The counterz package*

Christopher McClain†

Released 2023/06/05

Abstract

The counterz package provides additional tools for manipulating counters. The package facilitates the use of stealth prefixes for counter names in order to help distinguish between counters from multiple input files. The package also provides a means to generate random counters and save such counter values for future typesetting.

Contents

*This file describes version v1.1.1, last revised 2023/06/05.

†E-mail: christopher.mcclain@mail.wvu.edu

1 Introduction

1.1 About

This project emerged from the author's frequent use of LaTeX counters as traditional integer type variables when generating mathematics documents with random elements. While pdfTeX primitives such as `\pdfuniformdeviate` may be used to generate random integers, these integer values will be randomized with every typesetting. The counterz package provides a way to save the values of counters. Another .tex file is created so that, if desired, it can be inputted upon a subsequent typesetting in order to initialize the counters with the previously generated values. A boolean variable and accompanying commands allow an author to toggle between reusing and rerandomizing counters.

One of the consequences of preloading counter values in large projects with multiple source files is that one must take care to use distinct counter names throughout all of the different files. If the file Main.tex inputs File1.tex and File2.tex, and both input files define the counter *mycounter*, then this could result in typesetting errors. One way to address this problem is to prefix every counter name with the file name or some other marker so that the counter names will actually be distinct. For example, *File1mycounter* is distinct from *File2mycounter*. Very long counter names, however, can make code difficult to read and hinder consistent application of this practice. The counterz package provides a way to stealthily define and recall such prefixes so that the shorter non-prefixed names can be used for the manipulation, recall, and typesetting of counters.

1.2 License

1.3 Installation

Run (pdf)TeX on counterz.dtx to generate the file counterz.sty, and copy it to your local texmf directory. To generate both the package file counterz.sty and the documentation counterz.pdf, run (pdf)LaTeX on counterz.dtx. Typesetting the documentation requires the package hypdoc which is included in TeX distributions and at The Comprehensive TeX Archive Network.

2 User Guide

To use this package, include the following line in the preamble of your document:

```
\usepackage{counterz}
```

The package counterz loads the packages etoolbox and makecmds, both of which are included in TeX distributions and at The Comprehensive TeX Archive Network.

2.1 Counter Prefixes

\setcounterprefix

Counter prefixes are stored in an internal macro whose default value is an empty string. The command `\setcounterprefix{`⟨*prefix*⟩`}` is used to change this value. For example, to change the prefix to *PurpleMonkey*, use

```
\setcounterprefix{PurpleMonkey}
```

and to change it from *PurpleMonkey* to *Dishwasher*, use

```
\setcounterprefix{Dishwasher}
```

\clearcounterprefix

The command `\clearcounterprefix` returns the prefix to its empty default.

2.2 Manipulating Counters

\xnewcounter
\xsetcounter

The command `\xnewcounter{`⟨*countername*⟩`}` creates a counter with a prefixed name. The command `\xsetcounter{`⟨*countername*⟩`}{`⟨*integer*⟩`}` assigns the specified value to the counter with the prefixed name. For example, suppose that the file BoringFile1.tex contains the following:

```
\xnewcounter{bestcounterever}
\xsetcounter{bestcounterever}{100}
```

and suppose that the file BoringFile2.tex contains the following:

```
\xnewcounter{bestcounterever}
\xsetcounter{bestcounterever}{-29}
```

and, finally, suppose that the file Main.tex contains (in part) the following:

```
\setcounterprefix{PurpleMonkey}
\input{BoringFile1}
\setcounterprefix{Dishwasher}
\input{BoringFile2}
```

Then typesetting Main.tex will create a counter *PurpleMonkeybestcounterever* with the value 100 and a counter *Dishwasherbestcounterever* with the value -29. By using commands `\xnewcounter` and `\xsetcounter` instead of `\newcounter` and `\setcounter`, BoringFile1.tex and BoringFile2.tex may be written independently without considering any counter name conflicts. The distinction between the counters is determined by the prefixes defined in the file Main.tex. By changing prefixes, Main.tex can even input the same file multiple times without conflict.

\xprovidecounter
\xaddtocounter
\xvalue

The commands `\xprovidecounter`, `\xaddtocounter`, and `\xvalue` are likewise prefix versions of commands `\providecounter`, `\addtocounter`, and `\value`, respectively. When the prefix is empty, the commands expand like their standard counterparts. (Note: `\providecounter` defines a counter if it has not already been defined. See the documentation for the package makecmds for details.)

2.3 Conditional Statements

\ifctrequal The command \ifctrequal{⟨*counter1*⟩}{⟨*counter2*⟩}{⟨*foo*⟩}{⟨*bar*⟩} uses the command \xvalue to compare the values of the (prefixed) counters and then executes ⟨*foo*⟩ if the values are equal and otherwise executes ⟨*bar*⟩. The commands \ifctrless and \ifctrmore work analogously, based on whether the value of prefixed ⟨*counter1*⟩ is less than that of of prefixed ⟨*counter2*⟩ or more than that of prefixed ⟨*counter2*⟩, respectively. Consider the example code

```
\setcounterprefix{TigerTiger}
\xnewcounter{Small}
\xsetcounter{Small}{7}
\xnewcounter{Large}
\xsetcounter{Large}{11}
\ifctrequal{Small}{Large}{January}{February}
\ifctrless{Small}{Large}{March}{April}
\ifctrmore{Small}{Large}{May}{June}
```

which produces the output

February March June

because the value of the counter *TigerTigerSmall* is 7 which is less than 11, the value of the counter *TigerTigerLarge*.

\ifctrzero The command \ifctrzero{⟨*counter*⟩}{⟨*foo*⟩}{⟨*bar*⟩} executes ⟨*foo*⟩ if the value of the (prefixed) counter is zero and otherwise executes ⟨*bar*⟩. The commands \ifctrneg and \ifctrpos work analogously based on whether the value is negative or positive, respectively. The example code

```
\setcounterprefix{TigerTiger}
\xprovidecounter{Small}
\xsetcounter{Small}{7}
\ifctrzero{Small}{January}{February}
\ifctrneg{Small}{March}{April}
\ifctrpos{Small}{May}{June}
```

produces the output

February April May

because the value of the counter *TigerTigerSmall* is 7 which is positive (and thus nonzero, as well).

2.4 Displaying Counters

\xarabic The command \xarabic{⟨*counter*⟩} is simply a prefix version of the standard display command \arabic. The commands \xroman, \xRoman, \xalph, \xAlph, and \xfnsymbol are likewise prefix versions of the standard display commands

\xalph
\xAlph
\xfnsymbol

\roman, \Roman, \alph, \Alph, and \fnsymbol, inheriting the restrictions of their parent commands.

Note that the code

```
\setcounterprefix{Sneaky}
\xprovidecounter{Pete}
\xsetcounter{Pete}{42}
\arabic{Pete}
```

produces an error because the counter *Pete* is not defined, but the code

```
\setcounterprefix{Sneaky}
\xprovidecounter{Pete}
\xsetcounter{Pete}{42}
\xarabic{Pete}
```

produces the output

42

which is the value of the counter *SneakyPete*. The code

```
\setcounterprefix{Sneaky}
\xprovidecounter{Pete}
\xsetcounter{Pete}{42}
\clearcounterprefix
\xarabic{Pete}
```

also generates error because the final line is trying to use the undefined counter *Pete* after the prefix was returned to its default value.

In addition to prefix versions of the standard display commands, the package counterz defines some variants of \xarabic that are useful in the display of mathematical expressions. For example, consider the following code:

```
\xprovidecounter{a}
\xsetcounter{a}{5}
\xprovidecounter{b}
\xsetcounter{b}{0}
\xprovidecounter{c}
\xsetcounter{c}{-7}
$\xarabic{a}+\xarabic{b}+\xarabic{c}$
```

which produces

$5 + 0 + -7$

Using \arabicx causes the expression to contain the consecutive pair $+-$. The

\xsigned

command \xsigned{⟨*counter*⟩} is like \xarabic except that nonnegative values are preceded by a plus sign "+". The code

```
$\xarabic{a}\xsigned{b}\xsigned{c}$
```

produces

$5 + 0 - 7$

\xsignednz If we wish to suppress the 0, we can instead use the command \xsignednz{⟨*counter*⟩} which is a nonzero version of \xsigned and, if desired or necessary, the command
\xarabicnz \xarabicnz{⟨*counter*⟩} which is a nonzero version of \xarabic. The code

```
$\xarabicnz{a}\xsignednz{b}\xsignednz{c}$
```

produces

$5 - 7$

\xnegof The command \xnegof{⟨*counter*⟩} displays the negative of ⟨*counter*⟩. The com-
\xnegofnz mand \xnegofnz does the same except that it suppresses the number zero. The
\xnegsigned command \xnegsigned includes the appropriate signs of plus "+" and minus "-"
\xnegsignednz (assigning a minus to zero in this case). Finally, the command \xnegsignednz does the same except that it suppresses the number zero., as demonstrated by the following code:

```
\xprovidecounter{d}
\xsetcounter{d}{-2}

$\xarabic{a}\xsigned{b}\xsigned{c}=\xarabic{d}$

$\xnegof{d}=\xnegof{a}\xnegsigned{b}\xnegsigned{c}$

$\xnegofnz{d}=\xnegofnz{a}\xnegsignednz{b}\xnegsignednz{c}$
```

which produces

$5 + 0 - 7 = -2$
$2 = -5 - 0 + 7$
$2 = -5 + 7$

The preceding commands for displaying values related to counters were created by using some other commands that we make available in case they prove useful.
\xabsof The command \xabsof{⟨*counter*⟩} prints the absolute value of ⟨*counter*⟩. The
\xsignof command \xsignof{⟨*counter*⟩} prints a minus sign "-" if ⟨*counter*⟩ is negative and otherwise prints a plus sign "+". (Note that the latter case includes the value
\xnegsignof zero.) The command \xnegsignof{⟨*counter*⟩} prints a plus sign "+" if ⟨*counter*⟩ is negative and otherwise prints a minus sign "-". (Note that the latter case includes the value zero.)

Additional variants of these commands suppress certain output, as is conventional when using integers as coefficients in algebraic expressions. The com-
\xabsofcoef mand \xabsofcoef{⟨*counter*⟩} prints the absolute value of ⟨*counter*⟩ except that
\xsignofcoef it suppresses the values of 1 and 0. The command \xsignofcoef{⟨*counter*⟩}

prints the sign of ⟨*counter*⟩ if the value of ⟨*counter*⟩ is nonzero. The command

\xnegsignofcoef `\xnegsignofcoef{`⟨*counter*⟩`}` prints the opposite sign of ⟨*counter*⟩ if the value of ⟨*counter*⟩ is nonzero. These commands are used to build versions of `\xarabic` and `\xsigned` specific to typesetting coefficients, as we now illustrate.

Consider the following code

```
\xprovidecounter{a0}
\xsetcounter{a0}{-10}
\xprovidecounter{a1}
\xsetcounter{a1}{1}
\xprovidecounter{a2}
\xsetcounter{a2}{-5}
\xprovidecounter{a3}
\xsetcounter{a3}{-1}
\xprovidecounter{a4}
\xsetcounter{a4}{0}
\xprovidecounter{a5}
\xsetcounter{a5}{11}
$\xarabic{a5}x^5 + \xarabic{a4}x^4 + \xarabic{a3}x^3 + \xarabic{a2}x^2
                 + \xarabic{a1}x   + \xarabic{a0}    = 42$
```

and its output

$$11x^5 + 0x^4 + -1x^3 + -5x^2 + 1x + -10 = 42$$

We seek a better way to handle the coefficients, especially 1 and −1. The command

\xcoef `\xcoef{`⟨*counter*⟩`}` prints the value of ⟨*counter*⟩ except that it suppresses the values of 1, 0, and -1, printing a minus sign "-" in the latter case. The command

\xsignedcoef `\xsignedcoef{`⟨*counter*⟩`}` is like `\xcoef` except that positive values are preceded by a plus sign "+". We use these to write the code

```
$\xarabic{a5}x^5 + \xarabic{a4}x^4 + \xarabic{a3}x^3 + \xarabic{a2}x^2
                 + \xarabic{a1}x   + \xarabic{a0}    = 42$

$\xcoef{a5}\ifctrzero{a5}{}{x^5}
   \xsignedcoef{a4}\ifctrzero{a4}{}{x^4}
   \xsignedcoef{a3}\ifctrzero{a3}{}{x^3}
   \xsignedcoef{a2}\ifctrzero{a2}{}{x^2}
   \xsignedcoef{a1}\ifctrzero{a1}{}{x}
   \xsignednz{a0}
   = 42$
```

whose output is

$$11x^5 + 0x^4 + -1x^3 + -5x^2 + 1x + -10 = 42$$
$$11x^5 - x^3 - 5x^2 + x - 10 = 42$$

\xnegcoef The command `\xnegcoef{`⟨*counter*⟩`}` prints the negative of the value of ⟨*counter*⟩ except that it suppresses the values of 1, 0, and -1, printing a "-" in the

\xnegsignedcoef

latter case. The command \xnegsignedcoef{⟨*counter*⟩} is like \xnegcoef except that positive values are preceded by a plus sign "+". We use these to write the code

```
$\xcoef{a5}\ifctrzero{a5}{}{x^5}
   \xsignedcoef{a4}\ifctrzero{a4}{}{x^4}
   \xsignedcoef{a3}\ifctrzero{a3}{}{x^3}
   \xsignedcoef{a2}\ifctrzero{a2}{}{x^2}
   \xsignedcoef{a1}\ifctrzero{a1}{}{x}
   \xsignednz{a0}
   = 42$

$\xcoef{a5}\ifctrzero{a5}{}{x^5}
   \xsignedcoef{a4}\ifctrzero{a4}{}{x^4}
   \xsignedcoef{a2}\ifctrzero{a2}{}{x^2}
   \xsignednz{a0}
   = \xnegcoef{a3}\ifctrzero{a3}{}{x^3}
   \xnegsignedcoef{a1}\ifctrzero{a1}{}{x}
   +42$
```

whose output is

$$11x^5 - x^3 - 5x^2 + x - 10 = 42$$
$$11x^5 - 5x^2 - 10 = x^3 - x + 42$$

As the reader has probably already observed in the code above, these display commands appear to be less efficient than a manual adjustment of signs and numbers. For fixed, known values of counters, this assessment is correct. The real utility of these commands is not apparent until they are combined with randomly generated counter values.

2.5 Random Counters

\randomizectr
\norandomizectr
\ifrandomizectr

In order to effectively manage the options of randomizing counter values or reusing counter values, the commands \randomizectr and \norandomizectr are used to toggle an internal boolean variable. The internal boolean is initialized as TRUE when the counterz package is loaded. A conditional command \ifrandomizectr{⟨*foo*⟩}{⟨*bar*⟩} executes ⟨*foo*⟩ when the boolean is TRUE and otherwise executes ⟨*bar*⟩.

\randsetcounter
\xrandsetcounter
\randaddtocounter
\xrandaddtocounter

We next define random versions of \setcounter and \addtocounter. These commands will only execute when the document is set to randomize. The command \randsetcounter{⟨*counter*⟩}{⟨*min*⟩}{⟨*max*⟩} assigns to ⟨*counter*⟩ a random integer value between ⟨*min*⟩ and ⟨*max*⟩. The command \xrandsetcounter is a prefix version of \randsetcounter. Analogously, we define the command \randaddtocounter{⟨*counter*⟩}{⟨*min*⟩}{⟨*max*⟩} which adds to ⟨*counter*⟩ a random integer value between ⟨*min*⟩ and ⟨*max*⟩. \xrandaddtocounter is a prefix version of \randaddtocounter. The following code produces an expression in the form $ax + b$, where a and b are random integers between -10 and 10:

```
\randomizectr
\xprovidecounter{a}
\xprovidecounter{b}
\xrandsetcounter{a}{-10}{10}
\xrandsetcounter{b}{-10}{10}
$\xcoef{a}\ifctrzero{a}{\xarabic{b}}{x \xsignednz{b}}$
```

Organized in the following table are fifty instances of output that are randomly generated by the typesetting of this document:

$-7x-7$	$2x+10$	$10x-6$	$10x-10$	$2x-2$
$3x+2$	$4x-10$	$x-7$	$5x+4$	$6x-3$
$9x-7$	$-5x-8$	0	$-6x-1$	$-x+9$
$-2x$	$2x-8$	$6x-2$	$-10x+9$	-9
$6x$	$x-4$	$-x-5$	-6	$2x+5$
$-x-4$	$7x+4$	$10x+2$	$3x+1$	$9x-6$
$6x-8$	$4x-7$	$9x+7$	$-7x+7$	$-9x-9$
$4x-3$	$7x+7$	$4x+1$	$-2x+9$	$3x+2$
$x-8$	$-9x+6$	8	$-3x-1$	7
$-x+7$	$10x$	-2	$8x+10$	$3x-2$

If our document contains randomly generated counters, but we wish to typeset the document again without changing those values, then we need a way to save them. The command `\opencountersfile` creates and opens the write stream to the file ⟨*jobname*⟩.counters.tex to store the necessary information. For example, if the document is named Yellowdog.tex, then the previously generated counters and their assigned values will be stored the file Yellowdog.counters.tex. The author only has to include this command once, prior to any commands used to save the counter values. Additional instances of `\opencountersfile` will report an error, as will trying to use the command when the document is set to not randomize (e.g. `\norandomizectr`). These error reports are designed to prevent the accidental overwriting of ⟨*jobname*⟩.counters.tex.

`\opencountersfile`

After opening the write stream to ⟨*jobname*⟩.counters.tex, the command `\savecounter{`⟨*counter*⟩`}` may be used to "save" the value of *counter* by writing to the file the relevant `\providecounter` and `\setcounter` commands. The command `\xsavecounter` is a prefix version of `\savecounter`. When using `\xsavecounter`, the commands that are written to the file include the necessary counter prefixes. Consequently, an author can, if necessary or desired, manually search the file for the value assigned to any randomly generated counter.

`\savecounter`

`\xsavecounter`

Once we have generated a file for storing counters, we need a way to recover those values during a subsequent typesetting. The command `\inputcountersfile` will input the necessary file, if it exists, and report an error if it does not. Keep in mind that inputting the file will override any previous assignments of those counters, so it is probably best to invoke this command near the beginning of a document. For example, after including an instance of either `\randomizectr` or `\norandomizectr`, a document named Yellowdog.tex might include the code

`\inputcountersfile`

```
\ifrandomizectr{\opencountersfile}{\inputcountersfile}
```

to determine whether to preload previously stored counter values or open the write stream in anticipation of randomly generating new counter values.

\promptrandomizectr The command \promptrandomizectr[⟨*macro*⟩]{⟨*message*⟩}{⟨*string*⟩} offers an alternative to manually switching between the commands \randomizectr and \norandomizectr for different typesettings. The contents of ⟨*message*⟩ are displayed in the terminal, awaiting a response from the user at the prompt ⟨*macro*⟩. If the optional argument is not used then the default prompt is \@typein. If the optional argument is given, it must be a macro name that includes the backslash. The user's response is stored as a string in ⟨*macro*⟩ and compared to ⟨*string*⟩. If they are equal, then the command \randomizectr is executed. If they are not equal then \norandomizectr is executed.

Consider the following example code:

```
\promptrandomizectr[\EnterResponse]{%
   ^^J Enter 1 to randomize.
   ^^J Enter 2 to not randomize.
}{%
   1%
}%
\ifrandomizectr{%
   \opencountersfile
}{%
   \inputcountersfile
}%
```

which displays the following in the terminal:

```
 Enter 1 to randomize.
 Enter 2 to not randomize.

\EnterResponse=
```

Notice that the first (optional) argument \EnterResponse begins with a backslash and is displayed with an equals sign "=" at the prompt. Also note that the second argument contains two instances of the text ^^J which is used to produce a line break in the terminal output. Next, note that the third argument 1 is immediately followed by a percent symbol % to prevent extra space being included in the string. (If the 1 was immediately followed by a closing brace instead of a line break in the code, the percent symbol would not be used.) Finally, note that if the user types a 1 in the terminal and presses Enter, then the commands \randomizectr and \opencountersfile will be executed. If the user enters *any other text* or simply presses Enter with no text, then the commands \norandomizectr and \inputcountersfile will be executed, despite the instructions to enter a 2 to achieve this outcome.

\randprovidecounter The command \randprovidecounter{⟨*counter*⟩}{⟨*min*⟩}{⟨*max*⟩} combines the four commands \providecounter, \ifrandomizectr, \randsetcounter, and \savecounter. The command creates ⟨*counter*⟩ if it has not already been defined and, if the document is randomized, assigns to ⟨*counter*⟩ a random integer value between ⟨*min*⟩ and ⟨*max*⟩ and saves this value to the counters

\randprovidecounternz
\xrandprovidecounter
\xrandprovidecounternz

file. The command \randprovidecounternz is like \randprovidecounter except that the generated value is nonzero. The commands \xrandprovidecounter and \xrandprovidecounternz are prefix versions of \randprovidecounter and \randprovidecounternz, respectively. Suppose that Neverending.tex contains the code

```
\randomizectr
\ifrandomizectr{\opencountersfile}{}
\setcounterprefix{Southern}
\xrandprovidecounternz{Oracle}{-10}{10}
\xcoef{Oracle}x+42
```

After typesetting once, the resulting document might display an expression such as $-9x + 42$ and print to Neverending.counters.tex the line

```
\providecounter {SouthernOracle} \setcounter {SouthernOracle}{-9}
```

After typesetting a second time, the resulting document might display $4x+42$ and print to Neverending.counters.tex the line

```
\providecounter {SouthernOracle} \setcounter {SouthernOracle}{4}
```

If, however, the command \randomizectr is replaced by \norandomizectr, then a third typesetting will leave both the displayed text and the counters file unchanged.

3 Implementation

The counterz package loads the two packages etoolbox and makecmds for the use of conditional tests (boolean and numerical) and the macro \providecounter.

```
⟨*package⟩
\ProvidesPackage{counterz}[%
   2023/06/05 v1.1.1 Additional tools for counters
]%
\RequirePackage{etoolbox,makecmds}
```

3.1 Counter Prefixes

\@counterz@counterprefix
\setcounterprefix
\clearcounterprefix

The default expansion of \@counterz@counterprefix is null, but it can be changed with the commands \setcounterprefix and \clearcounterprefix.

```
\newcommand{\@counterz@counterprefix}{}
\newcommand{\setcounterprefix}[1]{%
   \renewcommand{\@counterz@counterprefix}{#1}
}%
\newcommand{\clearcounterprefix}{%
   \setcounterprefix{}
}%
```

3.2 Manipulating Counters

\xnewcounter
\xprovidecounter
\xsetcounter
\xaddtocounter
\xvalue

These commands are prefix versions of commands `\newcounter`, `\providecounter`, `\setcounter`, `\addtocounter`, and `\value`, respectively. The creation, modification, or use of the counters is carried out on a prefixed version of the specified counter name. When `\@counterz@counterprefix` is null, the commands expand like their standard counterparts.

```
\newcommand{\xnewcounter}[1]{%
   \newcounter{\@counterz@counterprefix #1}
}%
\newcommand{\xprovidecounter}[1]{%
   \providecounter{\@counterz@counterprefix #1}
}%
\newcommand{\xsetcounter}[2]{%
   \setcounter{\@counterz@counterprefix #1}{#2}
}%
\newcommand{\xaddtocounter}[2]{%
   \addtocounter{\@counterz@counterprefix #1}{#2}
}%
\newcommand{\xvalue}[1]{%
   \value{\@counterz@counterprefix #1}
}%
```

3.3 Conditional Statements

The following commands provide if-then-else constructs analogous to those in the package etoolbox. The notable difference is that the arguments are counter names. The command `\xvalue` is used to determine the values of the counters, so that the stored prefix is applied to the specified counter names before execution.

\ifctrequal

`\ifctrequal{`⟨*counter1*⟩`}{`⟨*counter2*⟩`}{`⟨*foo*⟩`}{`⟨*bar*⟩`}` executes ⟨*foo*⟩ if the value of ⟨*counter1*⟩ is equal to the value of ⟨*counter2*⟩ and otherwise executes ⟨*bar*⟩.

```
\newcommand{\ifctrequal}[4]{%
   \ifnumequal{\xvalue{#1}}{\xvalue{#2}}{#3}{#4}
}%
```

\ifctrless

`\ifctrless{`⟨*counter1*⟩`}{`⟨*counter2*⟩`}{`⟨*foo*⟩`}{`⟨*bar*⟩`}` executes ⟨*foo*⟩ if the value of ⟨*counter1*⟩ is less than the value of ⟨*counter2*⟩ and otherwise executes ⟨*bar*⟩.

```
\newcommand{\ifctrless}[4]{%
   \ifnumless{\xvalue{#1}}{\xvalue{#2}}{#3}{#4}
}%
```

\ifctrmore

`\ifctrmore{`⟨*counter1*⟩`}{`⟨*counter2*⟩`}{`⟨*foo*⟩`}{`⟨*bar*⟩`}` executes ⟨*foo*⟩ if the value of ⟨*counter1*⟩ is more than the value of ⟨*counter2*⟩ and otherwise executes ⟨*bar*⟩.

```
\newcommand{\ifctrmore}[4]{%
   \ifnumless{\xvalue{#2}}{\xvalue{#1}}{#3}{#4}
}%
```

\ifctrzero `\ifctrzero{⟨counter⟩}{⟨foo⟩}{⟨bar⟩}` executes ⟨*foo*⟩ if the value of ⟨*counter*⟩ is zero and otherwise executes ⟨*bar*⟩.

```
\newcommand{\ifctrzero}[3]{%
   \ifnumequal{\xvalue{#1}}{0}{#2}{#3}
}%
```

\ifctrneg `\ifctrneg{⟨counter⟩}{⟨foo⟩}{⟨bar⟩}` executes ⟨*foo*⟩ if the value of ⟨*counter*⟩ is negative and otherwise executes ⟨*bar*⟩.

```
\newcommand{\ifctrneg}[3]{%
   \ifnumless{\xvalue{#1}}{0}{#2}{#3}
}%
```

\ifctrpos `\ifctrpos{⟨counter⟩}{⟨foo⟩}{⟨bar⟩}` executes ⟨*foo*⟩ if the value of ⟨*counter*⟩ is positive and otherwise executes ⟨*bar*⟩.

```
\newcommand{\ifctrpos}[3]{%
   \ifnumless{\xvalue{#1}}{1}{#3}{#2}
}%
```

3.4 Displaying Counters

\xarabic
\xroman
\xRoman
\xalph
\xAlph
\xfnsymbol

These commands include prefix versions of the standard display commands.

```
\newcommand{\xarabic}[1]{\arabic{\@counterz@counterprefix #1}}
\newcommand{\xroman}[1]{\roman{\@counterz@counterprefix #1}}
\newcommand{\xRoman}[1]{\Roman{\@counterz@counterprefix #1}}
\newcommand{\xalph}[1]{\alph{\@counterz@counterprefix #1}}
\newcommand{\xAlph}[1]{\Alph{\@counterz@counterprefix #1}}
\newcommand{\xfnsymbol}[1]{\fnsymbol{\@counterz@counterprefix #1}}
```

The following commands likewise apply the stored prefix to the counter name. These commands are designed to aid in the typesetting of counter values within algebraic expressions while observing particular conventions about the display of numbers and their and their signs.

\xabsof `\xabsof{⟨counter⟩}` prints the absolute value of ⟨*counter*⟩.

```
\newcommand{\xabsof}[1]{%
   \ifctrneg{#1}{%
      \the \numexpr 0 - \xvalue{#1} \relax%
   }{%
      \xarabic{#1}%
   }%
}
```

\xsignof `\xsignof{⟨counter⟩}` prints a minus sign "-" if ⟨*counter*⟩ is negative and otherwise prints a plus sign "+". Note that the latter case includes the value zero.

```
\newcommand{\xsignof}[1]{%
   \ifctrneg{#1}{-}{+}
}%
```

\xnegsignof `\xnegsignof{`⟨*counter*⟩`}` prints a plus sign "+" if ⟨*counter*⟩ is negative and otherwise prints a minus sign "-". Note that the latter case includes the value zero.

```
\newcommand{\xnegsignof}[1]{%
   \ifctrneg{#1}{+}{-}
}%
```

\xsigned `\xsigned{`⟨*counter*⟩`}` prints the absolute value of ⟨*counter*⟩, preceded by a plus sign "+" or a minus sign "-" as defined by `\xsignof`.

```
\newcommand{\xsigned}[1]{%
   \xsignof{#1} \xabsof{#1}
}%
```

\xsignednz `\xsignednz{`⟨*counter*⟩`}` is like `\xsigned` but suppresses the number zero.

```
\newcommand{\xsignednz}[1]{%
   \ifctrzero{#1}{}{\xsigned{#1}}
}%
```

\xarabicnz `\xarabicnz{`⟨*counter*⟩`}` is like `\xarabic` but suppresses the number zero.

```
\newcommand{\xarabicnz}[1]{%
   \ifctrzero{#1}{}{\xarabic{#1}}
}%
```

\xnegsigned `\xnegsigned{`⟨*counter*⟩`}` prints the absolute value of ⟨*counter*⟩, preceded by a plus sign "+" or a minus sign "-" as defined by `\xnegsignof`.

```
\newcommand{\xnegsigned}[1]{%
   \xnegsignof{#1} \xabsof{#1}
}%
```

\xnegsignednz `\xnegsignednz{`⟨*counter*⟩`}` is like `\xnegsigned` but suppresses the number zero.

```
\newcommand{\xnegsignednz}[1]{%
   \ifctrzero{#1}{}{\xnegsigned{#1}}
}%
```

\xnegof `\xnegof{`⟨*counter*⟩`}` prints the negative of the value of ⟨*counter*⟩.

```
\newcommand{\xnegof}[1]{%
   \ifctrpos{#1}{-}{}\xabsof{#1}
}%
```

\xnegofnz `\xnegofnz{`⟨*counter*⟩`}` is like `\xnegof` but suppresses the number zero.

```
\newcommand{\xnegofnz}[1]{%
   \ifctrzero{#1}{}{\xnegof{#1}}
}%
```

\xcoef `\xcoef{`⟨*counter*⟩`}` prints the value of ⟨*counter*⟩ except that it suppresses the values of 1, 0, and -1, printing a "-" in the latter case.

```
\newcommand{\xcoef}[1]{%
   \ifboolexpr{%
```

```
      test {\ifnumless{\xvalue{#1}}{-1}}
      or test {\ifnumgreater{\xvalue{#1}}{1}}
   }{%
      \xarabic{#1}
   }{%
   }%
   \ifnumequal{\xvalue{#1}}{-1}{-}{}
}%
```

`\xnegcoef` `\xnegcoef{`⟨*counter*⟩`}` prints the value of ⟨*counter*⟩ except that it suppresses the values of 1, 0, and -1, printing a "-" in the former case.

```
\newcommand{\xnegcoef}[1]{%
   \ifboolexpr{%
      test {\ifnumless{\xvalue{#1}}{-1}}
      or test {\ifnumgreater{\xvalue{#1}}{1}}
   }{%
      \xnegof{#1}
   }{%
   }%
   \ifnumequal{\xvalue{#1}}{1}{-}{}
}%
```

`\xabsofcoef` `\xabsofcoef{`⟨*counter*⟩`}` prints the absolute value of ⟨*counter*⟩ except that it suppresses the values of 1 and 0.

```
\newcommand{\xabsofcoef}[1]{%
   \ifboolexpr{%
      test {\ifnumless{\xvalue{#1}}{-1}}
      or test {\ifnumgreater{\xvalue{#1}}{1}}
   }{%
      \xabsof{#1}
   }{%
   }%
}%
```

`\xsignofcoef` `\xsignofcoef{`⟨*counter*⟩`}` prints the sign of ⟨*counter*⟩ if ⟨*counter*⟩ is nonzero.

```
\newcommand{\xsignofcoef}[1]{%
   \ifctrzero{#1}{}{\xsignof{#1}}
}%
```

`\xnegsignofcoef` `\xnegsignofcoef{`⟨*counter*⟩`}` prints the opposite sign of ⟨*counter*⟩ if ⟨*counter*⟩ is nonzero.

```
\newcommand{\xnegsignofcoef}[1]{%
   \ifctrzero{#1}{}{\xnegsignof{#1}}
}%
```

`\xsignedcoef` `\xsignedcoef{`⟨*counter*⟩`}` is like `\xcoef` except that positive values are preceded by a plus sign "+".

```
\newcommand{\xsignedcoef}[1]{%
```

```
    \xsignofcoef{#1} \xabsofcoef{#1}
}%
```

`\xnegsignedcoef` `\xnegsignedcoef{`⟨*counter*⟩`}` is like `\xsignedcoef` except using the opposite sign.

```
\newcommand{\xnegsignedcoef}[1]{%
    \xnegsignofcoef{#1} \xabsofcoef{#1}
}%
```

3.5 Random Counters

`\randomizectr` `\norandomizectr` In order to assign a random value to a counter during one typesetting and avoid overwriting this value with a random assignment during another typesetting, the boolean *@counterz@random* is used to distinguish between the two typesetting options. The value of *@counterz@random* may be changed by the commands `\randomizectr` and `\norandomizectr`.

```
\newbool{@counterz@random}
\booltrue{@counterz@random}
\newcommand{\randomizectr}{\booltrue{@counterz@random}}
\newcommand{\norandomizectr}{\boolfalse{@counterz@random}}
```

`\ifrandomizectr` `\ifrandomizectr{`⟨*foo*⟩`}{`⟨*bar*⟩`}` executes ⟨*foo*⟩ if the boolean *@counterz@random* is TRUE and otherwise executes ⟨*bar*⟩.

```
\newcommand{\ifrandomizectr}[2]{%
    \ifbool{@counterz@random}{#1}{#2}
}%
```

`\promptrandomizectr` `\promptrandomizectr[`⟨*command*⟩`]{`⟨*message*⟩`}{`⟨*string*⟩`}` writes ⟨*message*⟩ to the terminal and awaits a response from the user at the prompt. The user's response is stored in ⟨*command*⟩ and compared to the text of ⟨*string*⟩. If they are equal, then `\randomizectr` is executed. If they are not equal, then `\norandomizectr` is executed.

```
\newcommand{\promptrandomizectr}[3][\@typein]{%
    \typein[#1]{#2}
    \ifdefstring{#1}{#3}{%
        \randomizectr
    }{%
        \norandomizectr
    }%
}%
```

The commands `\randsetcounter` and `\randaddtocounter` use the pdfTEX primitive `\pdfuniformdeviate` to provide random versions of `\setcounter` and `\addtocounter`. The commands `\xrandsetcounter` and `\xrandaddtocounter` are prefix versions of `\randsetcounter` and `\randaddtocounter`, respectively. Each of these four commands will generate random counter values only when the boolean *@counterz@random* is TRUE.

\randsetcounter
\xrandsetcounter

`\randsetcounter{`⟨*counter*⟩`}{`⟨*min*⟩`}{`⟨*max*⟩`}` assigns to ⟨*counter*⟩ a random integer value between ⟨*min*⟩ and ⟨*max*⟩, if *@counterz@random* is TRUE.

```
\newcommand{\randsetcounter}[3]{%
   \ifrandomizectr{%
      \setcounter{#1}{%
         \the \numexpr #2+\pdfuniformdeviate \numexpr #3-#2+1 \relax
      }%
   }{%
      % Do Nothing
   }%
}%
\newcommand{\xrandsetcounter}[3]{%
   \randsetcounter{\@counterz@counterprefix#1}{#2}{#3}
}%
```

\randaddtocounter
\xrandaddtocounter

`\randaddtocounter{`⟨*counter*⟩`}{`⟨*min*⟩`}{`⟨*max*⟩`}` adds to ⟨*counter*⟩ a random integer value between ⟨*min*⟩ and ⟨*max*⟩, if *@counterz@random* is TRUE.

```
\newcommand{\randaddtocounter}[3]{%
   \ifrandomizectr{%
      \addtocounter{#1}{%
         \the \numexpr #2+\pdfuniformdeviate \numexpr #3-#2+1 \relax
      }%
   }{%
      % Do Nothing
   }%
}%
\newcommand{\xrandaddtocounter}[3]{%
   \randaddtocounter{\@counterz@counterprefix#1}{#2}{#3}
}%
```

The following commands are designed to provide a means by which authors can generate random values for counters but also preserve those values for future typesettings. This is accomplished by storing counters and their values in an external file and then inputting the file before a subsequent typesetting.

\opencountersfile

The command `\opencountersfile` creates and opens the write stream to the file ⟨*jobname*⟩.counters.tex, referenced by the macro `\countersfile`. If the file already exists, it is overwritten. For this reason,

```
\newbool{@counterz@fileISopen}
\boolfalse{@counterz@fileISopen}
\newcommand{\opencountersfile}{%
   \ifbool{@counterz@fileISopen}{%
      \PackageError{counterz}{%
         The write stream is already open!
         \MessageBreak Process interrupted to prevent overwriting
         \MessageBreak \jobname.counters.tex
      }{%
         Be sure to include only one instance of
```

```
      \protect\opencountersfile.
    }%
  }{%
    \ifrandomizectr{%
      \newwrite\countersfile
      \immediate\openout\countersfile=\jobname.counters.tex
      \booltrue{@counterz@fileISopen}
    }{%
      \PackageError{counterz}{%
        \protect\opencountersfile\space requires
        \protect\randomizectr
        \MessageBreak Process interrupted to prevent overwriting
        \MessageBreak \jobname.counters.tex
      }{%
        \protect\opencountersfile\space is designed to open a file
        for saving newly randomized counters. See the Random
        Counters section of the counterz package documentation for
        details.
      }%
    }%
  }%
}
```

`\inputcountersfile` The command `\inputcountersfile` inputs ⟨*jobname*⟩.counters.tex if the file exists and reports a package error if the file does not exist.

```
\newcommand{\inputcountersfile}{%
  \InputIfFileExists{\jobname.counters}{%
  }{%
    \PackageError{counterz}{%
      The file \jobname.counters.tex does not exist.
    }{%
      See the Random Counters section of the counterz package
      documentation.
    }%
  }%
}%
```

`\@counterz@openbrace` `\@counterz@closebrace` The commands `\@counterz@openbrace` and `\@counterz@closebrace` facilitate the writing of the brace delimiters to `\countersfile`.

```
\begingroup
  \catcode`<=1 \catcode`>=2
  \catcode`{=12 \catcode`}=12
  \gdef\@counterz@openbrace<{>
  \gdef\@counterz@closebrace<}>
\endgroup
```

`\savecounter` `\xsavecounter` `\savecounter{`⟨*counter*⟩`}` writes `\providecounter` and `\setcounter` commands to the file ⟨*jobname*⟩.counters.tex so that they may be inputted as part of a future typesetting. The command reports a package error if the write stream to

⟨*jobname*⟩.counters.tex is not open. The command `\xsavecounter` is a prefix version of `\savecounter`.

```
\newcommand{\savecounter}[1]{%
   \ifbool{@counterz@fileISopen}{%
      \immediate\write\countersfile{%
         \unexpanded{\providecounter}
         \@counterz@openbrace#1\@counterz@closebrace
         \unexpanded{\setcounter}
         \@counterz@openbrace#1\@counterz@closebrace
         \@counterz@openbrace\arabic{#1}\@counterz@closebrace
      }%
   }{%
      \PackageError{counterz}{%
         The write stream to the file \jobname.counters.tex must be
         opened before \protect\savecounter\space can be executed.
      }{%
         See \protect\opencountersfile\space and
         \protect\savecounter\space in the counterz package
         documentation.
      }%
   }%
}%

\newcommand{\xsavecounter}[1]{%
   \savecounter{\@counterz@counterprefix#1}%
}%
```

`\randprovidecounter` `\randprovidecounter{`⟨*counter*⟩`}{`⟨*min*⟩`}{`⟨*max*⟩`}` creates ⟨*counter*⟩ if it does not already exist, and if the boolean *@counterz@random* is TRUE then ⟨*counter*⟩ is assigned a random integer value between ⟨*min*⟩ and ⟨*max*⟩ and then saved.

```
\newcommand{\randprovidecounter}[3]{%
   \ifltxcounter{#1}{%
      \@ifnextchar]{%
         \m@k@gobbleendoptarg
      }{%
      }%
   }{%
      \newcounter{#1}
      \ifrandomizectr{%
         \randsetcounter{#1}{#2}{#3}
         \savecounter{#1}
      }{%
      }%
   }%
}%
```

`\xrandprovidecounter` `\xrandprovidecounter{`⟨*counter*⟩`}{`⟨*min*⟩`}{`⟨*max*⟩`}` creates ⟨*counter*⟩ if it does not already exist, and if the boolean *@counterz@random* is TRUE then ⟨*counter*⟩ is assigned a random integer value between ⟨*min*⟩ and ⟨*max*⟩ and then saved.

```
\newcommand{\xrandprovidecounter}[3]{%
   \randprovidecounter{\@counterz@counterprefix#1}{#2}{#3}
}%
```

`\randprovidecounternz` `\randprovidecounternz{`⟨*counter*⟩`}{`⟨*min*⟩`}{`⟨*max*⟩`}` does the same job as the command `\xrandprovidecounter` except that the value of ⟨*counter*⟩ is randomized until it is nonzero.

```
\newcommand{\randprovidecounternz}[3]{%
   \ifltxcounter{#1}{%
      \@ifnextchar]{%
         \m@k@gobbleendoptarg
      }{%
      }%
   }{%
      \newcounter{#1}
      \ifrandomizectr{%
         \setcounter{#1}{0}
         \whileboolexpr{test {\ifnumequal{\value{#1}}{0}}}{%
            \randsetcounter{#1}{#2}{#3}
         }%
         \savecounter{#1}
      }{%
      }%
   }%
}%
```

`\xrandprovidecounternz` `\xrandprovidecounternz{`⟨*counter*⟩`}{`⟨*min*⟩`}{`⟨*max*⟩`}` does the same job as the command `\xrandprovidecounter` except that the value of ⟨*counter*⟩ is randomized until it is nonzero.

```
\newcommand{\xrandprovidecounternz}[3]{%
   \randprovidecounternz{\@counterz@counterprefix#1}{#2}{#3}
}%
```

⟨/package⟩

4 Change History

5 Index

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