# Counters

- Suspension & Resuming of counters
  - Cascaded suspension
- Super total counters
- Total
- Counter output
- Coupled counters
  - Coupled counter groups
- Loops on counters
- Label
- Associated counters
- Periodically resetting counters
- Extensions to LaTeX 2$\varepsilon$
  - Query macros
  - Document counters

**XASSOCCNT**

**Associated counters stepping simultaneously
(and other gadgets)**

**Documentation**

**Version 1.7**

**2018/12/28**

**Autor: Christian Hupfer[†]**

---

[†] typography.with.latex@gmail.com

# Contents

# Typographical conventions

Throughout this documentation following symbols and conventions are used:

- 💼 **foo** means a the class `foo`

- ✉ **foo** names a package `foo`

- ▦ **foo** indicates a counter named `foo`

- 📄 foo will indicate either a file named `foo` or a file extension `foo`

- 🗐 foo will indicate some files

- 🏷 foo names a special feature or tag `foo`

- 🔧 **foo** deals with a command or package option named `foo`

# Part I

# Introduction

## Table of Contents

## Preface

This package is the successor and a complete rewrite of ✉ **assoccnt** . Not all features of that package are implemented yet – if some functionality of your document depends on ✉ **assoccnt** , continue using the older version and shift gradually to ✉ **xassoccnt** please.

> **!** Most times class and package authors will benefit of this package, but there might be usual documents that need the features of |xassoccnt||

## 1   Introduction

The aim of this package is to provide some additional support for example for a package like ✉ **totcount** .
For example, the total number of pages in a document could be achieved by using

```
%
\regtotcounter{page}
...
The number of pages in the document is \number\totvalue{page} page(s) -- but in ↲
      ↳fact it has \total{totalpages} pages.
```
---
... The number of pages in the document is 68 page(s) – but in fact it has 67 pages.

This will work, as long there is no reset of the page counter, as it might happen in the case of `\setcounter` or `\pagenumbering` being applied in the document. The result is a false page counter total value.

This package provides associate counters, i.e. counters that are increased simultaneously with a driver counter and are not influenced by a a resetting of the driver counter, as long as not being added to the reset list by definition of the counter or explicitly by `\@addtoreset`.

This package defines some macros to handle associated counters. The only interception to the standard behaviour is within the redefined commands `\addtocounter` and `\stepcounter`. The usual commands still work, as there is code added to their definition.  In a previous version `\refstepcounter` was redefined, but since these use `\addtocounter` effectively, it was decided to use the basic command.

Internally, the associated counters are stored in one list per counter – it is not recommended to operate on those lists directly.

Please note that this package does not provide means for simultaneous stepping of counters defined by plain T<sub>E</sub>X `\newcount` command.

# 2   Requirements, loading and incompatibilities

## 2.1   Required packages and T<sub>E</sub>X engine

The package does not require features from XeLaTeX or LuaLaTeX but can be run with those features as well as with LaTeX or pdfLaTeX. The compilation documentation requires however pdfLaTeX as of version 1.7.

- ✉ **xcolor**
- ✉ **xparse**
- ✉ **l3keys2e**

The documentation file requires some more packages such as ✉ **tcolorbox**  but those packages are widely available on CTAN, MikTeX and TeXLive as well.

## 2.2   Loading of the package

Loading is done with

```
\usepackage[options]{xassoccnt}
```

For the relevant options see section 2.4

Concerning the packages ✉ **hyperref**  and ✉ **cleveref** : The preferred loading order is the usually recommended on:

- other packages

- ✉ **xassoccnt**

- ✉ **hyperref**

- ✉ **cleveref**

For potential problems see section 2.3

## 2.3  Incompatibilities

- This package cannot be used with plain TEX and will not provide support for counters (or better counter registers) that have defined with the TEX primitive `\countdef` directly in a LATEX 2$_\varepsilon$ document nor will it hook into plain TEX `\advance` commands used otherwise than in the usual LATEX 2$_\varepsilon$ wrappers `\addtocounter` etc.

- This package does not work really well with the ✉ **calc** package if that one is loaded after ✉ **xassoccnt** . Load ✉ **calc** **before** ✉ **xassoccnt** !

  Especially the ✉ **tcolorbox** bundle loads ✉ **calc** indirectly, so placing any loading of ✉ **tcolorbox** modules before ✉ **xassoccnt** is mandatory!

  > **!** As of version 1.7 ✉ **xassoccnt** will abort compilation if ✉ **calc** is loaded after this package, but will issue a warning only if ✉ **calc** is loaded first.

  > **!** Of course any package other that loads ✉ **calc** must be loaded before xassoccnt, i.e. ✉ **mathtools** .

- ✉ **xassoccnt** and ✉ **perpage** are not compatible completely. As of version 1.7 it is not possible to use the command `\AddAbsoluteCounter` from ✉ **perpage** – this feature is provided already by this package.

  > **!** It is not recommended to use counters under control of ✉ **perpage** with the association method!
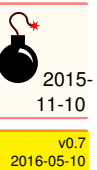
## 2.4  Package options

As of version 1.7 ✉ **xassoccnt** supports the package options

`autodefinecounters`=⟨*true,false*⟩　　　　　　　　　　　　　　　　　　　　　(initially false)

   Setting this package option to `true`, all counters used with the special package macros will be autodefined, unless disabled locally. See the commands

---

- `\DeclareAssociatedCounters`→P. 31

- `\AddAssociatedCounters`→P. 33

### nonumberofruns                                          (initially not set)

Using this package option the super total counter ▦ **numberofruns** counter will not be defined. `v0.6 2016-03-05`
See section 13.3 about this feature.

### redefinelabel=⟨*true,false*⟩                                (initially ⟨*true*⟩)

This is an experimental feature as of version 1.7. `Experimental 03-03`
Enables the redefinition of the `\label`→P. 49 command and takes care of the optional argument of
`\label`→P. 49 if ✉ **cleveref** is used. This will provide `\label`→P. 49 with a final optional argument
that can be used to allow labels for associated counters (either all or only a specified list of
counters), for more on this see `\label`→P. 49 in section Experimental features.

### standardcounterformats=⟨*choice=on/off*⟩                         (initially ⟨*on*⟩)

This key enables (⟨*on*⟩) or disables (⟨*off*⟩) the definition of the standard counter formats to be `v1.4 2017-05-10`
used in the macro `\CounterFormat`→P. 25.

**Part II**

# Tools for counters

## Table of Contents

# 3  Additions and extensions to standard counter related commands

## 3.1  Extension of LATEX 2ε commands

\addtocounter{⟨*counter*⟩}{⟨*increment value*⟩}[⟨*options*⟩]

The \addtocounter macro behaves like the usual \addtocounter counter, but takes care to specific counter features such as 🏷 periodic counters and has an optional argument in order to perform special settings.

As of 1.7, there is only one option used:

wrap=⟨*true/false*⟩                                          (initially true)    <span style="font-size:small">v0.9 2016-06-19</span>

    This key determines whether addition of values to a periodic counter (see Periodic counters) will lead to a modulo part addition.

© Dr. Christian Hupfer

## 3.2 Additions to LATEX 2ε commands

\NewDocumentCounter[⟨*options*⟩]{⟨*counter1,counter2,…*⟩}[⟨*resetting counter*⟩]

This command is a new interface to \newcounter and behaves effectively the same.

initial=⟨*integer value*⟩                  (initially 0)

     This is used for the start value of the new counter.

This command allows multiple counters (specified as a comma separated list) to be defined at once, but all have the same resetting counter then and initial value is the same for all those counters (if those options are given).

<div style="float:right">v1.2<br>2017-03-03</div>

\DeclareDocumentCounter[⟨*options*⟩]{⟨*counter*⟩}[⟨*resetting counter*⟩]

This command is the preamble-only version of \NewDocumentCounter. This command allows multiple counters (specified as a comma separated list) to be defined at once, but all have the same resetting counter then and initial value is the same for all those counters (if those options are given), i.e. the behaviour is like in \NewDocumentCounter.

<div style="float:right">v1.2<br>2017-03-03</div>

\SetDocumentCounter[⟨*options*⟩]{⟨*counter*⟩}{⟨*counter value*⟩}

This command behaves like the standard macro \setcounter, but has an additional optional 1st argument.

---

**Description of arguments of command \SetDocumentCounter**

#1 [⟨*options*⟩]:

associatedtoo=⟨*true/false*⟩            (initially false)

     If enabled (⟨*true*⟩), \SetDocumentCounter will use the counter value for <u>all</u> counters associated to this driver counter as well. Initially, this option is set to ⟨*false*⟩.

onlycounters=⟨*comma separated list of counters*⟩      (initially empty)

     If this key is used, only those associated counters are set as well that are given in the comma separated list.
     Names, that are either not referring to counters at all or to counters that are not associated to the given driver counter will be ignored silently.

#2 {⟨*counter*⟩} Holds the name of the (driver) counter to be set.

#3 {⟨*counter value*⟩} Holds the value to be set

---

Some notes on \SetDocumentCounter

- The option keys `associatedtoo` and `onlycounters` are mutually exclusive!

- The counter to be set can be either a driver counter or an otherwise associated counter.

`\StepDownCounter`[⟨*options*⟩]{⟨*counter*⟩}

This macro subtracts the value of 1 from the counter and is the counterpart of `\stepcounter`.

<div style="border:1px solid">

**Description of arguments of command `\StepDownCounter`**

#1  [⟨*options*⟩]: As of version 1.7, this option is not used

#2  {⟨*counter*⟩} Holds the name of the first counter.

</div>

v0.4 2016-01-26

`\SubtractFromCounter`[⟨*options*⟩]{⟨*counter*⟩}{⟨*delta value*⟩}

This macro subtracts the (positive) delta value from the counter and is the counterpart of `\addtocounter`

<div style="border:1px solid">

**Description of arguments of command `\SubtractFromCounter`**

#1  [⟨*options*⟩]: As of version 1.7, this option is not used

#2  {⟨*counter 1*⟩} Holds the name of the first counter.

#3  {⟨*delta value*⟩} Holds the (positive) value to be subtracted from the counter value.

</div>

v0.4 2016-01-26

`\CopyDocumentCounters`[⟨*options*⟩]{⟨*source counter*⟩}{⟨*target counter*⟩}

This document copies the counter value from the source counter in argument 2 to the target counter in argument 3.

<div style="border:1px solid">

**Description of arguments of command `\CopyDocumentCounters`**

#1  [⟨*options*⟩]: As of version 1.7, this option is not used

#2  {⟨*source counter*⟩} Holds the name of the source counter.

#3  {⟨*target counter*⟩} Holds the name of the target counter.

</div>

`\SwapDocumentCounters`[⟨*options*⟩]{⟨*counter 1*⟩}{⟨*counter 2*⟩}

This macro swaps the values of the counters given in arguments 2 and 3

> ### Description of arguments of command \SwapDocumentCounters
>
> #1  [⟨*options*⟩]: As of version 1.7, this option is not used
>
> #2  {⟨*counter 1*⟩} Holds the name of the first counter.
>
> #3  {⟨*counter 2*⟩} Holds the name of the second counter.

\SyncCounters[⟨*options*⟩]{⟨*driver counter*⟩}

This document synchronizes the driver counter value to the associated values. It has the same options as \SetDocumentCounter^(→P. 12). If the given counter is no driver counter, nothing is done.

> ### Description of arguments of command \SyncCounters
>
> #1  [⟨*options*⟩]: see \SetDocumentCounter^(→P. 12)
>
> #2  {⟨*source counter*⟩} Holds the name of the source counter.

```
%[breakable=true]
  \SetDocumentCounter{foocntr}{17}
  \SetDocumentCounter{foobarcntr}{20}

\begin{itemize}
\item Displaying counters

  \thefoocntr\ and \thefoobarcntr
\item Swapping counters

  \SwapDocumentCounters{foocntr}{foobarcntr}

  \thefoocntr\ and \thefoobarcntr

\item Step down counters

\StepDownCounter{foocntr}
\StepDownCounter{foobarcntr}

  \thefoocntr\ and \thefoobarcntr

\item Subtracting some value from the counters
  \SubtractFromCounter{foocntr}{5}
  \SubtractFromCounter{foobarcntr}{10}

  \thefoocntr\ and \thefoobarcntr
\end{itemize}
```

- Displaying counters

  17 and 20

- Swapping counters

  20 and 17

- Step down counters

  19 and 16

- Subtracting some value from the counters

  14 and 6

## 3.3   Commands checking whether a name refers to a counter

✉ **xassoccnt**  provides three commands that are quite similar – all check whether {⟨*name*⟩} is an already defined LATEX 2$_\varepsilon$ counter (name), in good tradition with the ✉ **xparse**  - syntax:

- \IfIsDocumentCounterTF[⟨⟩]{⟨*name*⟩}{⟨*true branch*⟩}{⟨*false branch*⟩}

    This macro performs the full branching

- `\IfIsDocumentCounterT[⟨⟩]{⟨name⟩}{⟨⟨long⟩ true branch⟩}`

    This command executes only if the name is a counter.

    `\IfIsDocumentCounterF[⟨⟩]{⟨name⟩}{⟨true branch⟩}`

    This command executes only if the name is not a counter.

The optional argument is not used as of version 1.7 for none of those three commands.

## 3.4   Information on counters

On occasions it might be important to have some information which counter has been changed last. Since there are four commands manipulating counter values, there are four corresponding routines for this:

<div align="right">

`Expandable`

`v1.3`
`2017-03-04`
</div>

`\LastAddedToCounter`

This command has no arguments and expands to the name of the counter which was used last in `\addtocounter`. There is no further typesetting done with the countername.

```
\newcounter{SomeCounter}

\addtocounter{SomeCounter}{10}

The last counter something added to was \LastAddedToCounter.
```
The last counter something added to was totalsubsections.

> **!** Please note that `\LastAddedToCounter` might fail!

<div align="right">

`Expandable`

`v1.3`
`2017-03-04`
</div>

`\LastSteppedCounter`

This command has no arguments and expands to the name of the counter which was stepped last using `\stepcounter`. There is no further typesetting done with the countername.

```
\stepcounter{SomeCounter}
The last counter being stepped  was \LastSteppedCounter.
```
The last counter being stepped was SomeCounter.

### \LastRefSteppedCounter

This macro gives the last counter being used in \refstepcounter and is expandable.

```
\begin{equation}
  E = mc^2 \label{eq::einstein}
\end{equation}
% \stepcounter{SomeCounter}

The last counter being refstepped  was \LastRefSteppedCounter.
```

$$E = mc^2 \tag{1}$$

The last counter being refstepped was equation.

### \LastSetCounter

This command has no arguments and expands to the name of the counter which was set last using \setcounter. There is no further typesetting done with the countername.

```
\setcounter{SomeCounter}{21}%

The last counter being set  was \LastSetCounter.
```

The last counter being set was SomeCounter.

### \LastCounterValue

This command has no arguments and expands to the value of the very last change of a counter, i.e. using \setcounter etc.

```
\setcounter{SomeCounter}{100}%

The last counter being set  was \LastSetCounter and it had the value ↵
  ↪\LastCounterValue{} then, where as \stepcounter{equation} will yield ↵
  ↪\fbox{\LastSteppedCounter} and \LastCounterValue!
```

The last counter being set was SomeCounterand it had the value 100 then, where as  will yield ⎡equation⎤ and 2!

The usage of \LastSetCounter is best together with one of the other \Last... macros.

> **!** All of the \Last... macros are expandable, i.e. it is possible to store the value to an macro defined with \edef

```
    \setcounter{SomeCounter}{50}%

    \edef\lastcounterset{\LastSetCounter}
    \edef\lastcountervalue{\LastCounterValue}

    \setcounter{equation}{81}%


    The last counter being set was \fbox{\LastSetCounter} and it had the value ↵
      ↪\LastCounterValue{} then, but we changed \lastcounterset{} earlier and it ↵
      ↪had the value \lastcountervalue{} then.
```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
The last counter being set was | equation | and it had the value 81 then, but we changed SomeCounter earlier and it had the value 50 then.

!  Please note, that all of this commands are only working in the current run of compilation, i.e. after there has been some operation on the counters. They can't be used for information on the last changed counter in a previous run.

# 4   Counter reset lists

The package ✉ **chngcntr** offers the possibility of add or remove counters to the reset list of a driver counter with the commands \counterwithin and \counterwithout, whereas the package ✉ **rem-reset** provides \@removefromreset as a counterpart to the LATEX $2_\varepsilon$ core command \@addtoreset macro.

## 4.1   Addition and Removal of counters from the reset list

v1.0 2016-07-28

\RemoveFromReset{⟨*counter name1, counter name2,…*⟩}{⟨*driver counter name*⟩}

This macro removes the counters given in the comma separated list in the first argument from the reset list of the driver counter given in the 2nd argument.
If the 2nd argument does not point to a LATEX $2_\varepsilon$ counter name an error message is shipped and the compilation fails.

v1.4
2017-05-10

'

\RemoveFromFullReset{⟨*counter name1, counter name2,…*⟩}{⟨*driver counter name*⟩}

This macro removes the counters given in the comma separated list in the first argument and all of its own reset list from the reset list of the driver counter given in the 2nd argument.
If the 2nd argument does not point to a LATEX $2_\varepsilon$ counter name an error message is shipped and the compilation fails.

`\ClearCounterResetList`{⟨*driver counter name*⟩}

> This macro removes all counters of the given driver counter reset list. The individual counter formatting macros `\theX` are reset both for the driver counter as well as the counters in the reset list to use the `\arabic` standard output macro. X means some arbitray LaTeX2e counter name. If the resetting shall not be applied, use `\ClearCounterResetList*` instead.

`\ClearCounterResetList*`{⟨*driver counter name*⟩}

> This behaves like `\ClearCounterResetList` but does **not** reset the relevant `\theX` macros.

`\AddToReset`{⟨*counter name1, counter name2,...*⟩}{⟨*driver counter name*⟩}

> This macro adds the counters given in the comma separated list in the first argument to the reset list of the driver counter given in the 2nd argument.
> If the 2nd argument does not point to a LaTeX 2ε counter name an error message is shipped and the compilation fails.
> An accidental specificiation of the driver counter to be added to its own reset list is ignored internally.

## 4.2   Information macros about the counter reset list

Sometimes it might be necessary or convenient to know how many counters are on a reset list of some other counters, i.e. added by `\newcounter{counter}[resetting counter]` or `\NewDocumentCounter`[→P. 12]. There are some macros that provide this information:

`\countersresetlistcount`{⟨*counter name*⟩}

> This macro determines the number of counters being in the reset list of the counter specified as mandatory argument.
> Please note: This command isn't expandable. The number is stored internally to another macro, which can be accessed with `\getresetlistcount`, which returns a pure integer number.

`\getresetlistcount`

> This macro returns the number of counters being in the reset list of the counter specified as mandatory argument. It needs a previous call of `\countersresetlistcount` first!
> If the counter has no other counters in its reset list, the value of 0 is returned.

`\CounterFullResetList`{⟨*counter name*⟩}

> This macro determines the full reset list of a counter as well of the counters being on the reset list, i.e. the list is tracked down until there are no counters left in a recursion.
> The counter names are stored internally in ☑ **expl3** - `\seq` - variable named `\xy_fullresetlist_seq` – the ⟨*xy*⟩ is replaced by the counter name, e.g. if the counter is named ▦ **foo**, the identifier would be `\foo_fullresetlist_seq`. Unless ❧ expl3 features are not applied, the

\CounterFullResetList is not really useful on a document or package/class developing level. However, to loop through the full reset list with some action performed on the members of the sequence, the command \LoopFullCounterResetList<sup>→P.23</sup> may be very useful.

> • The driver counter ▦ **foo** is not added to the relevant sequence.
>
> • If the name given to \CounterFullResetList does not indicate a LATEX2$_\varepsilon$ counter an error message is shipped and the compilation fails.

\IfInResetListTF[⟨⟩]{⟨*resetting counter*⟩}{⟨*reset counter*⟩}{⟨*true branch*⟩}{⟨*false branch*⟩}

This command sequence tests whether the counter ⟨*reset counter*⟩ is in the reset list of ⟨*resetting counter*⟩ and expands the relevant branch then. See the short-circuit commands \IfInResetListT and \IfInResetListF as well.

\IfInResetListT[⟨⟩]{⟨*resetting counter*⟩}{⟨*reset counter*⟩}{⟨*true branch*⟩}

This command sequence tests whether the counter ⟨*reset counter*⟩ is in the reset list of ⟨*resetting counter*⟩ and expands to the true branch. See the related commands \IfInResetListTF and \IfInResetListF as well.

\IfInResetListF[⟨⟩]{⟨*resetting counter*⟩}{⟨*reset counter*⟩}{⟨*false branch*⟩}

This command sequence tests whether the counter ⟨*reset counter*⟩ is not in the reset list of ⟨*resetting counter*⟩ and expands to the false branch. See the related commands \IfInResetListTF and \IfInResetListT as well.

\DisplayResetList[⟨*separator=,*⟩]{⟨*resetting counter*⟩}

This command displays the reset list of a counter as a separated list. If the counter has no resetting list, nothing is shown.    <sub>v0.8 2016-06-10</sub>

> **Description of arguments of command \DisplayResetList**
>
> #1 [⟨*separator*⟩] This separator is used for display, it defaults to a comma character.
>
> #2 {⟨*resetting counter*⟩}
>
>      Contains the name of counter whose resetting list should be displayed.

\ShowResetList{⟨*resetting counter*⟩}

This command displays the reset list of a counter on the terminal as the \show command would do. This is rather useful for debugging purposes only.    <sub>v0.8 2016-06-10</sub>

`\GetAllResetLists`

This determines all reset lists and stores the information internally. It should be called right before `\begindocument` or at any time inside the document environment, when new counters are added there (which is not recommended)

The information can be retrieved with `\GetParentCounter`.

`\GetParentCounter`{⟨*counter*⟩}

This macro tries to detect the counter that was responsible for the resetting of the counter named {⟨*counter*⟩} and is expandable. In order to minimize the amount of searching and maintaining expandability, the counter reset data must be stored beforehand, i.e. with `\GetAllResetLists`.

> **!** If a counter has been added to more than one parent counter as their resetting driver counter, only the most recent addition is in action. This may be correct in some occasions but there is no guarantee that the given counter name really caused the last reset of the counter given as argument.

# 5   Performing the same action for many counters

Sometimes it might be necessary to set the values of many counters at once. This can be done with consecutive `\setcounter` statements, for example. This poses no problem, but might become tedious if there are more than three counters or if this task occurs more than once. ☑ **xassoccnt** provides some macros that can do the usual operations like stepping, refstepping, adding to, resetting or setting counter values.

All macros concerning this feature use the first macro argument having a comma-separated list of counters. Whether there's a second argument depends on the specific nature of the operation that should be performed.

> **!**
> - As of version 1.7 xassoccnt does not check whether the names given in the first argument refer to counters.
>
> - All macros use the extended counter macros, i.e. are aware of associated counters and step them too if their driver counter is given in the argument list. If an associated counter itself is given in the list, this one is stepped or operated on too!

`\LoopAddtoCounters`{⟨*counter1, counter2,...*⟩}{⟨*counter increment/decrement*⟩}

The 2nd argument value is added (or subtracted) to the counters given in the list of the 1st argument using the `\addtocounter`.

#1   {⟨*counter1, counter2,...*⟩} Holds the comma separated list of counter names

#2 {⟨*counter increment/decrement*⟩} Specifies the value to be added or subtracted.

No check is performed whether #2 **is** or **expands** to an integer value.

\LoopResetCounters{⟨*counter1, counter2,...*⟩}

All counters given in the first argument are set to zero using the regular \setcounter. This is a shorthand version of \LoopSetCounters for this specific case.

#1 {⟨*counter1, counter2,...*⟩} Holds the comma separated list of counter names

\LoopRefstepCounters{⟨*counter1, counter2,...*⟩}

All counters given in the first argument are stepped using the regular \refstepcounter to allow labels – however, only the last counter will have the correct label reference.

> **!** This macro is meant only to complete the number of \Loop...Counters but is not regarded as being really useful.

#1 {⟨*counter1, counter2,...*⟩} Holds the comma separated list of counter names

\LoopSetCounters{⟨*counter1, counter2,...*⟩}{⟨*new counter value*⟩}

The 2nd argument value is used as new counter value added (or subtracted) to the counters given in the list of the 1st argument using the \addtocounter.

#1 {⟨*counter1, counter2,...*⟩} Holds the comma separated list of counter names

#2 {⟨*new counter value*⟩} Specifies the value to be set.

No check is performed whether **is** or **expands** to an integer value.

\LoopStepCounters{⟨*counter1, counter2,...*⟩}

> **Description of arguments of command \LoopStepCounters**
>
> All counters given in the first argument are stepped using the regular \stepcounter.
>
> #1 {⟨*counter1, counter2,...*⟩} Holds the comma separated list of counter names

A more general command for doing "arbitrary" operations with counters (and more setup, for example) is

\LoopCountersFunction{⟨*counter1, counter2,...*⟩}{⟨*counter operation macro*⟩}

> **Description of arguments of command \LoopAddToCounters**
>
> The 2nd argument value should hold a macro with any number of arguments, but the last mandatory argument of this macro is reserved for counter name.
>
> #1 {⟨*counter1, counter2,...*⟩} Holds the comma separated list of counter names
>
> #2 A macro name that is to be called and that operates on a counter.

```
% We assume we have the counters foocntr and foobarcntr
 \newcommand{\showcountervalues}[2]{%
   \textcolor{#1}{\csname the#2\endcsname}% Now, an extra empty line to show the ↄ
   ↄvalues in rows

 }
 % Note that the 2nd argument is not given here -- it's added by the ↄ
   ↄ\LoopCountersFunction macro
 \LoopCountersFunction{foocntr,foobarcntr}{\showcountervalues{blue}}
```

14
6

**\LoopCounterResetList**{⟨*counter name*⟩}{⟨*counter operation macro*⟩}

This macro will perform the same action on the reset list of a the counter name given as first argument, the action is a control sequence name specified by the in the second mandatory argument. The loop provides all counters on the reset list of a counter.

As of version 1.7 the counter operation macro must have two mandatory arguments, the second one is meant for the current counter in the loop.

Do not confuse this command with \LoopFullCounterResetList which tracks all counters recursively on the reset list, so \LoopCounterResetList steeps only level down in the reset list hierarchy.

**\LoopFullCounterResetList**{⟨*counter name*⟩}{⟨*counter operation macro*⟩}

This macro determines the full reset list of a counter, i.e. it cascades down the reset list and tracks the reset lists of all 'sub'-counters too and performs the counter operation macro on this.

#1 {⟨*counter name*⟩}

Holds the comma separated list of counter names

#2 {⟨*counter operation macro*⟩} A macro name that is to be called and that expects the name of a counter as the last argument.

See the macro `\CounterFullResetList`<sup>→P. 19</sup> for more information about the internal storage of the full reset list.

`\CounterWithin`{⟨*counter nameA, counter nameB,...*⟩}{⟨*drivercounter*⟩}

This macro sets all counters nameA, nameB, . . . to the reset list of the {⟨*drivercounter*⟩} and redefines the corresponding macros `\thenameA`, etc. to be prepended with `\thedrivercounter`, i.e. `\CounterWithin{equation}{section}` would mean that `\theequation` expands to `\thesection.\arabic{section}`

The default format for the counter output is arabic numbers, i.e. `\arabic` will be used.

If the macros `\thenameA` etc. should not be changed, use the starred version of this command: `\CounterWithin*`.

> **!** Please note that the redefinition of `\thenameA` etc. is only local, i.e. it is group safe.

`\CounterWithin*`{⟨*counter nameA, counter nameB,...*⟩}{⟨*drivercounter*⟩}

This macro sets all counters nameA, nameB, . . . to the reset list of the {⟨*drivercounter*⟩}, but does not change the corresponding macros `\thenameA`, etc. at all.

The default format for the counter output is arabic numbers, i.e. `\arabic` will be used.

If the macros `\thenameA` etc. should be changed, use non-starred version of this command: `\CounterWithin`.

`\CounterWithout`{⟨*counter nameA, counter nameB,...*⟩}{⟨*drivercounter*⟩}

This macro removes all counters nameA, nameB, . . . from the reset list of the {⟨*drivercounter*⟩} and redefines the corresponding macros `\thenameA`, etc. without `\thedrivercounter`, i.e. `\CounterWithout{equation}{section}` would mean that `\theequation` expand to `\arabic{section`

The default format for the counter output is arabic numbers, i.e. `\arabic` will be used.

If the macros `\thenameA` etc. should not be changed, use the starred version of this command: `\CounterWithout*`.

> **!** Please note that the redefinition of `\thenameA` etc. is only local, i.e. it is group safe.

`\CounterWithout*`{⟨*counter nameA, counter nameB,...*⟩}{⟨*drivercounter*⟩}

This macro removes all counters nameA, nameB, . . . from the reset list of the {⟨*drivercounter*⟩}, but does not redefine the corresponding macros `\thenameA`, etc.

If the macros `\thenameA` etc. should be changed, use the non-starred version of this command: `\CounterWithout`.

# 6   Counter output

Once in a while it might be necessary to provide counter output not only as integer numbers, letters or Roman figures but also using binary, octal or hexdecimal number output. The ✉ **fmtcount** package has support for this already – here are some alternatives.

## 6.1   Extra counter output types

> **!** None of the commands checks whether the argument refers to counter name.

\BinaryValue{⟨*counter name*⟩}

  This command will print the value of the counter using binary digits.

\hexValue{⟨*counter name*⟩}

  This command will print the value of the counter using lowercase hexadecimal digits.

\HexValue{⟨*counter name*⟩}

  This command will print the value of the counter using uppercase hexadecimal digits.

\OctalValue{⟨*counter name*⟩}

  This command will print the value of the counter using octal digits.

\xalphalph{⟨*counter name*⟩}

  This is allows to use more than 26 characters for the usual alphabet and prints the counter value with style aa etc. in the same manner as the ✉ **alphalph** does, but with the \int_to_alph:n macro from the ✉ **expl3** bundle. For usage with uppercase characters see \xAlphAlph.

\xAlphAlph{⟨*counter name*⟩}

  This is allows to use more than 26 characters for the usual alphabet and prints the counter value with style AA etc. in the same manner as the ✉ **alphalph** does, but with the \int_to_Alph:n macro from the ✉ **expl3** bundle. For usage with uppercase characters see \xalphalph.

## 6.2   Quick counter output changes

\CounterFormat[⟨*options*⟩]{⟨*counter1!formatname1,counter2!formatname2,...*⟩}

> **!** This macro needs the package option standardcounterformats<sup>→ P. 10</sup> to be activated with ⟨*standardcounterformats=on*⟩, which is the default.

#1  [⟨*options*⟩]

`recursive`=⟨*true/false*⟩                                              (default: false)

If this key is set, the same counter format is used for the relevant counter and its resetting counters, i.e. the macro will pursue the reset counter list chain and recursively adds `\the...` to the output format of `\thenameA` etc.

> **!**  Since the ⟨*recursive*⟩ option needs information on the parent counters, the macro `\GetAllResetLists`[→P.21] must have been called before `\CounterFormat`[→P.25] with this option can be applied. In order to provide the most recent information (which includes recently added counters or changed resetting levels), use `\GetAllResetLists`[→P.21] just before `\CounterFormat`[→P.25].

`separator`=⟨*separator character/string*⟩                             (default: !)

Specifies the separator that is used to split the counter name from the format, e.g. ⟨*chapter!R*⟩ where ⟨*chapter*⟩ is the counter name and ⟨*R*⟩ will be recognized as a counter format, meaning `\Roman` here, see table 1 for a list of predefined counter formats.

> **!**  The chosen separator must be the same for all counters in the given list of the 2nd argument (see below) and mustn't occur in the counter name itself[a].
>
> ---
> [a]It is not recommended to use counter names with non alphabetic characters anyway.

#2  {⟨*counter1!formatname1,counter2!formatname2,...*⟩} A comma separated list of counters with a given format name, each separated with a separator charactor, default is ⟨*!*⟩. If the format is omitted, the default format is `\arabic`, i.e. arabic numbers are used.

---

**Simple usages of `\CounterFormat`[→P.25]**

```
% Assume foobar is a defined counter
\setcounter{foobar}{17}

\CounterFormat{foobar!b}
\thefoobar

\CounterFormat{foobar!h}
\thefoobar


\CounterFormat{foobar!H}
\thefoobar

\CounterFormat{foobar!R}
\thefoobar
```

---

Currently following counter formats shorthands and their output macros are stored in `\AtBeginDocument` if the package option `standardcounterformats`[→P.10] is set to on, which is the default behaviour of the package.

| | |
|---|---|
| a | `\alph` |
| A | `\Alph` |
| aa | `\xalphalph`[→P.25] |
| AA | `\xAlphAlph`[→P.25] |
| b | `\BinaryValue`[→P.25] |
| h | `\hexValue`[→P.25] |
| H | `\HexValue`[→P.25] |
| n | `\arabic` |
| o | `\OctalValue`[→P.25] |
| r | `\roman` |
| R | `\Roman` |

Table 1: List of predefined counter format shorthands – please note that $\langle n \rangle$ has been used in order to allow $\langle a \rangle$ to be used for output with lowercase characters.

```
\setcounter{foobar}{30}

\CounterFormat{foobar!aa}
\thefoobar

\CounterFormat{foobar!o}
\thefoobar

\CounterFormat{foobar!AA}
\thefoobar
```

```
10001
11
11
XVII
ad
36
AD
```

Showing the ⟨*recursive*⟩ option of `\CounterFormat`[→P.25]

```
% All counters are using \arabic by default from \newcounter or \NewDocumentCounter
\NewDocumentCounter{foolevelzero}
\NewDocumentCounter{foolevelone}[foolevelzero]
```

```
\NewDocumentCounter{fooleveltwo}[foolevelone]

% Get the current reset lists! (Important}

\GetAllResetLists

% Now change to Hex format (!H) for all counters in the hierarchy.
\CounterFormat[recursive]{fooleveltwo!H}

\setcounter{foolevelzero}{20}% Should be 14
\setcounter{foolevelone}{15}% Should be F
\setcounter{fooleveltwo}{10}% Should be A

\thefoolevelzero % -> 14

\thefoolevelone % 14.F

\thefooleveltwo % 14.F.A
```
---
14
14.F
14.F.A

\StoreCounterFormats[⟨*options*⟩]{⟨*formatshorthandA!formatmacroA,formatshorthandB!formatmacroB,…*⟩}

Stores the counter formats separated by the separator charactor as given in the option to the global list. Existing formats will be overwritten if the format shorthand already exists. There is no warning about this! The only handled option is separator$^{\to P.\,26}$ and has the same meaning as in \CounterFormat$^{\to P.\,25}$.

The formatmacro must be a command sequence with exactly one mandatory argument, which may not be specified in the format storage process.

> **!**  This macro will become a preamble-only command most likely.

```
\StoreCounterFormats{foo!\Roman,foobarnice!\OctalValue}
\setcounter{foobar}{17}
\CounterFormat{foobar!foo}
\thefoobar

\CounterFormat{foobar!foobarnice}
\thefoobar
```
---
XVII
21

\AddCounterFormats[⟨*options*⟩]{⟨*formatshorthandA!formatmacroA,formatshorthandB!formatmacroB,...*⟩}

Adds the counter formats separated by the separator charactor as given in the option to the global list, similar to \StoreCounterFormats<sup>→P.28</sup>. Existing formats will be overwritten if the format shorthand already exists. There is no warning about this! The only handled option is separator<sup>→P.26</sup> and has the same meaning as in \CounterFormat<sup>→P.25</sup>.

The formatmacro must be a command sequence with exactly one mandatory argument, which may not be specified in the format storage process.

<div style="text-align: right">v1.4 2017-05-10</div>

\RemoveCounterFormats[⟨*options*⟩]{⟨*formatshorthandA,formatshorthandB,...*⟩}

Removes the given counter formats from the global list.
The optional argument is ignored as of version 1.7.

<div style="text-align: right">v1.4 2017-05-10</div>

**Part III**

# Features

## Table of Contents

© Dr. Christian Hupfer

# 7 Associated counters

The main purpose of this package is co-stepping of counters, but there are some helper commands in addition to macros provided LaTeX 2$_\varepsilon$ already, see section Additions to standard commands.

- Section Association macros describes the most important macros for setting up associated counters

- Section Driver macros informs about the macros for setting up, removing or clearing driver counters

- Section Query macros deals with query command sequences about counters being a driver or an associated counters

- Section Information macros contains routines that show which counters have been changed last

## 7.1 Associated counters commands

All macros have the general rule, that the driver counter is specified as 1st mandatory argument to the macro, which is in almost all cases the 2nd argument of the macro.

\DeclareAssociatedCounters[⟨*options*⟩]{⟨*driver counter*⟩}{⟨*associated counters list*⟩}

This command is the main macro of the package. It declares the counter names being specified in comma - separated - list (CSV) which should be stepped simultaneously when the driver counter is increased by \stepcounter. If only counter is to be associated, omit a trailing ","!

#1 [⟨*options*⟩]:

autodefine=⟨*choice*⟩                                                      (initially none)

This choice - key can be specified if the specified counters should be defined if they not already available. Possible values are
- none – no counter is autodefined
- all – all counters will be autodefined
- driver – only driver counters will be autodefined
- associated – only associated counters will be autodefined

Default is none

sloppy

If autodefine key is used, the sloppy key disables the check whether a counter is defined already.

#2 {⟨*driver counter*⟩}

     Holds the name of the driver counter to which the list of counters should be associated

#3 {⟨*associated counters list*⟩}

     A comma separated list of counter names that should be associated to the driver counter

- This command is a preamble command, i.e. it can be used in the preamble of the document or within other packages or class files only.

- This command should be used as early as possible, i.e. in the preamble of the document, since the driven counters are not increased as long as they are not associated to the driver counter. On the hand, it is possible or may be required to control the starting point of the association at any position in the body of the document, when the association should start later on. Use the command `\AddAssociatedCounters`<sup>→P.33</sup> if counters should be associated within the document body.

```
%%%% The association of anothertotalpages in this example just takes place ↄ
    ↄhere, so the stepping of the counter will start from here and providing ↄ
    ↄa 'wrong' value.
%%%%
\DeclareAssociatedCounters{page}{totalpages,anothertotalpages}%
This document has \number\totvalue{totalpages} (note: ↄ
    ↄ\number\totvalue{anothertotalpages}) pages.
```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

This document has 67 (note: 32) pages.

- Current version (1.7) rules:

  - No checking whether the 2nd and 3rd arguments hold counter names is applied.

  - Mutually cross - association of two counters is not supported! The compilation will stop on this!
  A driver counter, say, ▦ **foo**) of, say ▦ **foobar** can not be an associated counter of ▦ **foobar**, which in turn can be a driver counter of other counters, of course.
  A contrary feature are the 🏷 coupled counters – If some counters should share a common base, i.e. increasing one arbitrary member counter of a group of counters then all should be increased, this called coupling of counters – all group members are on an equal footing. See section 9 about this feature.
  On the other side, 🏷 associated counters belong to a hierarchy. The driver counter dominates the associated counters.

- A self-association of the driver counter to itself is ignored internally as this would lead to inconsistent counter values.

- The order of the specification of associated counters in the 2nd arguments is of no importance.

- Specifying an associated counter name multiple times has no effect, only the first occurence of the name will be used.

**\AddAssociatedCounters**[⟨*options*⟩]{⟨*driver counter*⟩}{⟨*associated counters list*⟩}

The usage of this macro is similar to \DeclareAssociatedCounters<sup>→P.31</sup>; if it is called in the document preamble (or in package file), \AddAssociatedCounters falls back to

$$\text{\DeclareAssociatedCounters}^{\rightarrow P.31},$$

having the same optional argument functionality with autodefine<sup>→P.31</sup> and sloppy<sup>→P.31</sup>; if it is called in the document body, this command adds some counters to the associated counter list for a specific driver counter – if this list does not exists, the LATEX run will issue a warning, but add the driver counter to the driver list and the associated counters analogously.

Using \AddAssociatedCounters in the document body automated generation of counters is disabled.

> **Description of arguments of command \AddAssociatedCounters**
>
> #1 [⟨*options*⟩]: As of version 1.7, the optional argument [⟨*options*⟩] are the same as for \DeclareAssociatedCounters<sup>→P.31</sup>, see autodefine<sup>→P.31</sup> and sloppy<sup>→P.31</sup>.
>
> #2 {⟨*driver counter*⟩}
>
> Holds the name of the driver counter to which the list of counters should be associated
>
> #3 {⟨*associated counters list*⟩}
>
> A comma separated list of counter names that should be associated to the driver counter

**\RemoveAssociatedCounter**{⟨*driver counter*⟩}{⟨*associated counter*⟩}

This command removes a counter from the existing list for a driver counter, i.e. the counter will not be increased any longer by \stepcounter. It can be increased however manually, of course.

```
\RemoveAssociatedCounter{page}{anothertotalpages}
This document has \number\totvalue{totalpages} (beware:
    \number\totvalue{anothertotalpages}) pages.
```
---
This document has 67 (beware: 32) pages.

**\RemoveAssociatedCounters**{⟨*driver counter*⟩}{⟨*list of associated counters*⟩}

© Dr. Christian Hupfer

This command removes the comma-separated-value list of counters from the existing list for a driver counter, i.e. the counters will not be increased any longer by \stepcounter. They can be increased however manually, of course.

Take care not to confuse the commands \RemoveAssociatedCounters and \RemoveAssociatedCounter

### \ClearAssociatedCounters[⟨*options*⟩]{⟨*driver counter*⟩}

This command clears the internal list for all counters associated to the {⟨*driver counter*⟩}. The counters will not be increased automatically any longer.

The optional argument is not used as of version 1.7.

Please note that the driver counter is not removed from the list of driver counters – this simplifies reassociating of (other) counters to this one later on with the macro \AddAssociatedCounters<sup>→P. 33</sup> and suppress the relevant warning.

If the driver counter and all its associated counters should be removed, use \RemoveDriverCounter<sup>→P. 35</sup> instead.

### \DeclareTotalAssociatedCounters[⟨*options*⟩]{⟨*driver counter*⟩}{⟨*associated counters list*⟩}   v1.2 2017-03-03

This command combines the features of ❧ associated counters and ❧ total counters, i.e. the associated counters are defined with \NewTotalDocumentCounter<sup>→P. 48</sup> and associated to the driver counter.

See section 12 for more information on ❧ total counters.

## 7.2  **Driver counter commands**

\AddDriverCounter[⟨*options*⟩]{⟨*driver counter name*⟩}

> ### Description of arguments of command \AddDriverCounter
>
> #1 [⟨*options*⟩]: As of 1.7, the optional argument [⟨*options*⟩] is not used so far, but is reserved for later purposes.
>
> #2 {⟨*driver counter name*⟩}
>
> Holds the name of the driver counter that should be added to the list of driver counters.

\RemoveDriverCounter[⟨*options*⟩]{⟨*driver counter*⟩}

This command clears the internal list for all counters associated to the {⟨*driver counter*⟩}. The counters will not be increased automatically any longer.
The optional argument is not used as of version 1.7.
If all driver counters should be unregistered, use \ClearDriverCounters instead!

\ClearDriverCounters[⟨*options*⟩]

This clears completely the list of driver counters, such that no counters are regarded as being associated – i.e. no driver is hold as being a driver counter.
The optional argument is not used as of version 1.7.

## 7.3  **Commands for queries**

Sometimes it might be necessary to get information, whether a counter is regarded as a driver or as an associated counter. This section describes some query macros in order to obtain this information.

\IsAssociatedToCounter{⟨*driver counter*⟩}{⟨*associated counter*⟩}{⟨*True branch*⟩}{⟨*False branch*⟩}

This macro checks, whether a counter is associated to a particular given driver counter and expands the corresponding branch. If the internal driver counter list does not exist, the false branch will be used, since this also means, that the possibly associated counter is not associated at all.

---

**Description of arguments of command \IsAssociatedToCounter**

#1 {⟨*driver counter*⟩}

Holds the name of the driver counter to which {⟨*associated counter*⟩} the could possibly be associated.

#2 {⟨*associated counter*⟩}

Contains the name of the possibly associated counter.

#3 {⟨*True branch*⟩}

This code is expanded if the counter is associated to the driver, otherwise it is ignored.

#4 {⟨*True branch*⟩}

This code is expanded if the counter is **not** associated to the driver, otherwise it is ignored.

---

```
% Remove associated counter first for demonstration purposes
\RemoveAssociatedCounter{page}{anothertotalpages}
\IsAssociatedToCounter{page}{totalpages}{Yes, totalpages is associated}{No, ↲
    ↳totalpages is not associated}

\IsAssociatedToCounter{page}{anothertotalpages}{Yes, anothertotalpages is ↲
    ↳associated}{No, anotherpages is not associated}
```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Yes, totalpages is associated
No, anotherpages is not associated

See also

- \IsAssociatedCounter<sup>→P.37</sup> for checking whether a counter is associated
- \IsDriverCounter<sup>→P.37</sup> in order to check whether a counter is a driver.
- \GetDriverCounter returns the driver counter name for a given associated counter name

\GetDriverCounter{⟨*counter name*⟩}

This commands returns the driver counter to which the counter name of the first argument is connected to. If the counter is not defined, the macro returns nothing.

- No check whether the counter name is defined is performed
- No check whether the counter is associated at all is performed. Usage of this command in conjunction with \IsAssociatedCounter<sup>→P.37</sup> is strongly encouraged.

---

```
%
totalpages is associated to the ↩
    ↪\textcolor{blue}{\textbf{\GetDriverCounter{totalpages}}} counter.
% Try with an undefined counter name
humptydumpty is associated to the ↩
    ↪\textcolor{blue}{\textbf{\GetDriverCounter{humptydumpty}}} counter.
```

totalpages is associated to the **page** counter. humptydumpty is associated to the **–** counter.

\IsAssociatedCounter{⟨*counter name*⟩}{⟨*True branch*⟩}{⟨*False branch*⟩}

This commands tests, whether a given counter name is an associated counter and expands correspondingly the true or the false branch. The command does not tell to which driver the counter it is associated – this information can be obtained by \GetDriverCounter[→ P. 36].

### Description of arguments of command \IfAssociatedCounter

#1 {⟨*counter name*⟩}

Contains the name of the possibly associated counter

#2 {⟨*True branch*⟩}

This code is expanded if the counter is associated to a driver, otherwise it is ignored

#3 {⟨*True branch*⟩}

This code is expanded if the counter is **not** associated a driver, otherwise it is ignored

```
\IsAssociatedCounter{section}{Yes, section is an associated counter}{No, ↩
    ↪section counter does not have the associated counter properties}
\IsAssociatedCounter{totalpages}{Yes, totalpages is an associated counter}{No, ↩
    ↪totalpages counter does not have the associated counter properties}
```

No, section counter does not have the associated counter properties Yes, totalpages is an associated counter

\IsDriverCounter{⟨*driver counter name*⟩}{⟨*True branch*⟩}{⟨*False branch*⟩}

This commands tests, whether a given counter name is a driver counter and expands correspondingly the true or the false branch.

---

**Description of arguments of command** `\IfDriverCounter`

#1   {⟨*driver counter name*⟩}

Contains the name of the possible driver counter

#2   {⟨*True branch*⟩}

This code is expanded if the counter is a driver, otherwise it is ignored

#3   {⟨*True branch*⟩}

This code is expanded if the counter is **not** a driver, otherwise it is ignored

---

```
\IsDriverCounter{section}{Yes, section is a driver counter}{No, section counter ⤸
    ⤷does not have driver properties}
```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Yes, section is a driver counter

# 8   Backup and restore of counter values

**!** This feature is currently disabled!    <span>v1.0 2016-07-28</span>

# 9   Coupled counters

**!** The features described here are very experimental and not fully implemented so far.    <span>v0.5 2016-02-27</span>

Occasionally there are requests where the figure or table environment should use the same counter in the sense of using continued counter values, e.g figure 1 is followed by table 2, the next figure is numbered as 3 etc.

This can be achieved with the concept of coupled counters. As usual, those counters belonging to a 'group' should be declared first in the preamble. In some sense coupled counters are similar to associated counters.

## 9.1   Common options for most of the coupled counter macros

`name`=⟨*name of a group*⟩

This option has the name of the counter group that should be coupled, say "figuretablegroup" etc.    <span>v0.6 2016-03-05</span>

`multiple`=⟨*true,false*⟩                                                                      (initially false)

This option allows to add a counter multiple times to a counter group. In general, using this style is not recommended.    <span>v0.6 2016-03-05</span>

---

## 9.2  Macros for declaring, adding and removing coupled counters

\DeclareCoupledCounters[⟨*options*⟩]{⟨*counter name1, counter name2, ...*⟩}

v0.5 2016-
02-27

> #1  [⟨*options*⟩]: See section 9.1 for a explanation about available options.
>
> #2  {⟨*counter name 1, counter name2, ...*⟩}:  The list of counters that should should be stepped together for the given counter group.

This macro is a preamble-only command.

\DeclareCoupledCountersGroup{⟨*counter group name*⟩}

v0.5 2016-
02-27

> This macro defines a name for a counter group and allocates a new group list for the counter names. If the name already exists, nothing is done.
>
> #1  {⟨*counter group name*⟩}: The name of the counter group.

This macro is a preamble-only command and does not add counters to the group container. Use \DeclareCoupledCounters or \AddCoupledCounters to add counters to the relevant group.

\RemoveCoupledCounters[⟨*options*⟩]{⟨*counter name1, counter name2, ...*⟩}

v0.5 2016-
02-27

> This removes the comma separated counter names from the coupled counter list given in the name$^{\to P.38}$ option.
>
> #1  [⟨*options*⟩]: As of version 1.7 the only recognized option is name$^{\to P.38}$.
>
> #2  {⟨*counter name 1, counter name2, ...*⟩}: The list of counters that should should removed from the given counter group.

> **!**
> - The list name itself is still available
>
> - If the list given by the name$^{\to P.38}$ option does not exist, \RemoveCoupledCounters issues a warning on the console and ignores this list then.

If all counters from a group name should be removed, this is equal to clearing – just use \ClearCoupledCounters$^{\to P.40}$ for simpler usage of this feature.

\AddCoupledCounters[⟨*options*⟩]{⟨*counter name1, counter name2, ...*⟩}

v0.6
2016-03-05

> This adds the listed counter names to coupled counter list. It acts like \DeclareCoupledCounters, but does not setup new counter groups. Please use \DeclareCoupledCounters first, then apply \AddCoupledCounters later on.
>
> #1  [⟨*options*⟩]: See section 9.1 for a explanation about available options.
>
> #2  {⟨*counter name 1, counter name2, ...*⟩}:  The list of counters that should should be stepped together for the given counter group.

> **!** If the list given by the `name`[→P.38] option does not exist, `\AddCoupledCounters` issues a warning on the console and ignores this list then. The counters are not added to any list at all.

### `\ClearCoupledCounters`{⟨*options*⟩}

This removes all names from the given name of a group of coupled counters.

**#1** [⟨*options*⟩] : As of version 1.7 the only recognized option is `name`[→P.38].

v0.6 2016-03-05

After clearing a list, the coupling stops for the counters on that list (unless they are part of another list, which is possible, but not recommended). using `\AddCoupledCounters`[→P.39] with the relevant `name`[→P.38] option adds counters again to the list and the coupling is active again, however, for different counters (eventually).

In order to clear all coupled counter lists, use `\ClearAllCoupledCounters` instead.

> **!**
> - The list name itself is still available
> - If the list given by the `name`[→P.38] option does not exist, `\ClearCoupledCounters` issues a warning on the console and ignores this list then.

### `\ClearAllCoupledCounters`

This removes all coupled counter groups, but not the group names, i.e. the list names can be used later on to add counter names again. In order to clear a specific list, use `\ClearCoupledCounters`.

v0.6 2016-03-05

### `\IsCoupledCounterTF`{⟨*counter name*⟩}{⟨*true branch*⟩}{⟨*false branch*⟩}

This macro tests if a counter is under the administration of the coupled counter commands and expands to the relevant branch then. There are two short-circuit commands `\IsCoupledCounterT` and `\IsCoupledCounterF`.

v0.6 2016-03-05

### `\IsCoupledCounterT`{⟨*counter name*⟩}{⟨*true branch*⟩}

This macro tests if a counter is under the administration of the coupled counter commands and executes the true branch then. There are two related commands `\IsCoupledCounterTF` and `\IsCoupledCounterF`.

v0.6 2016-03-05

### `\IsCoupledCounterF`{⟨*counter name*⟩}{⟨*false branch*⟩}

This macro tests if a counter is under the administration of the coupled counter commands and executes the false branch then if this is not the case. There are two related commands `\IsCoupledCounterTF` and `\IsCoupledCounterT`.

v0.6 2016-03-05

# 10   Periodic counters

It might be very convenient to have counters that are automatically reset not only by a driving counter such as ▦ **chapter** but also periodically, i.e. after a certain amount of steps – this can be achieved with the concept of periodic counters.

## 10.1   Commands related to periodic counters setup

\DeclarePeriodicCounter[⟨⟩]{⟨*counter name*⟩}{⟨*counter treshold value*⟩}

> This defines the counter given in the first mandatory argument as a periodic counter and is automatically reset if the treshold value is reached.

> ! The command \DeclarePeriodicCounter does not define a new counter, however but is the preamble-only version of \AddPeriodicCounter.

> Please note that in case of \addtocounter applied to a periodic counter the value to be added leads to a modulo division such that the counter might be reset if the addition would increase the counter beyond the treshold value, the module part will be added then. In order to prevent this wrapping, use the wrap<sup>→ P. 11</sup> option to \addtocounter<sup>→ P. 11</sup>:

```
\setcounter{foocntr}{3}%
\AddPeriodicCounter{foocntr}{8}%

Value of foocntr is: \thefoocntr   % Should be 3

\addtocounter{foocntr}{20} % Is it 23? No, it is 23 % 8 = 7
Value of foocntr is \thefoocntr\ now!

Adding a value of 4 again:
\addtocounter{foocntr}{4} % Is it 11? No, it is 11 % 8 = 3
Value of foocntr is \thefoocntr\ now!

Now prevent the wrapping
\addtocounter{foocntr}{10}[wrap=false] % Is it 13? Yes, it is!
Value of foocntr is \thefoocntr\ now!
```
----
Value of foocntr is: 3
Value of foocntr is 7 now!
Adding a value of 4 again:  Value of foocntr is 3 now!
Now prevent the wrapping  Value of foocntr is 13 now!

\AddPeriodicCounter[⟨⟩]{⟨*counter name*⟩}{⟨*counter treshold value*⟩}

> This defines the counter given in the first mandatory argument as a periodic counter and is automatically reset if the treshold value is reached.

\RemovePeriodicCounter[⟨*options*⟩]{⟨*counter name*⟩}

This removes the counter given in the first mandatory argument as a periodic counter. The counter is reset unless the `reset` is set to ⟨*false*⟩.

#1  [⟨*options*⟩]

As of version 1.7, there is only one option:

reset=⟨*true/false*⟩                                                      (initially true)

Use 'false' to prevent the resetting of the relevant counter after removal!

#2  {⟨*counter name*⟩} – the name of the counter that should be no periodic counter any longer.

If all periodic counters should be removed, use the macro \RemoveAllPeriodicCounters instead.

\RemoveAllPeriodicCounters[⟨*options*⟩]

This command removes all counters given in the first mandatory argument as a periodic counter. All counters are reset unless the `reset` option is set to ⟨*false*⟩.

#1  [⟨*options*⟩] As of version 1.7, there is only one option: `reset`, having the same meaning as in \RemovePeriodicCounter.

#2  {⟨*counter name*⟩} – the name of the counter that should be no periodic counter any longer.

If only a specific counter shall be removed from the periodic counter property use the command \RemovePeriodicCounter instead.

\ChangePeriodicCounterCondition[⟨*options*⟩]{⟨*counter   name*⟩}{⟨*new   counter   treshold value*⟩}

This changes the counter treshold condition – the counter is reset automatically if not specified otherwise with the `reset` option.

#1  [⟨*options*⟩]

As of version 1.7, there is only one option: `reset`, which serves the same functionality as in \RemovePeriodicCounter.

#2  {⟨*counter name*⟩} – the name of the counter that should be no periodic counter any longer.

#3  {⟨*new counter value treshold*⟩} – the new value after which an automatic resetting will occur.

## 10.2   Commands to query for periodic counter feature

\IsPeriodicCounterTF{⟨*counter name*⟩}{⟨*true branch*⟩}{⟨*false branch*⟩}

This macro tests if a counter is under the administration of the periodic counter commands and expands to the relevant branch then. There are two short-circuit commands: \IsPeriodicCounterT<sup>→ P. 43</sup> and \IsPeriodicCounterF<sup>→ P. 43</sup>.

\IsPeriodicCounterT{⟨*counter name*⟩}{⟨*true branch*⟩}

This macro tests if a counter is under the administration of the periodic counter commands and expands to the ⟨*true*⟩ branch then. There are two related commands: \IsPeriodicCounterTF and \IsPeriodicCounterF<sup>→P.43</sup>.

\IsPeriodicCounterF{⟨*counter name*⟩}{⟨*false branch*⟩}

This macro tests if a counter is under the administration of the periodic counter commands and expands to the ⟨*false*⟩ branch then if this is not the case. There are two related commands: \IsPeriodicCounterTF<sup>→P.42</sup> and \IsPeriodicCounterT.

# 11   Suspending and resuming (associated) counters

Rather than removing an associated counter from the list, it is possible to suspend the automatic stepping for a while and then resume it (or completely drop it), for example, if the value of a counter should not be stepped within a specific chapter etc.

> **!** Suspension and resuming counters can cause wrong hyper links if ☑ **hyperref** is used.

## 11.1   Macros for suspension and resume

\SuspendCounters[⟨*options*⟩]{⟨*counters list*⟩}

---

**Description of arguments of command \SuspendCounters**

**#1** [⟨*options*⟩]

Not used so far, reserved for later usage.

**#2** {⟨*counters list*⟩}

Contains the name of counters to be suspended, separated by commas (CSV - list)

---

\CascadeSuspendCounters[⟨*options*⟩]{⟨*counters list*⟩}

This macro is more powerful than \SuspendCounters, since it tries to detect whether a counter has a reset list and 'mutes' the counters on this list as well and checks whether those counters themselves have reset lists and cascades down to the final state.

> **!** Stated differently: All counters anyhow connected to a counter named ▦ **foo** will be suspended, e.g. for the 💼 **book** class and ▦ **chapter**, this means in a standard setup, that ▦ **section,figure,table,equation,footnote** will be suspended, as well as in consequence ▦ **subsection,subsubsection,paragraph,subparagraph**, assuming hereby no other counters have been added to the reset lists.

> **Description of arguments of command** `\CascadeSuspendCounters`
>
> #1  [⟨*options*⟩]
>
>    Not used so far, reserved for later usage.
>
> #2  {⟨*counters list*⟩}
>
>    Contains the name of counters to be suspended, separated by commas (CSV - list)

`\ResumeSuspendedCounters`[⟨*options*⟩]{⟨*counters list*⟩}

   As of version 1.7 the optional argument is not used and reserved for later purposes. This command revokes the suspension of the counters in the {⟨*counters*⟩} list.

`\ResumeAllSuspendedCounters`[⟨*options*⟩]

   As of version 1.7 the optional argument is not used and reserved for later purposes. This command revokes all suspended counters.

v0.8 2016-06-10

## 11.2   Query suspension

`\IsSuspendedCounter`{⟨*counter name*⟩}{⟨*true branch*⟩}{⟨*false branch*⟩}

   See Suspending and Resuming on this topic.
   This command checks, whether a counter is suspended, i.e. not updated at all and expands the corresponding branches.

   #1  {⟨*counter name*⟩}

      Contains the name of counter presumed to be suspended

   #2  {⟨*True branch*⟩}

      This code is expanded if the counter is suspended, otherwise it is ignored

   #3  {⟨*True branch*⟩}

      This code is expanded if the counter is **not** suspended, otherwise it is ignored

> **!**  If a driver counter is suspended, all counters associated to it are suspended too!

```
\textbf{This example shows 4 equations, but only two of them are counted}

\begin{equation}
E_{0} = mc^2
\end{equation}

Now suspend the equations:

\SuspendCounters{equation}
\begin{equation}
E^2 = \left({ pc}\right)^2 + E^{2}_{0}
\end{equation}

\begin{equation}
  m(v) = \frac{m_{0}}{\sqrt{1-\frac{v^2}{c^2}}}
\end{equation}

And resume it: \ResumeSuspendedCounters{equation}

\begin{equation}
  E = h \nu
\end{equation}

There are \number\totvalue{totalequations}~equations in here!
```

**This example shows 4 equations, but only two of them are counted**

$$E_0 = mc^2 \tag{1}$$

Now suspend the equations:

$$E^2 = (pc)^2 + E_0^2 \tag{1}$$

$$m(v) = \frac{m_0}{\sqrt{1 - \frac{v^2}{c^2}}} \tag{1}$$

And resume it:

$$E = h\nu \tag{2}$$

There are 2 equations in here!

## 12   Total counters

Similarly to the package ✉ **totcount**  or the features of ✉ **totalcount**  by Axel Sommerfeldt this package provides the possibility of defining a counter that stores its finally value to the auxiliary file and starts from this value then, if not set otherwise to another value.

2015-11-27

The declaration of a total counter is a preamble - only event and \DeclareTotalDocumentCounter<sup>→P.48</sup> is a preamble-only command in order to prevent counter register confusion. If a certain existing counter should be treated with total counter features, use \RegisterTotalDocumentCounter<sup>→P.46</sup> instead.

Use \NewTotalDocumentCounter<sup>→P.48</sup> only in rare cases, if a total counter must be defined within the document body.

> **!** The standard LaTeX $2_\varepsilon$ commands \stepcounter, \addtocounter and \setcounter support the specification of a total counter, but \refstepcounter will fail since the usage of a total counter for labelling purposes is most probably of no use (as of version 1.7)

## 12.1 Defining total counters

\RegisterTotalDocumentCounter[⟨*options*⟩]{⟨*total counter name*⟩}

v0.5 2016-02-27

#1 [⟨*options*⟩]: As of version 1.7, only this option is used

supertotal=⟨*true,false*⟩ (false)

Set this key to switch the super total counter on or off.

#2 {⟨*total counter name*⟩}: The name of the total counter. This must be the same like the name of an already existing counter. Internally another counter is defined which has a prefix to prevent name clashes with counter names defined by the package ✉ **totalcount** . If the counter name does not exist, the compilation exits with an error message.

\TotalCounterInternalName{⟨*counter name*⟩}

This command reports the internal name of a total counter or the usual name if this counter is not a total one.

```
\TotalCounterInternalName{chapter}

\TotalCounterInternalName{foototal}
```
---
```
chapter
xassoccnt@total@foototal
```

Expandable

v1.0 2016-07-28

\TotalCounterInternalNameExp{⟨*counter name*⟩}

This command is the expandable version of \TotalCounterInternalName

**\TotalValue**{⟨*counter name*⟩}

This command prints the value of a total counter or falls back to the value of the counter if this is not a total counter.

```
''Total'' value of the section non-total counter: \TotalValue{section}

Total value of the foototal total counter: \TotalValue{foototal}
```

"Total" value of the section non-total counter: 3
Total value of the foototal total counter: 1

## 12.2   Queries about total counters

**\IsTotalCounterTF**{⟨*counter name*⟩}{⟨*true branch*⟩}{⟨*false branch*⟩}

This macro tests if a counter is under the administration of the total counter commands and expands to the relevant branch then. There are two short-circuit commands \IsTotalCounterT and \IsTotalCounterF.

**\IsTotalCounterT**{⟨*counter name*⟩}{⟨*true branch*⟩}

This macro checks if a counter is under the administration of the total counter commands and expands to the code in the second argument if this is true.

**\IsTotalCounterF**{⟨*counter name*⟩}{⟨*false branch*⟩}

This macro checks if a counter is under the administration of the total counter commands and expands to the code in the second argument if this is not the case.

```
\IsTotalCounterTF{foototal}{Yes, this is a total counter}{No, this is no total ↰
   ↳counter}

\IsTotalCounterTF{section}{Yes, this is a total counter}{No, this is no total ↰
   ↳counter}

\IsTotalCounterT{foototal}{Yes, this is a total counter}

\IsTotalCounterF{page}{No, page isn't a total counter}
```

Yes, this is a total counter
Yes, this is a total counter
Yes, this is a total counter
No, page isn't a total counter

> **!** The features of using other 🗐 aux files or a different external file as provided by ✉ **totcount** is not (yet) support as of version 1.7.

# 13   Super total counters

In addition to the concept of a total counter, there is also the possibility of using super total counters – those counters survive the reset at the beginning of a compilation, i.e. the value of a super total counter might be stepped in each run and as such the number of compilation runs etc. can be tracked. The values of the last run are persistent as long as the ▮ .aux file isn't deleted.

<div align="right">2015-11-25</div>

## 13.1   Defining super total counters

\NewTotalDocumentCounter[⟨*options*⟩]{⟨*total counter name1, total counter name2,...*⟩}

This macro defines a new counter (which mustn't exist before of course) and puts it under control of the total counter features.

#1 [⟨*options*⟩]: As of version 1.7, only this option is used

supertotal→P.46 – this has the same meaning as in \RegisterTotalDocumentCounter→P.46 and defaults to false.

#2 {⟨*total counter name1, total counter name2,...*⟩}: The names of the total counter, separated by a comma. This must not be the same like the name of any already existing counter. Internally another counter is defined which has a prefix to prevent name clashes with counter names defined by the package ✉ **totcount** .

\DeclareTotalDocumentCounter[⟨*options*⟩]{⟨*total counter name1, total counter name2,...*⟩}

This is the preamble - only version of \NewTotalDocumentCounter and should be preferred in most cases over that command.

If an already existing counter should be tracked with total counter features, use \RegisterTotalDocumentCo instead.

This command allows multiple counters (specified as a comma separated list) to be defined at once.

<div align="right">v1.2<br>2017-03-03</div>

## 13.2   Queryies about super total counters

\IsSuperTotalCounterTF{⟨*counter name*⟩}{⟨*true branch*⟩}{⟨*false branch*⟩}

This macro tests if a counter is under the administration of the super total counter commands and expands to the relevant branch then. There are two short-circuit commands \IsSuperTotalCounterT and \IsSuperTotalCounterF→P.49.

\IsSuperTotalCounterT{⟨*counter name*⟩}{⟨*true branch*⟩}

This macro checks if a counter is under the administration of the super total counter commands and expands to the code in the second argument if this is true.

\IsSuperTotalCounterF{⟨*counter name*⟩}{⟨*false branch*⟩}

>   This macro checks if a counter is under the administration of the super total counter commands and expands to the code in the second argument if this is not the case.

```
\IsSuperTotalCounterTF{numberofruns}{Yes, this is a super total counter}{No, this ↵
    ⮡is no super total counter}

\IsSuperTotalCounterT{numberofruns}{Yes, this is a super total counter}

\IsSuperTotalCounterTF{chapter}{Yes, this is a total counter}{No, this is no ↵
    ⮡super total counter}
```
---
No, this is no super total counter
No, this is no super total counter

## 13.3   The ▦ **numberofruns** counter

This package adds a counter of its own: ▦ **numberofruns** which is a super total counter and is stepped each compilation run. It's added in \AtBeginDocument and can be retrieved with \TotalValue[→ P. 47]. Use the nonumberofruns[→ P. 10] package option to prevent the definition of this counter.

# 14   Experimental features

> **!** The content here is only of experimental nature and there is no guarantee that the feature will be maintained in future releases.   💣

## 14.1   Labels

> **!** To enable the redefined \label macro, specify the package option redefinelabel[→ P. 10]

\label[⟨*cleveref-counter-overrule*⟩]{⟨*label name*⟩}[⟨*options for associated counters*⟩]    v1.2 2017-03-03

>   all=⟨*true,false*⟩                                                       (initially ⟨*false*⟩)    v1.2 2017-03-03
>
>   This will enable that all associated counters to a driver counter will cause the generation of    v1.2 2017-03-03
>   a label too. By default this option is ⟨*false*⟩. This option deliberately overrules select, the
>   value of the option prefix[→ P. 50] is disregarded.
>
>   select=counter1, counter2,...                                            (initially empty)    v1.2 2017-03-03
>                                                                                                  v1.2 2017-03-03

Select only some of the associated counters to be able to be labeled. As of version 1.7 there is no check whether the given names refer to counters at all or are associated counters to the last counter that has been used with `\refstepcounter`.

`prefix`=⟨*text*⟩ (initially empty) <sup>v1.2 2017-03-03</sup>

This gives the prefix of the label of the associated counter. If the option `all`<sup>→P.49</sup> is en- <sup>v1.2 2017-03-03</sup> abled, the label name is generated from the name of the associated counter, the value of `prefix-sep` and the value of the 2nd argument.

```
% Assume that some counter has the associated counters foobar, ↲
↳morefoobar and yetanotherfoobar
\label{foo}[prefix=assoc,all]
```

will cause a label named `foobar::foo`, `morefoobar::foo` and `yetanotherfoobar::foo`, whereas

```
\label{foo}[prefix=assoc,select=morefoobar]
```

would generate the label `assocc::foobar` only and will be tied to the value of the counter `morefoobar`

`prefix-sep`=⟨*text*⟩ (initially ::) <sup>v1.2 2017-03-03</sup>

Defines the separator between the `prefix` and the label name for the driver counter, i.e. <sup>v1.2 2017-03-03</sup> the 2nd argument of the `\label`<sup>→P.49</sup> command.

> **!** If the package option `redefinelabel`<sup>→P.10</sup> is set to ⟨*false*⟩, the usage of the third optional argument will leave spurious content at the position `\label` was used. The reason is that the content of third optional argument with `[]` is not recognized as an argument any longer.

`\LaTeXLabel`[⟨*cleveref-counter-overrule*⟩]{⟨*label name*⟩} <sup>v1.3 2017-03-04</sup>

This is the default label macro, either with the ☑ **cleveref** extension or the classical LaTeX2e macro (eventually modified by ☑ **hyperref** ) and is not modified by this package.
The feature of label hooks from `\RegisterPreLabelHook`<sup>→P.51</sup> or `\RegisterPostLabelHook`<sup>→P.51</sup> is not used here.

## 14.2 Hooks

This feature is experimental and only realized for the modified `\label`<sup>→P.49</sup> command until now. See `xassoccnt_getparentcounter_example.tex` as an example of usage.

\RegisterPreLabelHook{⟨*command name1, command name2,. . .*⟩}                           v1.3 2017-03-04

This macro declares a possible list of hooks (commands) that should be executed **before** the traditional \label command is applied. If the hook name refers to some unknown macro, nothing is done.

The hook names must be given with the \ as command sequence indicator, i.e. \zlabel, more than one macro name is possible by using comma as separator.

As of version 1.7 the hook macro does not allow more than one argument, which is automatically used from the surrounding \label call and is the usual label name.

\RegisterPostLabelHook{⟨*command name1, command name2,. . .*⟩}                          v1.3 2017-03-04

This macro declares a possible list of hooks (commands) that should be executed after the traditional \label command is applied. If the hook name refers to some unknown macro, nothing is done.

The hook names must be given with the \ as command sequence indicator, i.e. \zlabel, more than one macro name is possible by using comma as separator.

As of version 1.7 the hook macro does not allow more than one argument, which is automatically used from the surrounding \label call and is the usual label name.

# Part IV
# Meta-Information

## Table of Contents

# 15  To - Do list

- Merging of counter groups, removing counters from counter groups

- Backup and restoration of individual counters not being member of a counter group

- Switch to the container support for all features – this is a major task and will be done in (tiny) steps.

- Better counter definition/copy counter routines $\longrightarrow$ another package perhaps

- More examples

- Some macro names might be non-intuitive

- Improve documentation

- Hooks for conditionals on ▦ **numberofruns** (see section 13.3)

Some issues that have been addressed partially are:

- Add counter group support for the ◆ backup feature, i.e. define a symbolic name for a group of counters that should be controlled by the backup feature. This will allow multiple backup groups, which might be necessary. <span>v1.0 2016-07-28</span>

If you

- find bugs

- errors in the documentation

- have suggestions

- have feature requests

don't hesitate and contact me using my mail address: typography.with.latex@gmail.com.

© Dr. Christian Hupfer

# 16   Acknowledgments

I would like to express my gratitudes to the developers of fine LaTeX packages and of course to the users at tex.stackexchange.com, especially to

- Paulo Roberto Massa Cereda

- Enrico Gregorio

- Joseph Wright

- David Carlisle

and many others for their invaluable help on many questions on macros.

# 17   Version history

- Version v1.7 2018-12-28

  – Maintenance update due to updates to ✉ **expl3** .
    Changed from `\c_minus_one` to an explizit `-1`

- Version v1.6 2018-01-03

  – Maintenance update due to updates to ✉ **expl3**

  – The Backup/Restore feature is currently broken and needs a thorough inspection

- Version v1.5 2017-07-28

  Added `\LastRefSteppedCounter`→P.17 as dummy version which expands to nothing as long as `\refstepcounter` has not been called.

- Version v1.4 2017-05-10

  – Improved the core macros `\refstepcounter` and `\stepcounter` in order to fit the ✉ **expl3** and ✉ **xparse** changes of Februar - April 2017.

  – Added following experimental features:

  – `\CounterFormat`→P.25 with quick and possible recursive change of the counter output

  – `\StoreCounterFormats`→P.28, `\AddCounterFormats`→P.29 and `\RemoveCounterFormats`→P.29 for defining own short hand counter formats.

  – Provided the macros `\xalphalph`→P.25 and `\xAlphAlph`→P.25 in order to allow counter output in the same manner as the ✉ **alphalph** does.

  – Added the macros `\CounterWithin`→P.24, `\CounterWithin*`→P.24, `\CounterWithout`→P.24 and `\CounterWithout*`→P.24 which provides a quicker access to add or remove counters from the reset list and changing the corresponding `\the...` macros.

  – Provided the `\LoopCounterResetList`→P.23 to perform the same action on all counters being in the reset list of a given counter.

  – New macros `\ClearCounterResetList`→P.19 and `\ClearCounterResetList*`→P.19 to remove all counters on first level of a driver counter.

  – Added the explanation (missing in previous versions) to the documentation that the `\AddToReset`→P.19, `\RemoveFromReset`→P.18 and `\RemoveFromFullReset`→P.18 macros actually support a comma separated list of counter names for the first argument.

- Version v1.3 2017-03-04

  - Provided the `\LaTeXLabel`<sup>→P.50</sup> macro to access the non-xassoccnt version of the `\label`<sup>→P.49</sup> command.

  - Added the concept of label hooks, see section 14 for more information.  <span style="background-color: yellow">Experimental</span>

  - The macros `\Last...` are defined with ✉ **expl3** methods.

  - Added `\GetAllResetLists`<sup>→P.21</sup> and `\GetParentCounter`<sup>→P.21</sup> for information on parent (or driver) counters.

- Version v1.2 2017-03-03

  - Corrected some typos in the manual.

  - The macros `\NewDocumentCounter`<sup>→P.12</sup>, `\DeclareDocumentCounter`<sup>→P.12</sup> and `\NewTotalDocume` allow multiple counters to be specified and defined.

  - Added the macro `\DeclareTotalAssociatedCounters`<sup>→P.34</sup> in order to combine total counters and the associated feature, i.e. the counters are total ones and associated to a driver counter.

  - An extended version of `\label`<sup>→P.49</sup> is provided to allow labels also for associated counters <span style="background-color: yellow">Experimental</span> during the stepping process of the driver counter.

- Version v1.1 2016-10-29

  - Added some missing basic functions needed after the more restrictive ✉ **expl3** update from 2016/10/19

  - Added a statement about the requirement to load ✉ **tcolorbox** before ✉ **xassoccnt** in the documentation (i.e. this file!)

- Version v1.0 2016-07-28

  - Restructured the ✉ **xassoccnt** manual file.  <span style="background-color: #aaaaff">v1.0 2016-07-28</span>

  - Added some improvements for counter reset lists macros  <span style="background-color: #aaaaff">v1.0 2016-07-28</span>

  - Added new backup/restore features, with cascading counters possibility – the old backup/restore macros are still available but renamed with a prefix `\Former...`  <span style="background-color: #aaaaff">v1.0 2016-07-28</span>

  - Added the `\RemoveAllPeriodicCounters`<sup>→P.42</sup> – it was missing in the 🏷 periodic counter features – see Periodic counters for more information on this.  <span style="background-color: #aaaaff">v1.0 2016-07-28</span>

  - Added the expandable version of `\TotalCounterInternalName`<sup>→P.46</sup> named `\TotalCounterInternalNameExp`<sup>→P.46</sup>.  <span style="background-color: #aaaaff">v1.0 2016-07-28</span>

- Version v0.9 2016-06-19

  – \TotalValue$^{\rightarrow P.47}$ is an expandable command now.

  – Added the 🏷 periodic counter features – see Periodic counters for more information on this.

  > v0.9
  > 2016-06-19

  > v0.9 2016-06-19

- Version v0.8 2016-06-10

  ✓ Fixed the \SuspendCounters$^{\rightarrow P.43}$ and \ResumeSuspendedCounters$^{\rightarrow P.44}$ macros – the comma separated list of counters was not used (contrary to the purpose and the documentation description).

  ✓ Additions of commands

  ▷ \ResumeAllSuspendedCounters$^{\rightarrow P.44}$

  ▷ \CascadeSuspendCounters$^{\rightarrow P.43}$

  ▷ \DisplayResetList$^{\rightarrow P.20}$

  ▷ \ShowResetList$^{\rightarrow P.20}$

  > v0.8
  > 2016-06-10

  > v0.8 2016-06-10

- Version v0.7 2016-05-10

  ✓ Fixed a small bug in the ✉ **xassoccnt** version of \stepcounter

  ✓ Added some macros that support the output of binary, octal or hexadecimal (both lower/uppercase) values of counters.

  ✓ Added the \Loop...Counters macros that perform an action in loop on all given counter names.

  > v0.7
  > 2016-05-10

  > v0.7 2016-05-10

  > v0.7 2016-05-10

- Version v0.6 2016-03-05

  ✓ The coupled counters allow to specify a counter group to which all relevant counters belong, this allows several coupled counter groups then

  ✓ Fixed a small bug within backup counter support – the resetting was not done any more

  ✓ Added the 🔧 **nonumberofruns** package option.

  > v0.6 2016-03-05

  > v0.6 2016-03-05

- Version v0.5 2016-02-27

  ✓ Added support (very experimental!) for the 🏷 coupled counters feature, see section 9 about this feature!

  ✓ Added \RegisterTotalDocumentCounter and improved \TotalValue support

  > v0.5 2016-02-27

  > v0.5 2016-02-27

- Version v0.4 2016-01-26

  ✓ Added `\BackupCounterValues` and `\RestoreCounterValues` support

  ✓ Added `\StepDownCounter` and `\SubtractFromCounter` macros

- Version v0.3 2016-01-08

  ✓ Added the 🏷 totalcounter features similar to the packages ✉ **totcount** or ✉ **totalcount**

  ✓ Added the 🏷 super total counter features

  ✓ Added the ▦ **numberofruns** counter

- Version v0.2 2016-11-14

  Improved `\stepcounter` to remove some incompatibilities with the ✉ **perpage** . This is only partially managed so far.

- Version v0.1 2016-11-07

  A major bug fixed due to some error in usage together with ✉ **calc** when the driven counters are not stepped any longer. The culprit was in ✉ **assoccnt** that the counter reset list was not really disabled.

  Thanks to this question `http://tex.stackexchange.com/questions/269731/calc-breaks-assoccn` this bug was detected.

  This however lead to some internal inconsistencies and it was decided to rewrite ✉ **assoccnt** with ✉ **expl3** and the features of the new LaTeX 3 - Syntax.

v0.4 2016-01-26

v0.4 2016-01-26

v0.3 2016-01-08

2015-11-11

2015-25-11

2015-25-11

v0.2 2016-11-14

# Part V

# Appendix

Note: The \DeclareAssociatedCounters command has to be used in the preamble of the document. It's missing here for the sake of a compact example.

## A   Example: Total number of sections

In this example, all sections of this document are counted, i.e. the current one as well as all following ones.

```
This document has \total{totalsections} section(s)%
```
---
This document has 20 section(s)

## B   Example: Total number of subsections with suspension

In this example, the subsections of this document are counted but later on, the associatedcounter is removed from the list, so it is frozen.

© Dr. Christian Hupfer

```
\subsection{First dummy subsection}
SubSection counter: \thesubsection~-- \number\totvalue{totalsubsections}
\subsection{Second dummy subsection}
SubSection counter: \thesubsection~-- \number\totvalue{totalsubsections}

\RemoveAssociatedCounter{subsection}{totalsubsections}%
\subsection{Third dummy subsection after removing the associated counter}

SubSection counter: \thesubsection~-- \number\totvalue{totalsubsections}
```

## B.1   First dummy subsection

SubSection counter: B.1 – 30

## B.2   Second dummy subsection

SubSection counter: B.2 – 30

## B.3   Third dummy subsection after removing the associated counter

SubSection counter: B.3 – 30

## B.4   Suspension of a non-associated counter

This example will show the suspension of a non-associated counter

```
\setcounter{equation}{0}%
\SuspendCounters{equation}%
\begin{equation}
E_{0} = mc^2
\end{equation}

\begin{equation}
E^2 = \left({ pc}\right)^2 + E^{2}_{0}
\end{equation}

\begin{equation}
  m(v) = \frac{m_{0}}{\sqrt{1-\frac{v^2}{c^2}}}
\end{equation}


There are \number\value{equation}~equations in here!
```

$$E_0 = mc^2 \tag{0}$$

$$E^2 = (pc)^2 + E_0^2 \tag{0}$$

$$m(v) = \frac{m_0}{\sqrt{1 - \frac{v^2}{c^2}}} \tag{0}$$

There are 0 equations in here!

# C   Former backup and restore of counter values

## C.1   Description of backup and restoring macros for counter values

\FormerBackupCounterValues[⟨*options*⟩]{⟨*counter name1, counter name2,...*⟩}

This macro adds counter names (separated by a comma) to a list and stores the current values of the counters to another list. The values are used from the current state where this command is used, not a previous or a later state is stored.

- All counters in the list will be reset to zero (after storing the values) for the next usage, unless the `resetbackup` key is set to ⟨*false*⟩.

- Multiple specification of the same counter name is possible, but only the first occurence will be regarded – consecutive occurences of the same counter name are not taken into account.

resetbackup=⟨*true/false*⟩ (initially true)

---

This key decides whether **all** counters in the backup list should be reset to zero or should keep the current value. The default value is ⟨*true*⟩.

Please note: If a name does not belong to a counter register the compilation aborts with an error message!

Some remarks

> **!** If a specific counter name is suffixed with an ∗ at its end the resetting is disabled for this particular counter, regardless whether `resetbackup`→P.61 is set to true or not.

v0.4 2016-01-26

> **!** Strangely enough, a counter name like ▦ **foo**∗ is possible, but `\thefoo∗` would fail. Be careful about choosing counter names for new counters – just restrict yourself to the usual letters (and if really needed, using @)

## \FormerRestoreAllCounterValues[⟨*options*⟩]

This macro restores all stored counter values corresponding to the counter names.

v0.5 2016-02-27

As of version 1.7 the optional argument isn't used and reserved for later purposes.

The backup list is cleared after the restoring has been finished.

> **!** The command `\FormerRestoreAllCounterValues` was previously called `\FormerRestoreCounterValues` – that macro is now reserved for updating only particular counters, not all in a row.

## \FormerRestoreCounterValues[⟨*options*⟩]{⟨*counter name1,counter name2,...*⟩}

This macro restores only the stored counter values given by the counter names. As of version 1.7 the optional argument isn't used and reserved for later purposes.

v0.5 2016-02-27

```
   \captionof{figure}{A dummy figure}

   \captionof{table}{A dummy table}


\FormerBackupCounterValues{figure,table*}

   \captionof{figure}{Another dummy figure}

   \captionof{table}{Another dummy table}

   \captionof{figure}{Even another dummy figure}

   \captionof{table}{Even another dummy table}

Before restoring: \thefigure\ and \thetable

\FormerRestoreAllCounterValues

Restored the values: \thefigure\ and \thetable

\captionof{figure}{Yet another dummy figure}
\captionof{table}{Yet another dummy table}
```

Figure 1: A dummy figure

Table 2: A dummy table

Figure 1: Another dummy figure

Table 3: Another dummy table

Figure 2: Even another dummy figure

Table 4: Even another dummy table

Before restoring: 2 and 4
Restored the values: 1 and 2

Figure 2: Yet another dummy figure

Table 3: Yet another dummy table

\FormerAddBackupCounter[⟨*options*⟩]{⟨*counter name1,counter name2,...*⟩}

v0.5 2016-02-27

© Dr. Christian Hupfer

This is similar to `\FormerBackupCounterValues`<sup>→P.61</sup>, but adds the counter names to an existing global list and can be applied after `\FormerBackupCounterValues`<sup>→P.61</sup>.

`\FormerRemoveBackupCounters[`⟨*options*⟩`]{`⟨*counter name1, counter name2,...*⟩`}`

This macro removes the counters from the list of backup counters and immediately restores the counter value unless the starred version `\FormerRemoveBackupCounters*` is used.

If the package ✉ **hyperref** is used, the macro `\theH...` is restored to the original meaning.

As of version 1.7 the optional argument isn't used and reserved for later purposes.

`\FormerRemoveBackupCounters*[`⟨*options*⟩`]{`⟨*counter name*⟩`}`

This command is basically similar to `\FormerRemoveBackupCounters`, but does not restore the counter value right at the place the macro is used.

As of version 1.7 the optional argument isn't used and reserved for later purposes.

# Index