
   \else\ifx #1\muexpr second%
   \else first%
   \fi\fi\fi\fi\fi\fi oftwo\endcsname}
```

Evaluation macros

They evaluate each *atomic expression* according to its type:

```
\def\bex@true@or@false#1{\csname bex@%
   \ifnum\numexpr#1 true\else false\fi part\endcsname}
\def\bex@false@or@true#1{\csname bex@%
   \ifnum\numexpr#1 false\else true\fi part\endcsname}

\def\bex@eq#1=#2:{\bex@true@or@false{#1-(#2)=0}}
\def\bex@neq#1<>#2:{\bex@false@or@true{#1-(#2)=0}}
\def\bex@infeq#1<=#2:{\bex@false@or@true{#1-(#2)>0}}
\def\bex@inf#1<#2:{\bex@true@or@false{#1-(#2)<0}}
\def\bex@supeq#1>=#2:{\bex@false@or@true{#1-(#2)<0}}
\def\bex@sup#1>#2:{\bex@true@or@false{#1-(#2)>0}}
\def\bex@eval#1{\bex@true@or@false{#1=0}}
```

2.7 \AND and \OR management

\bex@OR

\bex@OR splits the string to evaluate into two parts: before the first \OR and after:

```
\def\bex@OR#1\OR#2:{%
      \bex@ifblank{#2}
         {\bex@AND{}#1\AND:}
         {\expandafter\bex@AND\expandafter{\bex@removeOR#2:}#1\AND:}}
```

bex@removeOR

Just remove the extra \OR: inserted into the argument of \bex@OR:

```
\def\bex@removeOR#1\OR:{#1}
```

\bex@AND

\bex@AND splits the string to evaluate into two parts: before the first \AND and after:

```
\def\bex@AND#1#2\AND#3:{%
   \bex@rel{#2}
      {\bex@ifblank{#3}
         {0}
         {\bex@ANDAND{#1}#3:}}
      {\bex@ifblank{#1}{+1}{\bex@OR#1 \OR:}}}
```

\bex@ANDAND

\bex@ANDAND evaluate successive *atomic expressions* related by \AND until false is found or until the end if every expression is true:

```
\def\bex@ANDAND#1#2\AND#3:{%
   \bex@rel{#2}
      {\bex@ifblank{#3}{+0}{\bex@ANDAND{#1}#3:}}
      {\bex@ifblank{#1}{+1}{\bex@OR#1 \OR:}}}
```

\boolexpr `\boolexpr` is the entry point for evaluating boolean expressions:

```
\newcommand\boolexpr[1]{\bex@ifblank{#1}{1}{\numexpr\bex@OR#1 \OR:+0}}
```

\@first@or@second

```
\newcommand\bex@first@or@second[1]{%
   \csname bex@\ifnum\boolexpr{#1}=0 true\else false\fi part\endcsname}
```

\ifboolexpr `\ifboolexpr` is the LaTeX form of `\boolexpr` tests:

```
\newcommand\ifboolexpr{}
\let\ifboolexpr\bex@first@or@second
```

\switch `\switch` is not long to implement... see:

```
\def\switch#1\endswitch{\bex@ifblank{#1}{}%
   {\bex@switch@otherwise#1\otherwise\endswitch}}

\def\bex@switch@otherwise#1\otherwise#2\endswitch{%
   \bex@switch@case#1\case\endswitch
      {\bex@ifblank{#2}{}{\bex@otherwise#2\endswitch}}
      \endswitch}

\def\bex@switch@case#1\case#2\endswitch{%
   \bex@ifblank{#1}
      {\bex@ifblank{#2}{\@firstoftwo}{\bex@switch@case#2\endswitch}}
      {\case#1\endcase
      {\bex@ifblank{#2}{\@firstoftwo}{\bex@switch@case#2\endswitch}}%
      }}

\def\case#1#2\endcase{%
   \ifboolexpr{#1}{\bex@after@endswitch{#2}}}

\def\bex@after@endswitch#1#2\endswitch{#1}
\def\bex@otherwise#1\otherwise#2\endswitch{#1}
```

Restore original catcode:

```
\bex@AtEnd
⟨/package⟩
```

2.8 Future developments : to do

boolexpr should work either with ε-TEX or ε-TEX-LaTeX...

May be build a “real” `\NOT` operator.

3 History

[2009/08/12 v2.1]

- Added the `\switch` syntax
- Small bug (`\numexpr` forgotten in the “list of exceptions” (`\bex@test@Eval`)
- Redesigned tests for better compilation

[2009/07/22 v1.0]

- First version.

4 Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.