The ionumbers package*

Christian Schneider
<software(at)chschneider(dot)eu>

April 14, 2014

Warning: Use with caution and on your own risk! Check output!

Contents

*This document corresponds to ionumbers v0.3.3, dated 2014/04/06.

Abstract

ionumbers stands for 'input/output numbers'.

This package restyles numbers in math mode. If a number in the input file is written, e.g., as `$3,231.44$` as commonly used in English texts, this package is able to restyle it to be output as '3 231,44' as commonly used in German texts (and vice versa). This may be very useful, if you have a large table and want to include it in texts with different output conventions without the need of changing the table.

Furthermore this package can automatically group digits left to the decimal separator (*thousands*) and right to the decimal separator (*thousandths*) without the need of specifing commas (English) or points (German) as separators. E.g., the input `$1234.567890$` can be output as '1 234. 567 890'. By default, thousands/thousandths are grouped in triplets, but the grouping length is configurable, which is useful for numerical data.

Finally, an `e` starts the exponent of the number. For example, `$21e6$` may be output as '26×10^{6}'.

1 Details of number handling

1.1 General rules

Every input *in math mode* consisting of the following characters is treated by this package: `.,+-0123456789`. These characters get macro definitions. A number is any combination of these characters without anything—not even white spaces—in between them. There are two exceptions/special cases:

1. The separator characters `.` and `,` are not treated as part of the number at its end. This avoids problems with comma-separated lists (see below).
2. The sign characters `+` and `-` will only be considered as part of the number, if they appear at the begining of a number.

The lower case letter `e` plays a special role. An `e` immediatly following a number (as defined above) can be configured as begining of the exponential part. The letter `e` will be eaten from the input in this case and substituted by some configurable output. The next number following this `e` in the same group (even with other characters inbetween the `e` and the number) will be treated as exponential part and grouped with curly braces `{}`.

It is a good practice to always add a space before/after each number such that ionumbers knows the beginning/end of a number and does not misinterpret other input as part of it. Below, you will find a couple of examples that might lead to surprising output, if this rule is not followed.

1.2 Caveats

Comma-separated lists of numbers must be input with a space after each comma to prevent `,` to be treated as part of the number. An example is the list `1, 2, 3, \ldots`, where the commas are not part of in the numbers. Note, however, that the commas are treated as part of the numbers in the first two appearances in `1,2,3,\ldots`, as the commas are immediately followed by a digit. Depending on the configuration, this may lead to strange spaces between the numbers, disappearing commas etc.

If you use *indexes consisting of four or more digits* together with automatic grouping of thousands, the grouping will also apply to the indexes. So `a_{1234}` might be output as $a_{1,234}$. The simplest way to prevent undesired automatic grouping is to insert a space after each digit, e. g., as in `a_{1 2 3 4}`.

Please be aware that the first decimal separator of a number marks the begining of the thousandths part of a number; every part of a number appearing left to the first decimal separator is the thousands part. That is why, the input `$1.234.567$` with (only) the package option `autothousandths=true` (`.` is the decimal separator; options will be explained later) will lead to '1.234. 567' in the output. Note the small space after the second point as a result of `234.567` being treated as thousandths part. The thousandths separator—by default a small space—will be output between the third and fourth digit of the thousandths part; the additional point from the input will not be omitted. The input is *syntactically incorrect* (there must not be two decimal separators in one number!) and the output is *not* a bug.

The number following an `e` which has started the exponential part is treated as exponential part, *even if there is arbitrary input inbetween.* Hence, the input `$1e \Pi 2` with package option `exponent=timestento` (will be explained later) leads to a superscript `2` in the output. In some cases, e. g., `$1e \sqrt 2$` or `$1e^2$` with `e` configured as begining of the exponential part, even an error occurs. Again, the input is *syntactically incorrect* and you might want to prevent `e` from being treated as start of the exponential part by adding a space: `$1 e \sqrt 2$` or `$1 e^2$`.

In some rare cases, e. g., `$\sqrt ,$` or `$a^.$`, the usage of point and comma without curly braces `{}` around them will lead to an error. In these cases please add curly braces `{}` around the point or comma. (The ziffer package has the same problem, by the way.)

2 Conflicts with other packages

This package potentially conflicts with any other package that defines a macro for any of the following characters: `.,+-0123456789`

There are tests for these cases and warning or error messages may be output. Please load ionumbers as *last package* to be able to detect as many conflicts as possible. As there is no way to detect conflicts in any case, please report any package known to conflict with ionumbers to the author.

Packages known to conflict with ionumbers are:

ziffer	this package can be replaced by ionumbers except for ziffer's special handling of `--` enabled by `\ZifferStrichAn`
dcolumn	workaround: disable ionumbers for tabulars (e. g., put them inside `\ionumbersoff{`⟨...⟩`}`)
amsmath/amsopn	load ionumbers as last package and disable ionumbers for `\operatorname{`⟨...⟩`}` (e. g., put it inside `\ionumbersoff{`⟨...⟩`}`)

3 Usage

Package options are used to globally configure a default behaviour of ionumbers for the whole document. These options usually consist of a ⟨*key*⟩=⟨*value*⟩ pair. Local changes from this global configuration for arbitrary parts of the document can be applied with special commands.

3.1 Package options concerning the separators in the input

The following options configure the meaning of separators in the LaTeX input file:

`comma`=⟨*value*⟩	comma '`,`' will be treated as ⟨*value*⟩
`point`=⟨*value*⟩	point '`.`' will be treated as ⟨*value*⟩

The following ⟨*value*⟩s can be chosen for both of them:

`ignore`	the separator will be ignored (no output)
`decimal`	decimal separator (separating the thousands from the thousandths part of a number)
`thousands`	thousands separator (used for grouping of thousands part)
`default`	default behaviour of ionumbers (`decimal` for `point`; `thousands` for `comma`)

The separator for exponents is always the lowercase letter `e`. A thousandths separator does not exist in input files; such a separator will only be output, if automatic grouping of the thousandths part is enabled (see below).

3.2 Package options concerning the separators in the output

The previously described options assign a *meaning* to separators in the input file. The *output* of the *meanings* is configured via the following options:

`thousands`=⟨*value*⟩	thousands separator will be output as ⟨*value*⟩
`decimal`=⟨*value*⟩	decimal separator will be output as ⟨*value*⟩
`thousandths`=⟨*value*⟩	thousandths separator will be ⟨*value*⟩
`exponent`=⟨*value*⟩	exponent separator will be output as ⟨*value*⟩

The list of valid ⟨*value*⟩s for `thousands`, `decimal`, and `thousandths` is:

`none`	will be ignored (no output)
`point`	normal point; this is the default point without ionumbers
`comma`	normal comma
`punctpoint`	punctuation point (point followed by small space)
`punctcomma`	punctuation comma (point followed by small space); this is the default comma without ionumbers
`apostrophe`	apostrophe (actually `$^\prime$`; *not* for `decimal`)
`phantom`	space with width of a point (`$\phantom{.}$`; *not* for `decimal`)
`space`	small space (`$\,$`; *not* for `decimal`)
`default`	default behaviour of ionumbers (`punctcomma` for `thousands`; `point` for `decimal`; `space` for `thousandths`)

If a number is handled as exponent, it will be put into curly braces `{}` for correct output of, e. g., signs without spacing around them (mathord). In the following list of valid ⟨*value*⟩s for `exponent` a number immediatly following an `e` will be handled as exponent, unless specified otherwise:

`none`	will be ignored (not output; following number *not* handled as exponent)
`original`	a simple character 'e' (following number *not* handled as exponent)
`ite/itE`	italic lower/upper case letter 'e'
`rme/rmE`	roman lower/upper case letter 'e'
`timestento`	`$\times10\,$` with following number output as superscript
`cdottento`	`$\cdot10\,$` with following number output as superscript
`wedge`	`$^\wedge$`
`default`	default behaviour of ionumbers (`original`)

3.3 Package options concerning automatic grouping

Automatic grouping is a feature that automatically adds the thousands and thousandths separator, respectively. The separator will by default be added after each triplet of digits, but this may be changed (see below). Automatic grouping can be enable or disabled with the following options:

`autothousands`=⟨*value*⟩ automatic grouping of thousands (digits left to decimal separator)

`autothousandths`=⟨*value*⟩ automatic grouping of thousandths (digits right to decimal separator)

The grouping length for the thousands and thousandths, respectively, can be changed be the following options:

`grplenthousands`=⟨*number*⟩ group lengths for thousands (⟨*number*⟩ must be smaller than `10`; defaults to `3`)

`grplenthousandths`=⟨*number*⟩ group lengths for thousandths (⟨*number*⟩ must be smaller than `10`; defaults to `3`)

The available ⟨*value*⟩s are `true` and `false` (default).
Notes on automatic grouping:

1. Grouping of thousandths requires `autothousandths=true` in any case, as there is no thousandths separator for explicitly specifing separations in the input.

2. Automatic grouping of thousands will be skipped in a number, if it contains a thousands separator in the input.

3.4 Local style changes

`\ionumbersstyle` The command `\ionumbersstyle{`⟨*option list*⟩`}` changes the global style definitions as specified as package options for the rest of the group. The ⟨*option list*⟩ may contain any of the package options described in sections 3.1–3.3. An additional

⟨*value*⟩ for all ⟨*key*⟩s is available inside `\ionumbersstyle` to switch back to the configuration specified as package options: `reset`.

`\ionumbersresetstyle` The command `\ionumbersresetstyle` resets all ⟨*value*⟩s to the configuration specified as package options. Actually, it is only a shorthand for `\ionumbersstyle{comma=reset,point=reset,decimal=reset,...}`.

3.5 User-defined values for output separators

A user may specify further output separators. Any user-defined ⟨*value*⟩s for `thousands`, `decimal`, `thousandths`, and `exponent` can be used like the built-in options in section 3.2.

`\newionumbersthousands` `\newionumbersdecimal` `\newionumbersthousandths` `\newionumbersexponent` The command `\newionumbersthousands{`⟨*value*⟩`}{`⟨*definition*⟩`}` has two mandatory arguments. The first one is the name of the newly defined ⟨*value*⟩ for the `thousands` ⟨*key*⟩ and the second one its definition. The commands `\newionumbersdecimal`, `\newionumbersthousandths`, and `\newionumbersexponent` work the same way for the `decimal`, `thousandths`, and `exponent` ⟨*key*⟩, respectively. There is a starred version of `\newionumbersexponent` (called `\newionumbersexponent*`) that typesets the following number as superscript.

`\renewionumbersthousands` `\renewionumbersdecimal` `\renewionumbersthousandths` `\renewionumbersexponent` To redefine an existing ⟨*key*⟩ definition there are `\renew...` versions of the previously described commands.

Notes on definitions:

1. All ⟨*definition*⟩s are set inside `\ionumbersoff` (see section 3.6). This means that numbers appearing in the ⟨*definition*⟩s are not treated by this package.
2. The value `curr` has an internal meaning and should *not* be defined/redefined by the user.

3.6 Enabling and disabling features

`\ionumbers` The command `\ionumbers` makes comma, point, signs, and digits active in math mode. This is equivalent to enabling the features of this package. This command applies to the end of the current group.

`\endionumbers` To disable the features by making comma, point, signs, and digits inactive again the command `\endionumbers` can be used. This command applies to the end of the current group.

`\ionumbersoff` The command `\ionumbersoff{`⟨*stuff*⟩`}` disables the features only for ⟨*stuff*⟩.

4 License

5 Acknowledgements

The idea and parts of this package are based on ziffer.sty v2.1 by Martin Väth <vaeth@mathematik.uni-wuerzburg.de>.

Furthermore the \l@addto@macro (with changed name) from koma-script bundle v2.9t by Markus Kohm and Frank Neukam is used in this package.

Thanks to Martin Väth and Markus Kohm for permitting to use their code in this package.

6 Bugs, problems, and suggestions

Please report bugs and problems or send suggestions for this package to Christian Schneider. Check for updates before reporting bugs at the website mentioned above. Do *not* bother Martin Väth, Markus Kohm, or Frank Neukam with bugs, problems or suggestions concerning this package!

7 Implementation

The implementation is briefly described in this section. First of all, we need the keyval package for ⟨*key*⟩=⟨*value*⟩ options:

```
\RequirePackage{keyval}
```

7.1 Default/global configuration

In principle the definitions of all available ⟨*key'*⟩=⟨*value'*⟩ pairs is contained in the internal macros \ion@⟨*key'*⟩@⟨*value'*⟩. Setting a package option ⟨*key*⟩=⟨*value*⟩ defines \ion@⟨*key*⟩@reset to be \ion@⟨*key*⟩@⟨*value*⟩.

The following ifs will be required to remember, if automatic grouping is enabled. The counts will be required for the grouping lengths.

```
\newif\ifion@autothousands
\newif\ifion@autothousandths
\newcount\ion@grplenthousands
\newcount\ion@grplenthousandths
```

The next macro will be used for syntax checks of numerical arguments.

```
\newcommand*{\ion@grplencheck}[1]{%
  \ifnum#1>9%
    \PackageError{ionumbers}%
    {Group length argument too large (#1).\MessageBreak%
    Grouping lengths must be smaller than 10.}{}%
  \fi%
}
```

These shorthands are used to define the ⟨*key*⟩s for package options and set their ⟨*value*⟩s using keyval, respectively.

```
\newcommand*\ion@defpackopts{\define@key{ion@packopts}}
\newcommand*\ion@setpackopts{\setkeys{ion@packopts}}
```

Next the ⟨*key*⟩s are defined.

```
\ion@defpackopts{comma}{%
  \def\ion@comma@reset{\csname ion@comma@#1\endcsname}%
  \def\ion@aftercomma@reset{\csname ion@aftercomma@#1\endcsname}}
\ion@defpackopts{point}{%
  \def\ion@point@reset{\csname ion@point@#1\endcsname}%
  \def\ion@afterpoint@reset{\csname ion@afterpoint@#1\endcsname}}
\ion@defpackopts{decimal}{\def\ion@decimal@reset{%
  \csname ion@decimal@#1\endcsname}}
\ion@defpackopts{thousands}{\def\ion@thousands@reset{%
  \csname ion@thousands@#1\endcsname}}
\ion@defpackopts{thousandths}{\def\ion@thousandths@reset{%
  \csname ion@thousandths@#1\endcsname}}
\ion@defpackopts{exponent}{\def\ion@exponent@reset{%
  \csname ion@exponent@#1\endcsname}}
\ion@defpackopts{autothousands}[true]{\def\ion@autothousandsreset{%
  \csname ion@autothousands#1\endcsname}\ion@autothousandsreset}
\ion@defpackopts{autothousandths}[true]{\def\ion@autothousandthsreset{%
  \csname ion@autothousandths#1\endcsname}\ion@autothousandthsreset}
\ion@defpackopts{grplenthousands}{\ion@grplencheck{#1}%
  \def\ion@grplenthousandsreset{\ion@grplenthousands=#1}%
  \ion@grplenthousandsreset}
\ion@defpackopts{grplenthousandths}{\ion@grplencheck{#1}%
  \def\ion@grplenthousandthsreset{\ion@grplenthousandths=#1}%
  \ion@grplenthousandthsreset}
```

Finally, the default ⟨*value*⟩s are set and—if specified by the user as package option—overwritten with the user's configuration.

```
\ion@setpackopts{comma=default,point=default,thousands=default,%
  decimal=default,thousandths=default,exponent=default,autothousands=false,%
  autothousandths=false,grplenthousands=3,grplenthousandths=3}
\DeclareOption*{\expandafter\ion@setpackopts\expandafter{\CurrentOption}}
\ProcessOptions\relax
```

7.2 Local style changes

The currently active configuration of a ⟨*key*⟩ is stored in the macro `\ion@`⟨*key*⟩`@curr`. The `\ion@`⟨*key*⟩`@curr` macros for all ⟨*key*⟩s are defined using the mechanism for local configuration changes.

The local options are defined and set—analogous to the package option case—with two shorthands using keyval. The latter is publically available to the user.

```
\newcommand*\ion@deflocopts{\define@key{ion@locopts}}
```

`\ionumberstyle`

```
\newcommand*\ionumbersstyle[1]{\setkeys{ion@locopts}{#1}}
```

Now the ⟨*key*⟩s for the local options are defined (just as in the case of the package options):

```
\ion@deflocopts{comma}{%
  \def\ion@comma@curr{\csname ion@comma@#1\endcsname}%
  \def\ion@aftercomma@curr{\csname ion@aftercomma@#1\endcsname}}
\ion@deflocopts{point}{%
```

```
  \def\ion@point@curr{\csname ion@point@#1\endcsname}%
  \def\ion@afterpoint@curr{\csname ion@afterpoint@#1\endcsname}}
\ion@deflocopts{decimal}{\def\ion@decimal@curr{%
  \csname ion@decimal@#1\endcsname}}
\ion@deflocopts{thousands}{\def\ion@thousands@curr{%
  \csname ion@thousands@#1\endcsname}}
\ion@deflocopts{thousandths}{\def\ion@thousandths@curr{%
  \csname ion@thousandths@#1\endcsname}}
\ion@deflocopts{exponent}{\def\ion@exponent@curr{%
  \csname ion@exponent@#1\endcsname}}
\ion@deflocopts{autothousands}[true]{\csname ion@autothousands#1\endcsname}
\ion@deflocopts{autothousandths}[true]{\csname ion@autothousandths#1\endcsname}
\ion@deflocopts{grplenthousands}{%
  \def\@tempa{#1}%
  \def\@tempb{reset}%
  \ifx\@tempa\@tempb%
    \ion@grplenthousandsreset%
  \else%
    \ion@grplencheck{#1}%
    \ion@grplenthousands=#1%
  \fi%
}
\ion@deflocopts{grplenthousandths}{%
  \def\@tempa{#1}%
  \def\@tempb{reset}%
  \ifx\@tempa\@tempb%
    \ion@grplenthousandthsreset%
  \else%
    \ion@grplencheck{#1}%
    \ion@grplenthousandths=#1%
  \fi%
}
```

Finally, the command for resetting all ⟨*key*⟩s is defined.

`\ionumbersresetstyle`

```
\newcommand*\ionumbersresetstyle{%
  \ionumbersstyle{comma=reset,point=reset,thousands=reset,%
    decimal=reset,thousandths=reset,exponent=reset,autothousands=reset,%
    autothousandths=reset,grplenthousands=reset,grplenthousandths=reset}}
```

This command is issued at the end of the package to make the configuration of the package options active (and have no undefined `\ion@`⟨*key*⟩`@curr` macros).

```
\AtEndOfPackage{\ionumbersresetstyle}
```

7.3 User-defined values for output separators

The commands for user-defined ⟨*value*⟩s for output separators just (re)define the internal macro `\ion@`⟨*key*⟩`@`⟨*value*⟩ storing the definition for the ⟨*key*⟩=⟨*value*⟩ pair.

`\newionumbersthousands`

```
\newcommand*\newionumbersthousands[2]{\expandafter\newcommand%
```

```
88    \expandafter*\csname ion@thousands@#1\endcsname{\ionumbersoff{#2}}}
```

\newionumbersdecimal

```
89 \newcommand*\newionumbersdecimal[2]{\expandafter\newcommand%
90    \expandafter*\csname ion@decimal@#1\endcsname{\ionumbersoff{#2}}}
```

\newionumbersthousandths

```
91 \newcommand*\newionumbersthousandths[2]{\expandafter\newcommand%
92    \expandafter*\csname ion@thousandths@#1\endcsname{\ionumbersoff{#2}}}
```

\newionumbersexponent

```
93 \newcommand*\newionumbersexponent{%
94    \@ifstar{\newionumbersexponent@@}{\newionumbersexponent@}}
95 \newcommand*\newionumbersexponent@[2]{\expandafter\newcommand%
96    \expandafter*\csname ion@exponent@#1\endcsname{\ionumbersoff{#2}}}
97 \newcommand*\newionumbersexponent@@[2]{\expandafter\newcommand%
98    \expandafter*\csname ion@exponent@#1\endcsname{\ionumbersoff{#2}%
99    \ion@exponent@superscripttrue}}
```

\renewionumbersthousands

```
100 \newcommand*\renewionumbersthousands[2]{\expandafter\renewcommand%
101    \expandafter*\csname ion@thousands@#1\endcsname{\ionumbersoff{#2}}}
```

\renewionumbersdecimal

```
102 \newcommand*\renewionumbersdecimal[2]{\expandafter\renewcommand%
103    \expandafter*\csname ion@decimal@#1\endcsname{\ionumbersoff{#2}}}
```

\renewionumbersthousandths

```
104 \newcommand*\renewionumbersthousandths[2]{\expandafter\renewcommand%
105    \expandafter*\csname ion@thousandths@#1\endcsname{\ionumbersoff{#2}}}
```

\renewionumbersexponent

```
106 \newcommand*\renewionumbersexponent{%
107    \@ifstar{\renewionumbersexponent@@}{\renewionumbersexponent@}}
108 \newcommand*\renewionumbersexponent@[2]{\expandafter\renewcommand%
109    \expandafter*\csname ion@exponent@#1\endcsname{\ionumbersoff{#2}%
110    \ion@currnum@exponent}}
111 \newcommand*\renewionumbersexponent@@[2]{\expandafter\renewcommand%
112    \expandafter*\csname ion@exponent@#1\endcsname{\ionumbersoff{#2}%
113    \ion@currnum@exponent\ion@exponent@superscripttrue}}
```

7.4 Internal macros holding definitions for ⟨*key*⟩=⟨*value*⟩ pairs

First of all, macros with the original character definitions are defined.

```
114 \AtBeginDocument{
115    \mathchardef\ion@point@original=\the\mathcode`.
116    \mathchardef\ion@comma@original=\the\mathcode`,
117    \mathchardef\ion@plus@original=\the\mathcode`+
118    \mathchardef\ion@minus@original=\the\mathcode`-
119    \mathchardef\ion@zero@original=\the\mathcode`0
```

```
  \mathchardef\ion@one@original=\the\mathcode`1
  \mathchardef\ion@two@original=\the\mathcode`2
  \mathchardef\ion@three@original=\the\mathcode`3
  \mathchardef\ion@four@original=\the\mathcode`4
  \mathchardef\ion@five@original=\the\mathcode`5
  \mathchardef\ion@six@original=\the\mathcode`6
  \mathchardef\ion@seven@original=\the\mathcode`7
  \mathchardef\ion@eight@original=\the\mathcode`8
  \mathchardef\ion@nine@original=\the\mathcode`9
  \mathchardef\ion@e@original=\the\mathcode`e
  \mathchardef\ion@E@original=\the\mathcode`E
}
```

Here the `\ion@`⟨*key*⟩`@`⟨*value*⟩ macros are defined, begining with the definitions for the comma as input separator.

```
\def\ion@comma@ignore{}
\def\ion@comma@decimal{\ion@decimal@curr}
\def\ion@comma@thousands{\ion@thousands@curr}
\def\ion@comma@default{\ion@comma@thousands}
```

The macros `\ion@comma@`⟨*value*⟩ contain the output for a comma appearing in the input. Actually, a second set of `\ion@`aftercomma`@`⟨*value*⟩ macros is required containing commands to be issued whenever a comma appears. If comma is the decimal separator, the appearance of comma in the input will mean that input of the thousands part is complete and the thousandths thousandths part starts (`\ion@beforedecimalfalse` must be issued). If comma is the thousands separator, the automatic grouping of thousands will be switched of for that number (`\ion@noexplicitthousandsfalse` must be issued).

```
\def\ion@aftercomma@ignore{}
\def\ion@aftercomma@decimal{\ion@beforedecimalfalse}
\def\ion@aftercomma@thousands{\ion@noexplicitthousandsfalse}
\def\ion@aftercomma@default{\ion@aftercomma@thousands}
```

An analogous set of macros is defined for the point as input separator.

```
\def\ion@point@ignore{}
\def\ion@point@decimal{\ion@decimal@curr}
\def\ion@point@thousands{\ion@thousands@curr}
\def\ion@point@default{\ion@point@decimal}
```

For the same reasons as mentioned before a set of `\ion@`afterpoint`@`⟨*value*⟩ macros is required.

```
\def\ion@afterpoint@ignore{}
\def\ion@afterpoint@decimal{\ion@beforedecimalfalse}
\def\ion@afterpoint@thousands{\ion@noexplicitthousandsfalse}
\def\ion@afterpoint@default{\ion@afterpoint@decimal}
```

Next the definitions for the decimal output separator, ...

```
\def\ion@decimal@point{\mathord{\ion@point@original}}
\def\ion@decimal@comma{\mathord{\ion@comma@original}}
\def\ion@decimal@punctpoint{\mathpunct{\ion@decimal@point}}
\def\ion@decimal@punctcomma{\mathpunct{\ion@decimal@comma}}
\def\ion@decimal@default{\ion@decimal@point}
```

... the thousands output separator, ...

```
\def\ion@thousands@none{}
\def\ion@thousands@point{\mathord{\ion@point@original}}
```

```
\def\ion@thousands@comma{\mathord{\ion@comma@original}}
\def\ion@thousands@punctpoint{\mathpunct{\ion@decimal@point}}
\def\ion@thousands@punctcomma{\mathpunct{\ion@decimal@comma}}
\def\ion@thousands@apostrophe{^\prime}
\def\ion@thousands@phantom{\phantom{\ion@point@original}}
\def\ion@thousands@space{\,}
\def\ion@thousands@default{\ion@thousands@punctcomma}
```

... the thousandths output separator, ...

```
\def\ion@thousandths@none{}
\def\ion@thousandths@point{\mathord{\ion@point@original}}
\def\ion@thousandths@comma{\mathord{\ion@comma@original}}
\def\ion@thousandths@punctpoint{\mathpunct{\ion@decimal@point}}
\def\ion@thousandths@punctcomma{\mathpunct{\ion@decimal@comma}}
\def\ion@thousandths@apostrophe{^\prime}
\def\ion@thousandths@phantom{\phantom{\ion@point@original}}
\def\ion@thousandths@space{\,}
\def\ion@thousandths@default{\ion@thousandths@space}
```

... and the exponent output separator are given.

```
\def\ion@exponent@none{}
\def\ion@exponent@original{\ion@e@original}
\def\ion@exponent@ite{\mathit{\ion@e@original}\ion@currnum@exponenttrue}
\def\ion@exponent@itE{\mathit{\ion@E@original}\ion@currnum@exponenttrue}
\def\ion@exponent@rme{\mathrm{\ion@e@original}\ion@currnum@exponenttrue}
\def\ion@exponent@rmE{\mathrm{\ion@E@original}\ion@currnum@exponenttrue}
\def\ion@exponent@timestento{\times10\,\ion@currnum@exponenttrue%
  \ion@exponent@superscripttrue}
\def\ion@exponent@cdottento{\cdot10\,\ion@currnum@exponenttrue%
  \ion@exponent@superscripttrue}
\def\ion@exponent@wedge{^\wedge\ion@currnum@exponenttrue}
\def\ion@exponent@default{\ion@exponent@original}
```

7.5 Enabling and disabling features

The following helper macros make different subsets of `.,+-0123456789` active.

```
\def\ion@separators@active{\catcode`\,=\active\catcode`\.=\active\relax}
\def\ion@signs@active{\catcode`\+=\active\catcode`\-=\active\relax}
\def\ion@digits@active{\catcode`\,=\active\catcode`\.=\active%
  \catcode`\0=\active\catcode`\1=\active\catcode`\2=\active%
  \catcode`\3=\active\catcode`\4=\active\catcode`\5=\active%
  \catcode`\6=\active\catcode`\7=\active\catcode`\8=\active%
  \catcode`\9=\active\relax}
```

An analogous set of macros makes subsets of these characters active/inactive in math mode.

```
\def\ion@separators@math@active{\mathcode`,="8000\mathcode`.="8000\relax}
\def\ion@signs@math@active{\mathcode`+="8000\mathcode`-="8000\relax}
\def\ion@digits@math@active{\mathcode`0="8000\mathcode`1="8000\mathcode`2="8000%
  \mathcode`3="8000\mathcode`4="8000\mathcode`5="8000\mathcode`6="8000%
  \mathcode`7="8000\mathcode`8="8000\mathcode`9="8000\relax}
\def\ion@separators@math@inactive{%
  \mathcode`,=\the\ion@comma@original%
```

```
  \mathcode`.=\the\ion@point@original%
  \relax}
\def\ion@signs@math@inactive{%
  \mathcode`+=\the\ion@plus@original%
  \mathcode`-=\the\ion@minus@original%
  \relax}
\def\ion@digits@math@inactive{%
  \mathcode`0=\the\ion@zero@original%
  \mathcode`1=\the\ion@one@original%
  \mathcode`2=\the\ion@two@original%
  \mathcode`3=\the\ion@three@original%
  \mathcode`4=\the\ion@four@original%
  \mathcode`5=\the\ion@five@original%
  \mathcode`6=\the\ion@six@original%
  \mathcode`7=\the\ion@seven@original%
  \mathcode`8=\the\ion@eight@original%
  \mathcode`9=\the\ion@nine@original%
  \relax}
```

Next the user interface for making `.,+-0123456789` active/inactive follows.

`\ionumbers`

```
\def\ionumbers{\ion@separators@math@active\ion@signs@math@active%
  \ion@digits@math@active}
```

`\endionumbers`

```
\def\endionumbers{\ion@separators@math@inactive\ion@signs@math@inactive%
  \ion@digits@math@inactive}
```

`\ionumbersoff`

```
\newcommand\ionumbersoff[1]{\begingroup\endionumbers#1\ionumbers\endgroup}
```

Of course, at the begining of the document the charactars shall be active by default.

```
\AtBeginDocument{\ionumbers}
```

7.6 Definitions of active characters

The macro definitions for the characters `.,+-0123456789` are hold in the following macros. Number processing works by looking at the next character and performing one or more from the following actions:

- the currently configured output for the character will be added to the end of `\ion@currnum` by `\ion@currnum@append`; `\ion@currnum` stores the currently processed number
- only for comma/point: the corresponding `after...` macro will be issued
- the currently processed number will be output via `\ion@currnum@output`
- the `e` will be eaten and replaced by its configured output

The conditions in the macro definitions should be self-explanatory for each character. The extra `\ion@startnumber` is required to avoid problems with input like `$a_0$` or `$\sqrt 2$`, where curly braces around `0` and `2` have been omitted.

```
\def\ion@comma{%
  \ion@ifnextdigit{%
    \ion@currnum@append*{\ion@comma@curr}\ion@aftercomma@curr%
  }{%
    \ion@ifnextseparator{%
      \ion@currnum@append*{\ion@comma@curr}\ion@aftercomma@curr%
        \@warning{Too many separators}%
    }{%
      \ion@ifnextchar e{%
        \ion@currnum@append*{\ion@comma@curr}\ion@aftercomma@curr%
          \ion@currnum@output\ion@exponent@curr\@gobble%
      }{%
        \ion@currnum@output\ion@comma@original%
      }%
    }%
  }%
}
\def\ion@point{%
  \ion@ifnextdigit{%
    \ion@currnum@append*{\ion@point@curr}\ion@afterpoint@curr%
  }{%
    \ion@ifnextseparator{%
      \ion@currnum@append*{\ion@point@curr}\ion@afterpoint@curr%
        \@warning{Too many separators}%
    }{%
      \ion@ifnextchar e{%
        \ion@currnum@append*{\ion@point@curr}\ion@afterpoint@curr%
          \ion@currnum@output\ion@exponent@curr\@gobble%
      }{%
        \ion@currnum@output\ion@point@original%
      }%
    }%
  }%
}
\def\ion@plus{%
  \ion@iffirstchar{%
    \ion@plus@original%
  }{%
    \ion@currnum@append*{\ion@plus@original}%
  }%
  \ion@ifnextdigit{%
    %% nothing
  }{%
    \ion@ifnextseparator{%
      %% nothing
    }{%
      \ion@ifnextsign{%
        \@warning{Too many signs}%
      }{%
        \ion@currnum@output%
      }%
```

```
    }%
  }%
}
\def\ion@minus{%
  \ion@iffirstchar{%
    \ion@minus@original%
  }{%
    \ion@currnum@append*{\ion@minus@original}%
  }%
  \ion@ifnextdigit{%
    %% nothing
  }{%
    \ion@ifnextseparator{%
      %% nothing
    }{%
      \ion@ifnextsign{%
        \@warning{Too many signs}%
      }{%
        \ion@currnum@output%
      }%
    }%
  }%
}
\def\ion@zero{%
  \ion@iffirstchar{%
    \ion@zero@original\ion@currnum@append{}%
  }{%
    \ion@currnum@append{\ion@zero@original}%
  }%
  \ion@ifnextdigit{%
    %% nothing
  }{%
    \ion@ifnextseparator{%
      %% nothing
    }{%
      \ion@ifnextchar e{%
        \ion@currnum@output\ion@exponent@curr\@gobble%
      }{%
        \ion@currnum@output%
      }%
    }%
  }%
}
\def\ion@one{%
  \ion@iffirstchar{%
    \ion@one@original\ion@currnum@append{}%
  }{%
    \ion@currnum@append{\ion@one@original}%
  }%
  \ion@ifnextdigit{%
    %% nothing
  }{%
    \ion@ifnextseparator{%
      %% nothing
```

```
    }{%
      \ion@ifnextchar e{%
        \ion@currnum@output\ion@exponent@curr\@gobble%
      }{%
        \ion@currnum@output%
      }%
    }%
  }%
}
\def\ion@two{%
  \ion@iffirstchar{%
    \ion@two@original\ion@currnum@append{}%
  }{%
    \ion@currnum@append{\ion@two@original}%
  }%
  \ion@ifnextdigit{%
    %% nothing
  }{%
    \ion@ifnextseparator{%
      %% nothing
    }{%
      \ion@ifnextchar e{%
        \ion@currnum@output\ion@exponent@curr\@gobble%
      }{%
        \ion@currnum@output%
      }%
    }%
  }%
}
\def\ion@three{%
  \ion@iffirstchar{%
    \ion@three@original\ion@currnum@append{}%
  }{%
    \ion@currnum@append{\ion@three@original}%
  }%
  \ion@ifnextdigit{%
    %% nothing
  }{%
    \ion@ifnextseparator{%
      %% nothing
    }{%
      \ion@ifnextchar e{%
        \ion@currnum@output\ion@exponent@curr\@gobble%
      }{%
        \ion@currnum@output%
      }%
    }%
  }%
}
\def\ion@four{%
  \ion@iffirstchar{%
    \ion@four@original\ion@currnum@append{}%
  }{%
    \ion@currnum@append{\ion@four@original}%
```

```
  }%
  \ion@ifnextdigit{%
    %% nothing
  }{%
    \ion@ifnextseparator{%
      %% nothing
    }{%
      \ion@ifnextchar e{%
        \ion@currnum@output\ion@exponent@curr\@gobble%
      }{%
        \ion@currnum@output%
      }%
    }%
  }%
}
\def\ion@five{%
  \ion@iffirstchar{%
    \ion@five@original\ion@currnum@append{}%
  }{%
    \ion@currnum@append{\ion@five@original}%
  }%
  \ion@ifnextdigit{%
    %% nothing
  }{%
    \ion@ifnextseparator{%
      %% nothing
    }{%
      \ion@ifnextchar e{%
        \ion@currnum@output\ion@exponent@curr\@gobble%
      }{%
        \ion@currnum@output%
      }%
    }%
  }%
}
\def\ion@six{%
  \ion@iffirstchar{%
    \ion@six@original\ion@currnum@append{}%
  }{%
    \ion@currnum@append{\ion@six@original}%
  }%
  \ion@ifnextdigit{%
    %% nothing
  }{%
    \ion@ifnextseparator{%
      %% nothing
    }{%
      \ion@ifnextchar e{%
        \ion@currnum@output\ion@exponent@curr\@gobble%
      }{%
        \ion@currnum@output%
      }%
    }%
  }%
```

```
}
\def\ion@seven{%
  \ion@iffirstchar{%
    \ion@seven@original\ion@currnum@append{}%
  }{%
    \ion@currnum@append{\ion@seven@original}%
  }%
  \ion@ifnextdigit{%
    %% nothing
  }{%
    \ion@ifnextseparator{%
      %% nothing
    }{%
      \ion@ifnextchar e{%
        \ion@currnum@output\ion@exponent@curr\@gobble%
      }{%
        \ion@currnum@output%
      }%
    }%
  }%
}
\def\ion@eight{%
  \ion@iffirstchar{%
    \ion@eight@original\ion@currnum@append{}%
  }{%
    \ion@currnum@append{\ion@eight@original}%
  }%
  \ion@ifnextdigit{%
    %% nothing
  }{%
    \ion@ifnextseparator{%
      %% nothing
    }{%
      \ion@ifnextchar e{%
        \ion@currnum@output\ion@exponent@curr\@gobble%
      }{%
        \ion@currnum@output%
      }%
    }%
  }%
}
\def\ion@nine{%
  \ion@iffirstchar{%
    \ion@nine@original\ion@currnum@append{}%
  }{%
    \ion@currnum@append{\ion@nine@original}%
  }%
  \ion@ifnextdigit{%
    %% nothing
  }{%
    \ion@ifnextseparator{%
      %% nothing
    }{%
      \ion@ifnextchar e{%
```

```
        \ion@currnum@output\ion@exponent@curr\@gobble%
      }{%
        \ion@currnum@output%
      }%
    }%
  }%
}
```

The macro `\ion@define@charmacros` is used to assign the above macros to the (active) characters `.,+-0123456789`. It will be executed later in the conflict test section.

```
\begingroup
  \ion@separators@active\ion@signs@active\ion@digits@active
  \gdef\ion@define@charmacros{%
    \global\let,=\ion@comma%
    \global\let.=\ion@point%
    \global\let+=\ion@plus%
    \global\let-=\ion@minus%
    \global\let0=\ion@zero%
    \global\let1=\ion@one%
    \global\let2=\ion@two%
    \global\let3=\ion@three%
    \global\let4=\ion@four%
    \global\let5=\ion@five%
    \global\let6=\ion@six%
    \global\let7=\ion@seven%
    \global\let8=\ion@eight%
    \global\let9=\ion@nine%
  }
\endgroup
```

If one of `+-0123456789` is the first character of a number and this number not part of an exponent, then argument '1' will be used; otherwise argument '2' will be used. This macro is required to handle single characters not grouped in curly braces `{}` in expressions like `$a^0$` or `$\sqrt 2$` correctly.

```
\def\ion@iffirstchar#1#2{%
  \ifion@currnum@exponent%
    #2%
  \else%
    \ifion@currnum@firstchar%
      #1%
    \else
      #2%
    \fi%
  \fi%
  \ion@currnum@firstcharfalse%
}
```

Now the macros for the conditions in the above definitions follow. There are tests for a digit `0123456789`, ...

```
\long\def\ion@ifnextdigit#1#2{%
  \def\reserved@a{#1}%
  \def\reserved@b{#2}%
  \futurelet\@let@token\ion@ifnextdigit@}
\def\ion@ifnextdigit@{%
```

```
  \ifx\@let@token1\let\reserved@c\reserved@a\else%
    \ifx\@let@token2\let\reserved@c\reserved@a\else%
      \ifx\@let@token3\let\reserved@c\reserved@a\else%
        \ifx\@let@token4\let\reserved@c\reserved@a\else%
          \ifx\@let@token5\let\reserved@c\reserved@a\else%
            \ifx\@let@token6\let\reserved@c\reserved@a\else%
              \ifx\@let@token7\let\reserved@c\reserved@a\else%
                \ifx\@let@token8\let\reserved@c\reserved@a\else%
                  \ifx\@let@token9\let\reserved@c\reserved@a\else%
                    \ifx\@let@token0\let\reserved@c\reserved@a\else%
                      \let\reserved@c\reserved@b%
                    \fi%
                  \fi%
                \fi%
              \fi%
            \fi%
          \fi%
        \fi%
      \fi%
    \fi%
  \fi%
  \reserved@c}
```

... for a separator `.`, `,` ...

```
\long\def\ion@ifnextseparator#1#2{%
  \def\reserved@a{#1}%
  \def\reserved@b{#2}%
  \futurelet\@let@token\ion@ifnextseparator@}
\def\ion@ifnextseparator@{%
  \ifx\@let@token,\let\reserved@c\reserved@a\else%
    \ifx\@let@token.\let\reserved@c\reserved@a\else%
      \let\reserved@c\reserved@b%
    \fi%
  \fi%
  \reserved@c}
```

... and for a sign `+-` as next character.

```
\long\def\ion@ifnextsign#1#2{%
  \def\reserved@a{#1}%
  \def\reserved@b{#2}%
  \futurelet\@let@token\ion@ifnextsign@}
\def\ion@ifnextsign@{%
  \ifx\@let@token+\let\reserved@c\reserved@a\else%
    \ifx\@let@token-\let\reserved@c\reserved@a\else%
      \let\reserved@c\reserved@b%
    \fi%
  \fi%
  \reserved@c}
```

An additional test for an arbitrary character is also added. It obeys white spaces in contrast to LaTeX's `\@ifnextchar`.

```
\long\def\ion@ifnextchar#1#2#3{%
  \let\reserved@d=#1%
  \def\reserved@a{#2}%
  \def\reserved@b{#3}%
```

```
  \futurelet\@let@token\ion@ifnextchar@}
\def\ion@ifnextchar@{%
  \ifx\@let@token\reserved@d%
    \let\reserved@c\reserved@a%
  \else%
    \let\reserved@c\reserved@b%
  \fi%
  \reserved@c}
```

7.7 Test for conflicts with other packages

First of all we test for some packages known to conflict with ionumbers. This will be done by checking at the begining of the document, if one of these packages has been loaded and an error/warning will be issued.

```
\newcommand*{\ion@conflict@package}[1]{%
  \@ifpackageloaded{#1}{%
    \PackageError{ionumbers}%
    {Packages #1 and ionumbers conflict!\MessageBreak%
    Do not load both packages in the same document}{}%
  }{}%
}
\newcommand*{\ion@problem@package}[2]{%
  \@ifpackageloaded{#1}{%
    \PackageWarning{ionumbers}%
    {Loading #1 and ionumbers is problematic!\MessageBreak#2}%
  }{}%
}

\AtBeginDocument{%
  \ion@conflict@package{ziffer}%
  \ion@problem@package{dcolumn}{Use `tabular's inside \string\ionumbersoff}%
  \ion@problem@package{amsmath}{Load ionumbers after amsmath}%
  \ion@problem@package{amsmath}{Use \string\operatorname\space inside
    \string\ionumbersoff}%
  \ion@problem@package{amsopn}{Use \string\operatorname\space inside
    \string\ionumbersoff}%
}
```

Next the characters `.,+-0123456789` are checked for macro definitions (by other packages). This way conflicts with other packages may be detected with some probability (but only if the conflicting package has already been loaded).

```
\newcommand*\ion@conflict@definedtest[1]{%
  \ifx#1\@undefined\else\PackageWarning{ionumbers}%
    {Potential conflict with other package(s) detected.\MessageBreak%
    `\string#1' has already been defined. I will redefine it.\MessageBreak%
    This might break other package(s)!\MessageBreak}\fi}
\begingroup
  \ion@separators@active\ion@signs@active\ion@digits@active
  \ion@conflict@definedtest{,}
  \ion@conflict@definedtest{.}
  \ion@conflict@definedtest{+}
  \ion@conflict@definedtest{-}
```

```
  \ion@conflict@definedtest{0}
  \ion@conflict@definedtest{1}
  \ion@conflict@definedtest{2}
  \ion@conflict@definedtest{3}
  \ion@conflict@definedtest{4}
  \ion@conflict@definedtest{5}
  \ion@conflict@definedtest{6}
  \ion@conflict@definedtest{7}
  \ion@conflict@definedtest{8}
  \ion@conflict@definedtest{9}
\endgroup
```

After the above test the definitions of the characters of ionumbers can be applied.

```
\ion@define@charmacros
```

Additionally, ionumbers tests for redefinitions of the macros of the characters at the begining of the document.

```
\newcommand*{\ion@conflict@redefinedtest}[2]{%
  \ifx#1#2\else\PackageWarning{ionumbers}%
    {Potential conflict with other package(s) detected.\MessageBreak%
    `\string#1' has been redefined. This might break ionumbers!\MessageBreak}%
  \fi}
\begingroup
  \ion@separators@active\ion@signs@active\ion@digits@active
  \gdef\ion@conflict@redefinedtest@macro{%
    \ion@conflict@redefinedtest{,}{\ion@comma}%
    \ion@conflict@redefinedtest{.}{\ion@point}%
    \ion@conflict@redefinedtest{+}{\ion@plus}%
    \ion@conflict@redefinedtest{-}{\ion@minus}%
    \ion@conflict@redefinedtest{0}{\ion@zero}%
    \ion@conflict@redefinedtest{1}{\ion@one}%
    \ion@conflict@redefinedtest{2}{\ion@two}%
    \ion@conflict@redefinedtest{3}{\ion@three}%
    \ion@conflict@redefinedtest{4}{\ion@four}%
    \ion@conflict@redefinedtest{5}{\ion@five}%
    \ion@conflict@redefinedtest{6}{\ion@six}%
    \ion@conflict@redefinedtest{7}{\ion@seven}%
    \ion@conflict@redefinedtest{8}{\ion@eight}%
    \ion@conflict@redefinedtest{9}{\ion@nine}%
  }
\endgroup
\AtBeginDocument{\ion@conflict@redefinedtest@macro}
```

7.8 Commands for current number

Numbers are processed by first storing one character after the other in an internal macro to be able to automatically group digits. The basic idea when adding single characters is

- remember, whether we are processing the thousands or the thousandths part of a number (`\ifion@beforedecimal`)

- calculate the number of digits processed modulo 3 plus 1 in the current part and
  - for the thousands part: add `\ion@thousands@sepa` for 1, `\ion@thousands@sepb` for 2, `\ion@thousands@sepc` for 3, ... after a digit
  - for the thousandths part: add `\ion@thousandths@sep` after each third digit

The macros `\ion@thousands@sep...` and `\ion@thousandths@sep` are empty by default. Before outputting the number, the number of digits in the thousands part is known and the correct `\ion@thousands@sep...` macro can be set to the thousands separator for correct grouping.

First of all, the `if`s, counters and empty separator macros are initialized.

```
\newif\ifion@currnum@firstchar\ion@currnum@firstchartrue
\newif\ifion@beforedecimal\ion@beforedecimaltrue
\newif\ifion@noexplicitthousands\ion@noexplicitthousandstrue
\newif\ifion@currnum@exponent\ion@currnum@exponentfalse
\newif\ifion@exponent@superscript\ion@exponent@superscriptfalse
\newcount\ion@thousands@currpos\ion@thousands@currpos=0
\newcount\ion@thousandths@currpos\ion@thousandths@currpos=0
\def\ion@currnum{}
\def\ion@thousands@sepa{}
\def\ion@thousands@sepb{}
\def\ion@thousands@sepc{}
\def\ion@thousands@sepd{}
\def\ion@thousands@sepe{}
\def\ion@thousands@sepf{}
\def\ion@thousands@sepg{}
\def\ion@thousands@seph{}
\def\ion@thousands@sepi{}
\def\ion@thousandths@sep{}
```

The macro `\ion@currnum@append` adds the character in its argument to the end of `\ion@currnum`. In the starred version adding of an empty separator macros is omitted.

```
\newcommand{\ion@currnum@append}{%
  \ion@currnum@firstcharfalse%
  \@ifstar{\ion@currnum@append@@}{\ion@currnum@append@}%
}
\newcommand*{\ion@currnum@append@@}[1]{%
  \ion@addto@macro{\ion@currnum}{#1}%
}
\newcommand*{\ion@currnum@append@}[1]{%
  \ifion@beforedecimal%
    %% push back (empty) separator and character
    \ifcase\ion@thousands@currpos%
      \ion@addto@macro{\ion@currnum}{#1}%
    \or%
      \ion@addto@macro{\ion@currnum}{\ion@thousands@sepa#1}%
    \or%
      \ion@addto@macro{\ion@currnum}{\ion@thousands@sepb#1}%
    \or%
      \ion@addto@macro{\ion@currnum}{\ion@thousands@sepc#1}%
```

```
    \or%
      \ion@addto@macro{\ion@currnum}{\ion@thousands@sepd#1}%
    \or%
      \ion@addto@macro{\ion@currnum}{\ion@thousands@sepe#1}%
    \or%
      \ion@addto@macro{\ion@currnum}{\ion@thousands@sepf#1}%
    \or%
      \ion@addto@macro{\ion@currnum}{\ion@thousands@sepg#1}%
    \or%
      \ion@addto@macro{\ion@currnum}{\ion@thousands@seph#1}%
    \or%
      \ion@addto@macro{\ion@currnum}{\ion@thousands@sepi#1}%
    \fi%
    %% advance thousands counter
    \advance\ion@thousands@currpos by1\relax%
    \ifnum\ion@thousands@currpos>\ion@grplenthousands%
      \ion@thousands@currpos=1%
    \fi%
  \else%
    %% push back (empty) separator and character
    \ifnum\ion@thousandths@currpos=\ion@grplenthousandths%
      \ion@addto@macro{\ion@currnum}{\ion@thousandths@sep#1}%
    \else%
      \ion@addto@macro{\ion@currnum}{#1}%
    \fi%
    %% advance thousandths counter
    \advance\ion@thousandths@currpos by1\relax%
    \ifnum\ion@thousandths@currpos>\ion@grplenthousandths%
      \ion@thousandths@currpos=1%
    \fi%
  \fi%
}
```

The `\ion@currnum@output` macro defines the empty separator macros (depending on the current configuration), outputs the current number, and resets everything for the next number.

```
\newcommand*{\ion@currnum@output}{%
  \begingroup%
    %% set automatic thousands separator
    \ifion@autothousands%
      \ifion@noexplicitthousands%
        \ifcase\ion@thousands@currpos%
          %% do nothing
        \or%
          \def\ion@thousands@sepa{\ion@thousands@curr}%
        \or%
          \def\ion@thousands@sepb{\ion@thousands@curr}%
        \or%
          \def\ion@thousands@sepc{\ion@thousands@curr}%
        \or%
          \def\ion@thousands@sepd{\ion@thousands@curr}%
        \or%
          \def\ion@thousands@sepe{\ion@thousands@curr}%
        \or%
```

```
          \def\ion@thousands@sepf{\ion@thousands@curr}%
        \or%
          \def\ion@thousands@sepg{\ion@thousands@curr}%
        \or%
          \def\ion@thousands@seph{\ion@thousands@curr}%
        \or%
          \def\ion@thousands@sepi{\ion@thousands@curr}%
        \fi%
      \fi%
    \fi%
    %% set automatic thousandths separator
    \ifion@autothousandths%
      \def\ion@thousandths@sep{\ion@thousandths@curr}%
    \fi%
    %% output number
    \ifion@currnum@exponent%
      \ifion@exponent@superscript%
        ^{\ion@currnum}%
      \else%
        {\ion@currnum}%
      \fi%
    \else
      \ion@currnum%
    \fi
  \endgroup%
  %% reset stuff for next number
  \ion@thousands@currpos=0%
  \ion@thousandths@currpos=0%
  \def\ion@currnum{}%
  \ion@currnum@firstchartrue%
  \ion@beforedecimaltrue%
  \ion@noexplicitthousandstrue%
  \ion@currnum@exponentfalse%
  \ion@exponent@superscriptfalse%
}
```

This macro is identical to `\l@addto@macro` from koma-script bundle.

```
\newcommand{\ion@addto@macro}[2]{%
  \begingroup\toks@\expandafter{#1#2}%
    \edef\@tempa{\endgroup\def\noexpand#1{\the\toks@}}%
  \@tempa}
```

Change History

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.

L

M

N

O

P

R

S

T

W