The **boolexpr**[*] package

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

<florent.chervet@free.fr>

2010/04/15– v3.14

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.

Be aware that `\boolexpr` (a little like `\numexpr`) works only if its argument is purely expandable; the same for `\switch`. If you wish a more general `\CASE` syntax refer to this excellent paper: http://www.tug.org/TUGboat/Articles/tb14-1/tb38fine.pdf.

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
\fi
```

⟶ boolexpr is false

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

⟶ boolexpr is true

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

⟶ boolexpr is false

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

⟶ boolexpr is true

```
\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
\fi
```

⟶ boolexpr is true

```
\ifcase\boolexpr{ 0=0 \AND \ifcase\boolexpr{1=1 \AND 5<=5} 1\else 0\fi }
      boolexpr is true
\else  boolexpr is false
\fi
```

⟶ boolexpr is false
α \AND NOT(β \AND γ)
= α \AND NOT β \OR α \AND NOT γ

Results in green were evaluated by boolexpr 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⟩ } ...
\otherwise ...
\endswitch
```

```
\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 $< 10$`

```
\newcounter{myCounter} \setcounter{myCounter}{2}
\edef\result{%
\switch[\value{myCounter}=]
\case{1}one%
      |  ---------------------------->   |
\case{2}two% <=> \case{value{myCounter}=2}
\case{3}three%
\case{2}vartwo%never found%
\otherwise something else%
\endswitch}
```

⟶ `result: two`

```
switch[\value{myCounter}]
\case{=1}It's $1$%
\case{=-1}It's $-1$%
                |-|  ------------------------->  |-|
\case{>=0}It's $>=0$% <=> \case{\value{myCounter}>=0}
\otherwise something else%
\endswitch
```

⟶ It's >= 0

```
switch[\pdfstrcmp{DuMmY}]
\case{{First}}It's "First"%
                |-|  ------------------------->  |-|
\case{{DuMmY}It's DuMmY%
\otherwise something else%
\endswitch
```

⟶ It's "DuMmY"

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

1.1 Purely expandable macros for tests with boolexpr

Please refer to the etextools package documentation at :
http://www.ctan.org/tex-archive/macros/latex/contrib/etextools/etextools.pdf

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`".

$\boxed{\text{A}}$ `\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.

$\boxed{\text{B}}$ `\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...

$\boxed{\text{C}}$ `\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.

```
⟨*package⟩
\ProvidesPackage{boolexpr}
   [2010/04/15 v3.14 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 8 (as in etextools):

```
\let\bex@AtEnd\@empty
\def\TMP@EnsureCode#1#2{%
  \edef\bex@AtEnd{%
    \bex@AtEnd
    \catcode#1 \the\catcode#1\relax
  }%
  \catcode#1 #2\relax
}
\TMP@EnsureCode{95}{11}% _
\TMP@EnsureCode{47}{8}% / etextool delimiter
\TMP@EnsureCode{60}{12}% <
\TMP@EnsureCode{61}{12}% =
\TMP@EnsureCode{62}{12}% >
\TMP@EnsureCode{43}{12}% -
```

```
\TMP@EnsureCode{45}{12}% +
\TMP@EnsureCode{58}{8}% : delimitor
```

2.5 Tree 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
```

`\bex@nbk`

The following macro is purely expandable and its code is most probably due to D. Arseneau (url.sty). `\bex@nbk` means if **n**ot **b**lan**k**.

```
\long\def\bex@nbk#1#2/#3#4#5//{#4}
```

`\bex@ifoptchar`

`\bex@ifoptchar` checks if a character is a single opening bracket ‘`[`‘.

```
\long\def\bex@ifoptchar#1[#2/#3#{\csname @\if @\detokenize{#1#2}@%
   first\else second\fi oftwo\endcsname}
```

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:

```
\long\def\bex@rel#1{%
   \bex@test_eval#1/{\bex@eval{#1}}
      {\bex@test_neq#1<>//{\bex@neq #1/}
         {\bex@test_infeq#1<=//{\bex@infeq #1/}
            {\bex@test_inf#1<//{\bex@inf #1/}
               {\bex@test_supeq#1>=//{\bex@supeq #1/}
                  {\bex@test_sup#1>//{\bex@sup #1/}
                     {\bex@test_eq#1=//{\bex@eq #1/}
                        {\@latex@error{Unknown relation found while scanning
                           \noexpand\boolexpr!}\@ehd}//}//}//}//}//}//}//}}
```

`The test macros`

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

```
\def\bex@test_neq#1<>#2/{\bex@nbk#2/}
\def\bex@test_eq#1=#2/{\bex@nbk #2/}
\def\bex@test_infeq#1<=#2/{\bex@nbk #2/}
\def\bex@test_inf#1<#2/{\bex@nbk #2/}
\def\bex@test_supeq#1>=#2/{\bex@nbk #2/}
\def\bex@test_sup#1>#2/{\bex@nbk #2/}
\long\def\bex@test_eval#1#2/{%
   \ifcat\noexpand#1\relax% #1 is a control sequence
         \bex@test_Eval{#1}
   \else \expandafter\@secondoftwo
   \fi}
\long\def\bex@test_Eval#1#2\fi{\fi\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\ifx #1\value second%
   \else first%
   \fi\fi\fi\fi\fi\fi\fi oftwo\endcsname}
```

`Evaluation macros` They evaluate each *atomic expression* according to its type:

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

```
\long\def\bex@OR#1\OR#2:{\bex@AND{#2}#1\AND:}
```

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

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

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

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

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

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

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

```
\ifdefined\ifboolexpr% etoolbox defines ifboolexpr...
\PackageWarning{boolexpr}{\string\ifboolexpr\space has been defined before\MessageBreak
    by etoolbox (I suppose) - Overwritting}
\renewcommand\ifboolexpr[1]{\bex@true_or_false{\boolexpr{#1}=0}}
\else
\newcommand\ifboolexpr[1]{\bex@true_or_false{\boolexpr{#1}=0}}
\fi
```

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

```
\long\def \switch#1\endswitch {\bex@nbk#1//{\bex@switch_opt#1\endswitch}{}//}
\long\def \bex@switch_opt#1#2\endswitch{\bex@ifoptchar#1/[/
      {\bex@switch_opti#1#2\endswitch}{\bex@switch_opti[]#1#2\endswitch}}%]
\def \bex@switch_opti[#1]#2\endswitch {\bex@switch_otherwise[{#1}]#2\otherwise\endswitch}

\def\bex@switch_otherwise[#1]#2\otherwise#3\endswitch{%
   \bex@switch_case[{#1}]#2\case\endswitch
      {\bex@nbk#3//{\bex@otherwise#3\endswitch}{}//}
      \endswitch}

\def\bex@switch_case[#1]#2\case#3\endswitch{\bex@nbk#2//%
      {\bex@case[{#1}]#2\endcase%
         {\bex@nbk#3//{\bex@switch_case[{#1}]#3\endswitch}\@firstoftwo//}}%
      {\bex@nbk#3//{\bex@switch_case[{#1}]#3\endswitch}\@firstoftwo//}//}

\long\def\bex@case[#1]#2#3\endcase{\ifboolexpr{#1#2}{\bex@after_endswitch{#3}}}

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

2.7.1 Purely expandable macros for tests with boolexpr

`\bex@pdfmatch`

```
\long\def\bex@pdfmatch#1#2{\ifnum\pdfmatch{#2}{#1}=0 1\else0\fi}
```

`\bex@ifempty`

```
\long\def\bex@ifempty#1{\if\relax\detokenize{#1}\relax0\else1\fi}
\long\def\bex_ifempty#1{\csname @\if\relax\detokenize{#1}\relax first\else second\fi oftwo
```

`\bex@ifblank`

```
\long\def\bex@ifblank#1{\bex@nbk#1//10//}
```

`\bex@ifx`

```
\long\def\bex@ifx#1#2{\bex__ifx#1#2//}
\long\def\bex_ifx#1#2#3/#4#5#6//{\bex@nbk#6//{\ifx#1#2\bex_else#5\else\bex_fi#6\fi}{#5}//}
\long\def\bex_else#1\else#2\fi{\fi#1}
\long\def\bex_fi#1\fi{\fi#1}
```

`\bex@comp`

```
\long\def\bex@comp#1{\bex@ifoptchar#1/[/\bex@c@mp{\bex@c@mp@[\numexpr]}}
\long\def\bex@c@mp[#1#2]#3#4#5{%
   \bex_ifempty{#2}{%
      \ifx #1\dimexpr         \bex@c@mp@\ifdim\dimexpr{#3}{#4}{#5}%
      \else\ifx #1\numexpr    \bex@c@mp@\ifnum\numexpr{#3}{#4}{#5}%
      \else\ifx #1\glueexpr   \bex@c@mp@\ifdim\glueexpr{#3}{#4}{#5}%
      \else\ifx #1\muexpr     \bex@c@mp@\ifdim\muexpr{#3}{#4}{#5}%
      \else\ifx #1\number     \bex@c@mp@\ifnum\numexpr{#3}{#4}{#5}%
      \else\PackageError{boolexpr}{%
         Invalid comparison test while scanning \string\bex@comp\MessageBreak
         found: \detokenize{#1}}%
      \fi\fi\fi\fi\fi}%
      {\PackageError{boolexpr}{Invalid comparison test while scanning \string\bex@comp\Mes
      found: \detokenize{#1}}}}
\long\def\bex@c@mp@#1#2#3#4#5{#1#2#3#4#5 0\else 1\fi}
```

```
\bex@AtEnd\let\bex@AtEnd\@undefined
⟨/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

[2010/04/15 v3.14]

- etoolbox now defines a `\ifboolexpr` macro (not purely expandable).
  Fix has been done (with a warning) to be able to use `\ifboolexpr` from boolexpr.

[2009/09/30 v3.1]

- Support of `\pdfmatch` added (`\bex@pdfmatch`)

[2009/09/03 v3.0 – ε-TeX- and XeTeX- stable]

- Many bug fixed in `\switch`. Tested on LaTeX, pdfLaTeX and XeLaTeX.
- Revision of this pdf documentation.

[2009/08/31 v2.9]

- Added `\value` in the “list of exceptions” (`\bex@test_Eval`) Enhancement of `\switch` with the optional first argument (refer to the examples).

[2009/08/13 v2.2]

- Small optimisation in `\bex@OR`

[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.