

# The etoolbox package

An e-TeX toolbox for class and package authors

Philipp Lehman  
plehman@gmx.net

Version 1.6  
June 22, 2008

## Contents

I	Introduction . . . . .	I	3	Author commands . . . . .	4
1.1	About . . . . .	I	3.1	Definitions . . . . .	4
1.2	License . . . . .	I	3.2	Expansion control . . . . .	7
1.3	Feedback . . . . .	I	3.3	Hook management . . . . .	8
1.4	Acknowledgments . . . . .	2	3.4	Patching . . . . .	10
2	User commands . . . . .	2	3.5	Generic tests . . . . .	12
2.1	Definitions . . . . .	2	3.6	Boolean switches . . . . .	14
2.2	Patching . . . . .	2	3.7	List processing . . . . .	16
2.3	Lengths and counters . . . . .	2	3.8	Miscellaneous tools . . . . .	19
2.4	Predefined hooks . . . . .	3	4	Revision history . . . . .	19

## 1 Introduction

### 1.1 About

The etoolbox package is a toolbox of programming facilities geared primarily towards LaTeX class and package authors. It provides LaTeX frontends to some of the new primitives provided by e-TeX as well as some generic tools which are not related to e-TeX but match the profile of this package. The package is work in progress. Note that the initial versions of this package were released under the name elatex.

### 1.2 License

Copyright © 2007–2008 Philipp Lehman. Permission is granted to copy, distribute and/or modify this software under the terms of the LaTeX Project Public License, version 1.3.<sup>1</sup> This package is author-maintained.

### 1.3 Feedback and contributions

I started to work on this package when I found myself implementing the same tools and shorthands I had employed in previous LaTeX packages for yet another package. For the most part, the facilities provided by etoolbox address my needs as a package author and future development is likely to be guided by these needs as well. Please note that I will not be able to address any feature requests. However, I am open to contributions by other class and package authors, provided that the contributed code is sufficiently generic, matches the profile of this package, and may be added to the package without requiring significant adaption.

<sup>1</sup> <http://www.ctan.org/tex-archive/macros/latex/base/lppl.txt>

## 1.4 Acknowledgments

The `\ifblank` test of this package is based on code by Donald Arseneau.

## 2 User commands

The facilities in this section are geared towards regular users as well as class and package authors.

### 2.1 Definitions

```
\newrobustcmd{<command>}[<arguments>][<optarg default>]{<replacement text>}  
\newrobustcmd*{<command>}[<arguments>][<optarg default>]{<replacement text>}
```

The syntax and behavior of this command is similar to `\newcommand` except that the newly defined `<command>` will be robust. The behavior of this command differs from the `\DeclareRobustCommand` command from the LaTeX kernel in that it issues an error rather than just an informational message if the `<command>` is already defined. Since it uses e-TeX's low-level protection mechanism rather than the corresponding higher-level LaTeX facilities, it does not require an additional macro to implement the 'robustness'.

```
\renewrobustcmd{<command>}[<arguments>][<optarg default>]{<replacement text>}  
\renewrobustcmd*{<command>}[<arguments>][<optarg default>]{<replacement text>}
```

The syntax and behavior of this command is similar to `\renewcommand` except that the redefined `<command>` will be robust.

```
\providerobustcmd{<command>}[<arguments>][<optarg default>]{<replacement text>}  
\providerobustcmd*{<command>}[<arguments>][<optarg default>]{<replacement text>}
```

The syntax and behavior of this command is similar to `\providecommand` except that the newly defined `<command>` will be robust. Note that this command will provide a robust definition of the `<command>` only if it is undefined. It will not make an already defined `<command>` robust.

### 2.2 Patching

```
\robustify{<command>}
```

Modifies a `<command>` defined with `\newcommand` such that it is robust without altering its parameters, its prefixes, or its replacement text. If the `<command>` has been defined with `\DeclareRobustCommand`, this will be detected automatically. LaTeX's high-level protection mechanism will be replaced by the corresponding low-level e-TeX facility in this case.

### 2.3 Length and counter assignments

The facilities in this section are replacements for `\setcounter` and `\setlength` which support arithmetic expressions.

```
\defcounter{<counter>}{<integer expression>}
```

Assigns a value to a LaTeX `<counter>` previously initialized with `\newcounter`. This

command is similar in concept and syntax to `\setcounter` except for two major differences. 1) The second argument may be an *integer expression* which will be processed with `\numexpr`. The *integer expression* may be any arbitrary code which is valid in this context. The value assigned to the *counter* will be the result of that calculation. 2) In contrast to `\setcounter`, the assignment is local by default but `\defcounter` may be prefixed with `\global`. The functional equivalent of `\setcounter` would be `\global\defcounter`.

`\deflength{<length>}{<dimen expression>}`

Assigns a value to a *length* register previously initialized with `\newlength`. This command is similar in concept and syntax to `\setlength` except that the second argument may be a *dimen expression* which will be processed with `\dimexpr`. The *dimen expression* may be any arbitrary code which is valid in this context. The value assigned to the *length* register will be the result of that calculation. The assignment is local by default but `\deflength` may be prefixed with `\global`. This command may be used as a drop-in replacement for `\setlength`.

## 2.4 Predefined all-purpose hooks

LaTeX provides two hooks which defer the execution of code either to the beginning or to the end of the document body. The `\AtBeginDocument` code is executed towards the beginning of the document body, *after* the main aux file has been read for the first time. The `\AtEndDocument` code is executed at the end of the document body, *before* the main aux file is read for the second time. The hooks introduced here are similar in concept but defer the execution of their *code* argument to slightly different locations. The *code* may be arbitrary TeX code. Parameter characters in the *code* argument need not be doubled. To sum that up, LaTeX will perform the following tasks ‘inside’ `\begin{document}`:

- Execute any `\AtEndPreamble` code
- Start initialization for document body
- Load the main aux file written on the previous LaTeX pass
- Open the main aux file for writing on the current pass
- Continue initialization for document body
- Execute any `\AtBeginDocument` code
- Complete initialization for document body
- Execute any `\AfterBeginDocument` code

Strictly speaking, `\AtBeginDocument` code is neither part of the preamble nor part of the document body but located in-between these parts of a LaTeX document because it gets executed in the middle of the initialization sequence performed prior to typesetting. `\AtEndPreamble` code, however, is part of the preamble; `\AfterBeginDocument` code is part of the body and may contain printable text to be typeset as the very first thing in the document. Inside `\end{document}`, LaTeX will perform the following tasks:

- Execute any `\AtEndDocument` code
- Close the main aux file for writing

- Load the main aux file written on the current LaTeX pass
- Perform final tests and issue warnings, if applicable
- Execute any `\AfterEndDocument` code

Any `\AtEndDocument` code may be considered as being part of the document body insofar as it is still possible to perform typesetting tasks and write to the main aux file when it gets executed. `\AfterEndDocument` code is not part of the document body. `\AtBeginDocument` and `\AtEndDocument` are provided by the LaTeX kernel. The `etoolbox` package introduces the following additional hooks:

`\AtEndPreamble{⟨code⟩}`

This hook differs from `\AtBeginDocument` in that the `⟨code⟩` is executed right at the end of the preamble, before the main aux file (as written on the previous LaTeX pass) is read and prior to any `\AtBeginDocument` code. It is sometimes desirable to defer code to the end of the preamble because all requested packages have been loaded at this point. However, `\AtBeginDocument` code is executed too late if it is required in the aux file.

`\AfterBeginDocument{⟨code⟩}`

This hook differs from `\AtBeginDocument` in that the `⟨code⟩` is executed as part of the document body, after any `\AtBeginDocument` code. It is possible to perform typesetting tasks such as printing a title page at this point.

`\AfterEndDocument{⟨code⟩}`

This hook differs from `\AtEndDocument` in that the `⟨code⟩` is executed at the very end of the document, after the main aux file (as written on the current LaTeX pass) has been read and after any `\AtEndDocument` code. This hook is useful if a package needs to evaluate any data in the aux file at the end of a LaTeX run.

`\AfterPreamble{⟨code⟩}`

This hook is a variant of `\AtBeginDocument` which may be used in both the preamble and the document body. When used in the preamble, it behaves exactly like `\AtBeginDocument`. When used in the document body, it immediately executes its `⟨code⟩` argument. `\AtBeginDocument` would issue an error in this case.

### 3 Author commands

The facilities in this section are geared towards class and package authors.

#### 3.1 Definitions

##### 3.1.1 Macro definitions

The facilities in this section are simple but frequently required shorthands which extend the scope of the `\@namedef` and `\@nameuse` macros from the LaTeX kernel.

`\csdef{<csname>}{<arguments>}{<replacement text>}`

Similar to the TeX primitive `\def` except that it takes a control sequence name as its first argument. This command is robust and corresponds to `\@namedef`.

`\csgdef{<csname>}{<arguments>}{<replacement text>}`

Similar to the TeX primitive `\gdef` except that it takes a control sequence name as its first argument. This command is robust.

`\csedef{<csname>}{<arguments>}{<replacement text>}`

Similar to the TeX primitive `\edef` except that it takes a control sequence name as its first argument. This command is robust.

`\csxdef{<csname>}{<arguments>}{<replacement text>}`

Similar to the TeX primitive `\xdef` except that it takes a control sequence name as its first argument. This command is robust.

`\protected@csedef{<csname>}{<arguments>}{<replacement text>}`

Similar to `\csedef` except that LaTeX's protection mechanism is temporarily enabled. To put it in other words: this command is similar to the LaTeX kernel command `\protected@edef` except that it takes a control sequence name as its first argument. This command is robust.

`\protected@csxdef{<csname>}{<arguments>}{<replacement text>}`

Similar to `\csxdef` except that LaTeX's protection mechanism is temporarily enabled. To put it in other words: this command is similar to the LaTeX kernel command `\protected@xdef` except that it takes a control sequence name as its first argument. This command is robust.

`\cslet{<csname>}{<command>}`

Similar to the TeX primitive `\let` except that the first argument is a control sequence name. This command is robust and may be prefixed with `\global`.

`\letcs{<command>}{<csname>}`

Similar to the TeX primitive `\let` except that the second argument is a control sequence name. This command is robust and may be prefixed with `\global`.

`\csletcs{<csname>}{<csname>}`

Similar to the TeX primitive `\let` except that both arguments are control sequence names. This command is robust and may be prefixed with `\global`.

`\csuse{<csname>}`

Takes a control sequence name as its argument and forms a control sequence token. This command differs from the `\@nameuse` macro from the LaTeX kernel in that it expands to an empty string if the control sequence is undefined.

`\undef` $\langle command \rangle$

Clears a  $\langle command \rangle$  such that e-TeX's `\ifdefined` and `\ifcsname` tests will consider it as undefined. This command is robust and may be prefixed with `\global`.

`\csundef` $\{\langle csname \rangle\}$

Similar to `\undef` except that it takes a control sequence name as its argument. This command is robust and may be prefixed with `\global`.

### 3.1.2 Arithmetic definitions

The facilities in this section permit calculations using macros rather than length registers and counters.

`\numdef` $\langle command \rangle \{ \langle integer expression \rangle \}$

Similar to `\edef` except that the  $\langle integer expression \rangle$  is processed with `\numexpr`. The  $\langle integer expression \rangle$  may be any arbitrary code which is valid in this context. The replacement text assigned to the  $\langle command \rangle$  will be the result of that calculation. If the  $\langle command \rangle$  is undefined, it will be initialized to 0 before the  $\langle integer expression \rangle$  is processed.

`\numgdef` $\langle command \rangle \{ \langle integer expression \rangle \}$

Similar to `\numdef` except that the assignment is global.

`\csnumdef` $\{\langle csname \rangle\} \{ \langle integer expression \rangle \}$

Similar to `\numdef` except that it takes a control sequence name as its first argument.

`\csnumgdef` $\{\langle csname \rangle\} \{ \langle integer expression \rangle \}$

Similar to `\numgdef` except that it takes a control sequence name as its first argument.

`\dimdef` $\langle command \rangle \{ \langle dimen expression \rangle \}$

Similar to `\edef` except that the  $\langle dimen expression \rangle$  is processed with `\dimexpr`. The  $\langle dimen expression \rangle$  may be any arbitrary code which is valid in this context. The replacement text assigned to the  $\langle command \rangle$  will be the result of that calculation. If the  $\langle command \rangle$  is undefined, it will be initialized to 0pt before the  $\langle dimen expression \rangle$  is processed.

`\dimgdef` $\langle command \rangle \{ \langle dimen expression \rangle \}$

Similar to `\dimdef` except that the assignment is global.

`\csdimdef` $\{\langle csname \rangle\} \{ \langle dimen expression \rangle \}$

Similar to `\dimdef` except that it takes a control sequence name as its first argument.

`\csdimgdef{⟨csname⟩}{⟨dimen expression⟩}`

Similar to `\dimgdef` except that it takes a control sequence name as its first argument.

`\gluedef⟨command⟩{⟨glue expression⟩}`

Similar to `\edef` except that the *⟨glue expression⟩* is processed with `\glueexpr`. The *⟨glue expression⟩* may be any arbitrary code which is valid in this context. The replacement text assigned to the *⟨command⟩* will be the result of that calculation. If the *⟨command⟩* is undefined, it will be initialized to 0pt plus 0pt minus 0pt before the *⟨glue expression⟩* is processed.

`\gluegdef⟨command⟩{⟨glue expression⟩}`

Similar to `\gluedef` except that the assignment is global.

`\csgluedef{⟨csname⟩}{⟨glue expression⟩}`

Similar to `\gluedef` except that it takes a control sequence name as its first argument.

`\csgluegdef{⟨csname⟩}{⟨glue expression⟩}`

Similar to `\gluegdef` except that it takes a control sequence name as its first argument.

`\mugdef⟨command⟩{⟨muglue expression⟩}`

Similar to `\edef` except that the *⟨muglue expression⟩* is processed with `\muexpr`. The *⟨muglue expression⟩* may be any arbitrary code which is valid in this context. The replacement text assigned to the *⟨command⟩* will be the result of that calculation. If the *⟨command⟩* is undefined, it will be initialized to 0mu before the *⟨muglue expression⟩* is processed.

`\mugdef⟨command⟩{⟨muglue expression⟩}`

Similar to `\mugdef` except that the assignment is global.

`\csmugdef{⟨csname⟩}{⟨muglue expression⟩}`

Similar to `\mugdef` except that it takes a control sequence name as its first argument.

`\csmugdef{⟨csname⟩}{⟨muglue expression⟩}`

Similar to `\mugdef` except that it takes a control sequence name as its first argument.

### 3.2 Expansion control

The facilities in this section are useful to control expansion in an `\edef` or a similar context.

`\expandonce` $\langle command \rangle$

This command expands a  $\langle command \rangle$  once and prevents further expansion of the replacement text.

`\csexpandonce` $\{\langle csname \rangle\}$

Similar to `\expandonce` except that it takes a control sequence name as its argument.

`\protecting` $\{\langle code \rangle\}$

This command applies LaTeX's protection mechanism, which normally requires prefixing each fragile command with `\protect`, to an entire chunk of arbitrary  $\langle code \rangle$ . Its behavior depends on the current state of `\protect`. Note that the braces around the  $\langle code \rangle$  are mandatory even if it is a single token.

### 3.3 Hook management

The facilities in this section are intended for hook management. A  $\langle hook \rangle$  in this context is a plain macro without any parameters and prefixes which is used to collect code to be executed later. These facilities may also be useful to patch simple macros by appending code to their replacement text. For more complex patching operations, see section 3.4. All commands in this section will initialize the  $\langle hook \rangle$  if it is undefined.

#### 3.3.1 Appending to a hook

The facilities in this section append arbitrary code to a hook.

`\appto` $\langle hook \rangle \{\langle code \rangle\}$

This command appends arbitrary  $\langle code \rangle$  to a  $\langle hook \rangle$ . If the  $\langle code \rangle$  contains any parameter characters, they need not be doubled. This command is robust.

`\gappto` $\langle hook \rangle \{\langle code \rangle\}$

Similar to `\appto` except that the assignment is global. This command may be used as a drop-in replacement for the `\g@addto@macro` command in the LaTeX kernel.

`\eappto` $\langle hook \rangle \{\langle code \rangle\}$

This command appends arbitrary  $\langle code \rangle$  to a  $\langle hook \rangle$ . The  $\langle code \rangle$  is expanded at definition-time. Only the new  $\langle code \rangle$  is expanded, the current replacement text of the  $\langle hook \rangle$  is not. This command is robust.

`\xappto` $\langle hook \rangle \{\langle code \rangle\}$

Similar to `\eappto` except that the assignment is global.

`\protected@eappto` $\langle hook \rangle \{\langle code \rangle\}$

Similar to `\eappto` except that LaTeX's protection mechanism is temporarily enabled.



`\protected@xappto⟨hook⟩{⟨code⟩}`

Similar to `\xappto` except that LaTeX's protection mechanism is temporarily enabled.

`\csappto{⟨csname⟩}{⟨code⟩}`

Similar to `\appto` except that it takes a control sequence name as its first argument.

`\csgappto{⟨csname⟩}{⟨code⟩}`

Similar to `\gappto` except that it takes a control sequence name as its first argument.

`\cseappto{⟨csname⟩}{⟨code⟩}`

Similar to `\eappto` except that it takes a control sequence name as its first argument.

`\csxappto{⟨csname⟩}{⟨code⟩}`

Similar to `\xappto` except that it takes a control sequence name as its first argument.

`\protected@cseappto⟨hook⟩{⟨code⟩}`

Similar to `\protected@eappto` except that it takes a control sequence name as its first argument.

`\protected@csxappto⟨hook⟩{⟨code⟩}`

Similar to `\protected@xappto` except that it takes a control sequence name as its first argument.

### 3.3.2 Prepending to a hook

The facilities in this section ‘prepend’ arbitrary code to a hook, i. e., the code is inserted at the beginning of the hook rather than being added at the end.

`\preto⟨hook⟩{⟨code⟩}`

Similar to `\appto` except that the `⟨code⟩` is prepended.

`\gpreto⟨hook⟩{⟨code⟩}`

Similar to `\preto` except that the assignment is global.

`\lepreto⟨hook⟩{⟨code⟩}`

Similar to `\eappto` except that the `⟨code⟩` is prepended.

`\xpreto⟨hook⟩{⟨code⟩}`

Similar to `\lepreto` except that the assignment is global.

`\protected@epreto<hook>{<code>}`

Similar to `\epreto` except that LaTeX's protection mechanism is temporarily enabled.

`\protected@xpreto<hook>{<code>}`

Similar to `\xpreto` except that LaTeX's protection mechanism is temporarily enabled.

`\cspreto{<cname>}{<code>}`

Similar to `\preto` except that it takes a control sequence name as its first argument.

`\csgpreto{<cname>}{<code>}`

Similar to `\gpreto` except that it takes a control sequence name as its first argument.

`\csepreto{<cname>}{<code>}`

Similar to `\epreto` except that it takes a control sequence name as its first argument.

`\csxpreto{<cname>}{<code>}`

Similar to `\xpreto` except that it takes a control sequence name as its first argument.

`\protected@csepreto<hook>{<code>}`

Similar to `\protected@epreto` except that it takes a control sequence name as its first argument.

`\protected@csxpreto<hook>{<code>}`

Similar to `\protected@xpreto` except that it takes a control sequence name as its first argument.

### 3.4 Patching

The facilities in this section are useful to hook into or modify existing code. All commands presented here preserve the parameters and the prefixes of the patched `<command>`. Note that `\outer` commands may not be patched. Also note that the commands in this section will not automatically issue any error messages if patching fails. Instead, they take a `<failure>` argument which should provide suitable fallback code or an error message. Issuing `\tracingpatches` in the preamble will cause the commands to write debugging information to the transcript file.

`\patchcmd[<prefix>]{<command>}{<search>}{<replace>}{<success>}{<failure>}`

This command extracts the replacement text of a `<command>`, replaces `<search>` with `<replace>`, and reassembles the `<command>`. The pattern match is category code agnostic and matches the first occurrence of the `<search>` pattern in the replacement text of the `<command>` to be patched. Note that the patching process

involves detokenizing the replacement text of the  $\langle command \rangle$  and retokenizing it under the current category code regime after patching. The category code of the @ sign is temporarily set to 11. If the replacement text of the  $\langle command \rangle$  includes any tokens with non-standard category codes, the respective category codes must be adjusted prior to patching. If the code to be replaced or inserted refers to the parameters of the  $\langle command \rangle$  to be patched, the parameter characters need not be doubled. If an optional  $\langle prefix \rangle$  is specified, it replaces the prefixes of the  $\langle command \rangle$ . An empty  $\langle prefix \rangle$  argument strips all prefixes from the  $\langle command \rangle$ . The assignment is local. This command implicitly performs the equivalent of an `\ifpatchable` test prior to patching. If this test succeeds, the command applies the patch and executes  $\langle success \rangle$ . If the test fails, it executes  $\langle failure \rangle$  without modifying the original  $\langle command \rangle$ . This command is robust.

`\ifpatchable`{ $\langle command \rangle$ }{ $\langle search \rangle$ }{ $\langle true \rangle$ }{ $\langle false \rangle$ }

This command executes  $\langle true \rangle$  if the  $\langle command \rangle$  may be patched with `\patchcmd` and if the  $\langle search \rangle$  pattern is found in its replacement text, and  $\langle false \rangle$  otherwise. This command is robust.

`\ifpatchable*`{ $\langle command \rangle$ }{ $\langle true \rangle$ }{ $\langle false \rangle$ }

Similar to `\ifpatchable` except that the starred variant does not require a search pattern. Use this version to check if a command may be patched with `\apptocmd` and `\pretocmd`.

`\apptocmd`{ $\langle command \rangle$ }{ $\langle code \rangle$ }{ $\langle success \rangle$ }{ $\langle failure \rangle$ }

This command appends  $\langle code \rangle$  to the replacement text of a  $\langle command \rangle$ . If the  $\langle command \rangle$  is a parameterless macro, it behaves like `\appto` from section 3.3.1. In contrast to `\appto`, `\apptocmd` may also be used to patch commands with parameters. In this case, it will detokenize the replacement text of the  $\langle command \rangle$ , apply the patch, and retokenize it under the current category code regime. The category code of the @ sign is temporarily set to 11. The  $\langle code \rangle$  may refer to the parameters of the  $\langle command \rangle$ . The assignment is local. If patching succeeds, this command executes  $\langle success \rangle$ . If patching fails, it executes  $\langle failure \rangle$  without modifying the original  $\langle command \rangle$ . This command is robust.

`\pretocmd`{ $\langle command \rangle$ }{ $\langle code \rangle$ }{ $\langle success \rangle$ }{ $\langle failure \rangle$ }

This command is similar to `\apptocmd` except that the  $\langle code \rangle$  is inserted at the beginning of the replacement text of the  $\langle command \rangle$ . If the  $\langle command \rangle$  is a parameterless macro, it behaves like `\preto` from section 3.3.1. In contrast to `\preto`, `\pretocmd` may also be used to patch commands with parameters. In this case, it will detokenize the replacement text of the  $\langle command \rangle$ , apply the patch, and retokenize it under the current category code regime. The category code of the @ sign is temporarily set to 11. The  $\langle code \rangle$  may refer to the parameters of the  $\langle command \rangle$ . The assignment is local. If patching succeeds, this command executes  $\langle success \rangle$ . If patching fails, it executes  $\langle failure \rangle$  without modifying the original  $\langle command \rangle$ . This command is robust.

`\tracingpatches` Enables tracing for all patching commands, including `\ifpatchable`. The debugging information will be written to the transcript file. This is useful if the reason why a patch is not applied or `\ifpatchable` yields `\false` is not obvious. This command must be issued in the preamble.

### 3.5 Generic tests

#### 3.5.1 Control sequence tests

`\ifdef`{*<control sequence>*}{*<true>*}{*<false>*}

Expands to *<true>* if the *<control sequence>* is defined, and to *<false>* otherwise. Note that control sequences will be considered as defined even if their meaning is `\relax`. This command is a LaTeX wrapper for the e-TeX primitive `\ifdefined`.

`\ifcsdef`{*<csname>*}{*<true>*}{*<false>*}

Similar to `\ifdef` except that it takes a control sequence name as its first argument. This command is a LaTeX wrapper for the e-TeX primitive `\ifcsname`.

`\ifundef`{*<control sequence>*}{*<true>*}{*<false>*}

Expands to *<true>* if the *<control sequence>* is undefined, and to *<false>* otherwise. Apart from reversing the logic of the test, this command also differs from `\ifdef` in that commands will be considered as undefined if their meaning is `\relax`.

`\ifcsundef`{*<csname>*}{*<true>*}{*<false>*}

Similar to `\ifundef` except that it takes a control sequence name as its first argument. This command may be used as a drop-in replacement for the `\@ifundefined` test in the LaTeX kernel.

`\ifdefmacro`{*<control sequence>*}{*<true>*}{*<false>*}

Expands to *<true>* if the *<control sequence>* is defined and is a macro, and to *<false>* otherwise.

`\ifcsmacro`{*<csname>*}{*<true>*}{*<false>*}

Similar to `\ifdefmacro` except that it takes a control sequence name as its first argument.

`\ifdefparam`{*<control sequence>*}{*<true>*}{*<false>*}

Expands to *<true>* if the *<control sequence>* is defined and is a macro with one or more parameters, and to *<false>* otherwise.

`\ifcsparam`{*<csname>*}{*<true>*}{*<false>*}

Similar to `\ifdefparam` except that it takes a control sequence name as its first argument.

`\ifdefprefix`{*<control sequence>*}{*<true>*}{*<false>*}

Expands to *<true>* if the *<control sequence>* is defined and is a macro prefixed with

`\long` and/or `\protected`, and to `\false` otherwise. Note that `\outer` macros may not be tested.

`\ifcsprefix`{*<csname>*}{*<true>*}{*<false>*}

Similar to `\ifdefprefix` except that it takes a control sequence name as its first argument.

`\ifdefprotected`{*<control sequence>*}{*<true>*}{*<false>*}

Expands to `\true` if the *<control sequence>* is defined and is a macro prefixed with `\protected`, and to `\false` otherwise.

`\ifcsprotected`{*<csname>*}{*<true>*}{*<false>*}

Similar to `\ifdefprotected` except that it takes a control sequence name as its first argument.

`\ifdefltxprotect`{*<control sequence>*}{*<true>*}{*<false>*}

Executes `\true` if the *<control sequence>* is defined and is a LaTeX protection shell, and `\false` otherwise. This command is robust. It will detect commands which have been defined with `\DeclareRobustCommand` or by way of a similar technique.

`\ifcsltxprotect`{*<csname>*}{*<true>*}{*<false>*}

Similar to `\ifdefltxprotect` except that it takes a control sequence name as its first argument.

`\ifdefempty`{*<control sequence>*}{*<true>*}{*<false>*}

Expands to `\true` if the *<control sequence>* is defined and is a parameterless macro whose replacement text is empty, and to `\false` otherwise. In contrast to `\ifx`, this test ignores the prefixes of the *<command>*.

`\ifcsempty`{*<csname>*}{*<true>*}{*<false>*}

Similar to `\ifdefempty` except that it takes a control sequence name as its first argument.

`\ifdefvoid`{*<control sequence>*}{*<true>*}{*<false>*}

Expands to `\true` if the *<control sequence>* is undefined, is a macro whose meaning is `\relax`, or is a parameterless macro whose replacement text is empty, and to `\false` otherwise.

`\ifcsvoid`{*<csname>*}{*<true>*}{*<false>*}

Similar to `\ifdefvoid` except that it takes a control sequence name as its first argument.

`\ifdequal`{*<control sequence>*}{*<control sequence>*}{*<true>*}{*<false>*}

Compares two control sequences and expands to `\true` if they are equal in the sense of `\ifx`, and to `\false` otherwise. In contrast to `\ifx`, this test will also yield `\false` if both control sequences are undefined or have a meaning of `\relax`.

`\ifcsequal{⟨csname⟩}{⟨csname⟩}{⟨true⟩}{⟨false⟩}`

Similar to `\ifdefequal` except that it takes control sequence names as arguments.

`\ifdefstring{⟨command⟩}{⟨string⟩}{⟨true⟩}{⟨false⟩}`

Compares the replacement text of a `⟨command⟩` to a `⟨string⟩` and executes `⟨true⟩` if they are equal, and `⟨false⟩` otherwise. Neither the `⟨command⟩` nor the `⟨string⟩` is expanded in the test and the comparison is category code agnostic. Control sequence tokens in the `⟨string⟩` argument will be detokenized and treated as strings. This command is robust. Note that it will only consider the replacement text of the `⟨command⟩`. For example, this code

```
\long\def\@gobbletwo#1#2{}
\ifdefstring{\@gobbletwo}{}{true}{false}
```

would yield `⟨true⟩`. The prefix and the parameters of `\@gobbletwo` are ignored.

`\ifcsstring{⟨csname⟩}{⟨string⟩}{⟨true⟩}{⟨false⟩}`

Similar to `\ifdefstring` except that it takes a control sequence name as its first argument.

### 3.5.2 String tests

`\ifstrequal{⟨string⟩}{⟨string⟩}{⟨true⟩}{⟨false⟩}`

Compares two strings and executes `⟨true⟩` if they are equal, and `⟨false⟩` otherwise. The strings are not expanded in the test and the comparison is category code agnostic. Control sequence tokens in any of the `⟨string⟩` arguments will be detokenized and treated as strings. This command is robust.

`\ifstrempy{⟨string⟩}{⟨true⟩}{⟨false⟩}`

Expands to `⟨true⟩` if the `⟨string⟩` is empty, and to `⟨false⟩` otherwise. The `⟨string⟩` is not expanded in the test.

`\ifblank{⟨string⟩}{⟨true⟩}{⟨false⟩}`

Expands to `⟨true⟩` if the `⟨string⟩` is blank (empty or spaces), and to `⟨false⟩` otherwise. The `⟨string⟩` is not expanded in the test.

## 3.6 Boolean switches

This package provides two interfaces to boolean switches which are completely independent of each other. The facilities in section 3.6.1 are a LaTeX frontend to `\newif`. Those in section 3.6.2 use a different mechanism.

### 3.6.1 TeX switches

Since the facilities in this section are based on `\newif` internally, they may be used to test and alter the state of switches previously defined with `\newif`. They are also compatible with the boolean tests of the `ifthen` package. The `\newif` approach requires a total of three macros per switch.

`\newbool{⟨name⟩}`

Defines a new boolean switch called *⟨name⟩*. If the switch has already been defined, this command issues an error. The initial state of newly defined switches is `false`. This command is robust.

`\providebool{⟨name⟩}`

Defines a new boolean switch called *⟨name⟩* unless it has already been defined. This command is robust.

`\booltrue{⟨name⟩}`

Sets the boolean switch *⟨name⟩* to `true`. This command is robust and may be prefixed with `\global`. It will issue an error if the switch is undefined.

`\boolfalse{⟨name⟩}`

Sets the boolean switch *⟨name⟩* to `false`. This command is robust and may be prefixed with `\global`. It will issue an error if the switch is undefined.

`\setbool{⟨name⟩}{⟨value⟩}`

Sets the boolean switch *⟨name⟩* to *⟨value⟩* which may be either `true` or `false`. This command is robust and may be prefixed with `\global`. It will issue an error if the switch is undefined.

`\ifbool{⟨name⟩}{⟨true⟩}{⟨false⟩}`

Expands to *⟨true⟩* if the state of the boolean switch *⟨name⟩* is `true`, and to *⟨false⟩* otherwise. If the switch is undefined, this command issues an error.

`\notbool{⟨name⟩}{⟨not true⟩}{⟨not false⟩}`

Similar to `\ifbool` but reverses the logic of the test.

### 3.6.2 LaTeX switches

In contrast to the switches from section 3.6.1, the facilities in this section require only one macro per switch. They also use a separate namespace to avoid name clashes with regular macros.

`\newtoggle{⟨name⟩}`

Defines a new boolean switch called *⟨name⟩*. If the switch has already been defined, this command issues an error. The initial state of newly defined switches is `false`. This command is robust.

`\providetoggle{⟨name⟩}`

Defines a new boolean switch called *⟨name⟩* unless it has already been defined. This command is robust.

`\toggletrue{⟨name⟩}`

Sets the boolean switch *⟨name⟩* to `true`. This command is robust and may be prefixed with `\global`. It will issue an error if the switch is undefined.

`\togglefalse{⟨name⟩}`

Sets the boolean switch *⟨name⟩* to `false`. This command is robust and may be prefixed with `\global`. It will issue an error if the switch is undefined.

`\settoggle{⟨name⟩}{⟨value⟩}`

Sets the boolean switch *⟨name⟩* to *⟨value⟩* which may be either `true` or `false`. This command is robust and may be prefixed with `\global`. It will issue an error if the switch is undefined.

`\iftoggle{⟨name⟩}{⟨true⟩}{⟨false⟩}`

Expands to *⟨true⟩* if the state of the boolean switch *⟨name⟩* is `true`, and to *⟨false⟩* otherwise. If the switch is undefined, this command issues an error.

`\nottoggle{⟨name⟩}{⟨not true⟩}{⟨not false⟩}`

Similar to `\iftoggle` but reverses the logic of the test.

### 3.7 List processing

#### 3.7.1 User input

The facilities in this section are primarily designed to handle user input. When building lists for internal use by a package, using the facilities in section 3.7.2 is preferable.

`\DeclareListParser{⟨command⟩}{⟨separator⟩}`

This command defines a list parser similar to the `\docsvlist` command below, which is defined like this:

```
\DeclareListParser{\docsvlist}{,}
```

Note that the list parsers are sensitive to the category code of the *⟨separator⟩*.

`\docsvlist{⟨item, item, ...⟩}`

This command loops over a comma-separated list and executes the auxiliary command `\do` for every item in the list, passing the item as an argument. In contrast to the `\@for` loop in the LaTeX kernel, `\docsvlist` is expandable. With a suitable definition of `\do`, lists may be processed inside an `\edef` or a comparable context. You may use `\listbreak` at the end of the replacement text of `\do` to stop processing and discard the remaining items in the list. Whitespace after list separators is ignored. If an item contains a comma or starts with a space, it must be wrapped in curly braces. The braces will be removed as the list is processed. Here is a usage example which prints a comma-separated list as an `itemize` environment:

```
\begin{itemize}
```



```

\renewcommand*\do}[1]{\item #1}
\docsvlist{item1, item2, {item3a, item3b}, item4}
\end{itemize}

```

Here is another example:

```

\renewcommand*\do}[1]{* #1\MessageBreak}
\PackageInfo{mypackage}{%
  Example list:\MessageBreak
  \docsvlist{item1, item2, {item3a, item3b}, item4}}

```

In this example, the list is written to the log file as part of an informational message. The list processing takes place during the `\write` operation.

### 3.7.2 Internal lists

The facilities in this section handle internal lists of data. An ‘internal list’ in this context is a plain macro without any parameters and prefixes which is employed to collect data. These lists use a special character as internal list separator. When processing user input in a list format, see the facilities in section 3.7.1.

`\listadd{<listmacro>}{<item>}`

This command appends an `<item>` to a `<listmacro>`.

`\listgadd{<listmacro>}{<item>}`

Similar to `\listadd` except that the assignment is global.

`\listead{<listmacro>}{<item>}`

Similar to `\listadd` except that the `<item>` is expanded at definition-time. Only the new `<item>` is expanded, the current list is not.

`\listxadd{<listmacro>}{<item>}`

Similar to `\listead` except that the assignment is global.

`\listcsadd{<listcsname>}{<item>}`

Similar to `\listadd` except that it takes a control sequence name as its first argument.

`\listcsgadd{<listcsname>}{<item>}`

Similar to `\listcsadd` except that the assignment is global.

`\listcseadd{<listcsname>}{<item>}`

Similar to `\listead` except that it takes a control sequence name as its first argument.

`\listcsxadd{<listcsname>}{<item>}`

Similar to `\listcseadd` except that the assignment is global.

`\dolistloop{<listmacro>}`

This command loops over all items in a `<listmacro>` and executes the auxiliary command `\do` for every item in the list, passing the item as an argument. The list loop itself is expandable. You may use `\listbreak` at the end of the replacement text of `\do` to stop processing and discard the remaining items in the list. Here is a usage example which prints an internal list called `\mylist` as an `itemize` environment:

```
\begin{itemize}
  \renewcommand*{\do}[1]{\item #1}
  \dolistloop{\mylist}
\end{itemize}
```

`\dolistcsloop{<listcsname>}`

Similar to `\dolistloop` except that it takes a control sequence name as its argument.

`\ifinlist{<item>}{<listmacro>}`

This command checks if an `<item>` is included in a `<listmacro>`. Note that this test uses pattern matching based on TeX's argument scanner to check if the search string is included in the list. This means that it is usually faster than looping over all items in the list, but it also implies that the items must not include curly braces which would effectively hide them from the scanner. In other words, this macro is most useful when dealing with lists of plain strings rather than printable data. When dealing with printable text, it may be preferable to use `\dolistloop` to check if an item is in the list as follows:

```
\renewcommand*{\do}[1]{%
  \ifstrequal{#1}{\item}
  {item found!\listbreak}
  {}}
\dolistloop{\mylist}
```

`\xifinlist{<item>}{<listmacro>}`

Similar to `\ifinlist` except that the `<item>` is expanded prior to the test.

`\ifinlistcs{<item>}{<listcsname>}`

Similar to `\ifinlist` except that it takes a control sequence name as its second argument.

`\xifinlistcs{<item>}{<listcsname>}`

Similar to `\xifinlist` except that it takes a control sequence name as its second argument.

### 3.8 Miscellaneous tools

`\rmntonum{<numeral>}`

The TeX primitive `\romannumeral` converts an integer to a Roman numeral but TeX or LaTeX provide no command which goes the opposite way. `\rmntonum` fills this gap. It takes a Roman numeral as its argument and converts it to the corresponding integer. Since it is expandable, it may also be used in counter assignments or `\ifnum` comparisons:

```
\rmntonum{mcmxcv}
\setcounter{counter}{\rmntonum{CXVI}}
\ifnum\rmntonum{mcmxcviii}<2000 ...
```

The `<numeral>` argument must be a literal string. It will be detokenized prior to parsing. The parsing of the numeral is case-insensitive and whitespace in the argument is ignored. If there is an invalid token in the argument, `\rmntonum` will expand to -1; an empty argument will yield an empty string. Note that `\rmntonum` will not check the numeral for formal validity. For example, both V and VX will yield 5.

`\ifrmnum{<string>}{<true>}{<false>}`

Expands to `<true>` if `<string>` is a Roman numeral, and to `<false>` otherwise. The `<string>` will be detokenized prior to performing the test. The test is case-insensitive and ignores whitespace in the `<string>`. Note that `\ifrmnum` will not check the numeral for formal validity. For example, both V and VXV will yield `<true>`. Strictly speaking, what `\ifrmnum` does is parse the `<string>` in order to find out if it consists of characters which may form a valid Roman numeral.

## 4 Revision history

### 1.6 2008-06-22

Improved <code>\robustify</code> . . . . .	2.2
Improved <code>\patchcmd</code> and <code>\ifpatchable</code> . . . . .	3.4
Modified and improved <code>\apptocmd</code> . . . . .	3.4
Modified and improved <code>\pretocmd</code> . . . . .	3.4
Added <code>\ifpatchable*</code> . . . . .	3.4
Added <code>\tracingpatches</code> . . . . .	3.4
Added <code>\AfterBeginDocument</code> . . . . .	2.4
Added <code>\ifdefmacro</code> . . . . .	3.5.I
Added <code>\ifcsmacro</code> . . . . .	3.5.I
Added <code>\ifdefprefix</code> . . . . .	3.5.I
Added <code>\ifcsprefix</code> . . . . .	3.5.I
Added <code>\ifdefparam</code> . . . . .	3.5.I
Added <code>\ifcsparam</code> . . . . .	3.5.I
Added <code>\ifdefprotected</code> . . . . .	3.5.I
Added <code>\ifcsprotected</code> . . . . .	3.5.I
Added <code>\ifdefltxprotect</code> . . . . .	3.5.I

Added \ifcsltxprotect . . . . .	3.5.I
Added \ifdefempty . . . . .	3.5.I
Added \ifcsempy . . . . .	3.5.I
Improved \ifdefvoid . . . . .	3.5.I
Improved \ifcsvoid . . . . .	3.5.I
Added \ifstrempy . . . . .	3.5.2
Added \setbool . . . . .	3.6.I
Added \settoggle . . . . .	3.6.2

## 1.5 2008-04-26

Added \defcounter . . . . .	2.3
Added \deflength . . . . .	2.3
Added \ifdefstring . . . . .	3.5.I
Added \ifcsstring . . . . .	3.5.I
Improved \rmnnum . . . . .	3.8
Added \ifrmnum . . . . .	3.8
Added extended PDF bookmarks to this manual	
Rearranged manual slightly	

## 1.4 2008-01-24

Resolved conflict with tex4ht

## 1.3 2007-10-08

Renamed package from elatex to etoolbox . . . . .	I
Renamed \newswitch to \newtoggle (name clash) . . . . .	3.6.2
Renamed \provideswitch to \providetoggle (consistency) . . . . .	3.6.2
Renamed \switchtrue to \toggletrue (consistency) . . . . .	3.6.2
Renamed \switchfalse to \togglefalse (consistency) . . . . .	3.6.2
Renamed \ifswitch to \iftoggle (consistency) . . . . .	3.6.2
Renamed \notswitch to \nottoggle (consistency) . . . . .	3.6.2
Added \AtEndPreamble . . . . .	2.4
Added \AfterEndDocument . . . . .	2.4
Added \AfterPreamble . . . . .	2.4
Added \undef . . . . .	3.I.I
Added \csundef . . . . .	3.I.I
Added \ifdefvoid . . . . .	3.5.I
Added \ifcsvoid . . . . .	3.5.I
Added \ifdefequal . . . . .	3.5.I
Added \ifcsequal . . . . .	3.5.I
Added \ifstrequal . . . . .	3.5.2
Added \listadd . . . . .	3.7.2
Added \listead . . . . .	3.7.2
Added \listgadd . . . . .	3.7.2
Added \listxadd . . . . .	3.7.2
Added \listcsadd . . . . .	3.7.2
Added \listcseadd . . . . .	3.7.2

Added \listcsgadd . . . . .	3.7.2
Added \listcsxadd . . . . .	3.7.2
Added \ifinlist . . . . .	3.7.2
Added \xifinlist . . . . .	3.7.2
Added \ifinlistcs . . . . .	3.7.2
Added \xifinlistcs . . . . .	3.7.2
Added \dolistloop . . . . .	3.7.2
Added \dolistcsloop . . . . .	3.7.2

## 1.2 2007-07-13

Renamed \patchcommand to \patchcmd (name clash) . . . . .	3.4
Renamed \apptocommand to \apptocmd (consistency) . . . . .	3.4
Renamed \pretocommand to \pretocmd (consistency) . . . . .	3.4
Added \newbool . . . . .	3.6.I
Added \providebool . . . . .	3.6.I
Added \booltrue . . . . .	3.6.I
Added \boolfalse . . . . .	3.6.I
Added \ifbool . . . . .	3.6.I
Added \notbool . . . . .	3.6.I
Added \newswitch . . . . .	3.6.2
Added \provideswitch . . . . .	3.6.2
Added \switchtrue . . . . .	3.6.2
Added \switchfalse . . . . .	3.6.2
Added \ifswitch . . . . .	3.6.2
Added \notswitch . . . . .	3.6.2
Added \DeclareListParser . . . . .	3.7.I
Added \docsvlist . . . . .	3.7.I
Added \rmntonum . . . . .	3.8

## 1.1 2007-05-28

Added \protected@csedef . . . . .	3.I.I
Added \protected@csxdef . . . . .	3.I.I
Added \gluedef . . . . .	3.I.2
Added \gluegdef . . . . .	3.I.2
Added \csgluedef . . . . .	3.I.2
Added \csgluegdef . . . . .	3.I.2
Added \mundef . . . . .	3.I.2
Added \mugdef . . . . .	3.I.2
Added \csmundef . . . . .	3.I.2
Added \csmugdef . . . . .	3.I.2
Added \protected@eappto . . . . .	3.3.I
Added \protected@xappto . . . . .	3.3.I
Added \protected@cseappto . . . . .	3.3.I
Added \protected@csxappto . . . . .	3.3.I
Added \protected@epreto . . . . .	3.3.2
Added \protected@xpreto . . . . .	3.3.2

Added \protected@ccsepreto . . . . .	3.3.2
Added \protected@csxpreto . . . . .	3.3.2
Fixed bug in \newrobustcmd . . . . .	2.I
Fixed bug in \renewrobustcmd . . . . .	2.I
Fixed bug in \providerobustcmd . . . . .	2.I

## 1.0 2007-05-07

Initial public release