The **ionumbers** package*

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

July 18, 2009

**Warning: This is alpha software and may contain serious bugs!
Use with caution and on your own risk! Check output!**

Contents

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*) in triplets 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'.

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 lists like `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. Please input lists with white spaces after the separators as shown in the first case.

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` following immediatly a number (without any white space or other input in between) 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. (It is virtually impossible to handle it the same way as the other and assigning a macro definition to `e`.) In this case the number following an `e` in the same group will be grouped with curly braces `{}`. This is important to understand, e. g., the spacing in cases when that number begins with a sign.

1.2 Caveats

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; option 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.

A number following an `e` treated as exponential part of a number may be type-set as superscript (depending on the configuration). In the case of an exponential part there *may* be arbitrary input between `e` and the number in the exponent. Especially, 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 there is no easy way to circumvent `e` from being treated as begining of the exponential part in these cases (at least, I did not find it).

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.)

If you use, e.g., 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 $a_{1,234}$. Possibilities to prevent undesired automatic grouping are on the one hand input as `a_{1 2 3 4}` (with a space after at least every third digit) or on the other hand selectively switching of automatic grouping (see below).

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 (; *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 be added after each triplet of digits. 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 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 The command \newionumbersthousands{⟨*value*⟩}{⟨*definition*⟩} has two manda-
\newionumbersdecimal tory arguments. The first one is the name of the newly defined ⟨*value*⟩
\newionumbersthousandths for the thousands ⟨*key*⟩ and the second one its definition. The commands
\newionumbersexponent

`\newionumbersdecimal`, `\newionumbersthousandths`, and `\newionumbersexponent` work the same way for the `decimal`, `thousandths`, and `exponent` $\langle key \rangle$, respectively. There is a starred version of `\newionumbersexponent` (called `\newionumbersexponent*`) that typesets the following number as superscript.

To redefine an existing $\langle key \rangle$ definition there are `\renew...` versions of the previously described commands.

`\renewionumbersthousands`
`\renewionumbersdecimal`
`\renewionumbersthousandths`
`\renewionumbersexponent`

Notes on definitions:

1. All $\langle definition \rangle$s are set inside `\ionumbersoff` (see section 3.6). This means that numbers appearing in the $\langle definition \rangle$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{`$\langle stuff \rangle$`}` disables the features only for $\langle stuff \rangle$.

4 License

ionumbers is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License version 3 as published by the Free Software Foundation, not any later version.

ionumbers is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with ionumbers. If not, see <http://www.gnu.org/licenses/>.

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:

```
1 \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.

```
2 \newif\ifion@autothousands
3 \newif\ifion@autothousandths
```

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

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

Next the ⟨*key*⟩s are defined.

```
 6 \ion@defpackopts{comma}{%
 7   \def\ion@comma@reset{\csname ion@comma@#1\endcsname}%
 8   \def\ion@aftercomma@reset{\csname ion@aftercomma@#1\endcsname}}
 9 \ion@defpackopts{point}{%
10   \def\ion@point@reset{\csname ion@point@#1\endcsname}%
11   \def\ion@afterpoint@reset{\csname ion@afterpoint@#1\endcsname}}
12 \ion@defpackopts{decimal}{\def\ion@decimal@reset{%
13   \csname ion@decimal@#1\endcsname}}
14 \ion@defpackopts{thousands}{\def\ion@thousands@reset{%
15   \csname ion@thousands@#1\endcsname}}
16 \ion@defpackopts{thousandths}{\def\ion@thousandths@reset{%
17   \csname ion@thousandths@#1\endcsname}}
18 \ion@defpackopts{exponent}{\def\ion@exponent@reset{%
19   \csname ion@exponent@#1\endcsname}}
20 \ion@defpackopts{autothousands}[true]{\def\ion@autothousandsreset{%
21   \csname ion@autothousands#1\endcsname}\ion@autothousandsreset}
22 \ion@defpackopts{autothousandths}[true]{\def\ion@autothousandthsreset{%
23   \csname ion@autothousandths#1\endcsname}\ion@autothousandthsreset}
```

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

```
24 \ion@setpackopts{comma=default,point=default,thousands=default,%
25   decimal=default,thousandths=default,exponent=default,autothousands=false,%
26   autothousandths=false}
27 \DeclareOption*{\expandafter\ion@setpackopts\expandafter{\CurrentOption}}
28 \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.

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

\ionumberstyle

```
30 \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):

```
31 \ion@deflocopts{comma}{%
32   \def\ion@comma@curr{\csname ion@comma@#1\endcsname}%
33   \def\ion@aftercomma@curr{\csname ion@aftercomma@#1\endcsname}}
34 \ion@deflocopts{point}{%
35   \def\ion@point@curr{\csname ion@point@#1\endcsname}%
36   \def\ion@afterpoint@curr{\csname ion@afterpoint@#1\endcsname}}
37 \ion@deflocopts{decimal}{\def\ion@decimal@curr{%
38   \csname ion@decimal@#1\endcsname}}
39 \ion@deflocopts{thousands}{\def\ion@thousands@curr{%
40   \csname ion@thousands@#1\endcsname}}
41 \ion@deflocopts{thousandths}{\def\ion@thousandths@curr{%
42   \csname ion@thousandths@#1\endcsname}}
43 \ion@deflocopts{exponent}{\def\ion@exponent@curr{%
44   \csname ion@exponent@#1\endcsname}}
45 \ion@deflocopts{autothousands}[true]{\csname ion@autothousands#1\endcsname}
46 \ion@deflocopts{autothousandths}[true]{\csname ion@autothousandths#1\endcsname}
```

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

\ionumbersresetstyle

```
47 \newcommand*\ionumbersresetstyle{%
48   \ionumbersstyle{comma=reset,point=reset,thousands=reset,%
49     decimal=reset,thousandths=reset,exponent=reset,autothousands=reset,%
50     autothousandths=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).

```
51 \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

```
52 \newcommand*\newionumbersthousands[2]{\expandafter\newcommand%
53   \expandafter*\csname ion@thousands@#1\endcsname{\ionumbersoff{#2}}}
```

\newionumbersdecimal

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

\newionumbersthousandths

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

\newionumbersexponent

```
58 \newcommand*\newionumbersexponent{%
59   \@ifstar{\newionumbersexponent@@}{\newionumbersexponent@}}
60 \newcommand*\newionumbersexponent@[2]{\expandafter\newcommand%
61   \expandafter*\csname ion@exponent@#1\endcsname{\ionumbersoff{#2}}}
62 \newcommand*\newionumbersexponent@@[2]{\expandafter\newcommand%
63   \expandafter*\csname ion@exponent@#1\endcsname{\ionumbersoff{#2}%
64   \ion@exponent@superscripttrue}}
```

\renewionumbersthousands

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

\renewionumbersdecimal

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

\renewionumbersthousandths

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

\renewionumbersexponent

```
71 \newcommand*\renewionumbersexponent{%
72   \@ifstar{\renewionumbersexponent@@}{\renewionumbersexponent@}}
73 \newcommand*\renewionumbersexponent@[2]{\expandafter\renewcommand%
74   \expandafter*\csname ion@exponent@#1\endcsname{\ionumbersoff{#2}%
75   \ion@currnum@exponent}}
76 \newcommand*\renewionumbersexponent@@[2]{\expandafter\renewcommand%
77   \expandafter*\csname ion@exponent@#1\endcsname{\ionumbersoff{#2}%
78   \ion@currnum@exponent\ion@exponent@superscripttrue}}
```

7.4   Internal macros holding definitions for $\langle key\rangle{=}\langle value\rangle$ pairs

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

```
79 \mathchardef\ion@point@original="013A
80 \mathchardef\ion@comma@original="613B
81 \mathchardef\ion@plus@original="202B
82 \mathchardef\ion@minus@original="2200
83 \mathchardef\ion@zero@original="7030
84 \mathchardef\ion@one@original="7031
85 \mathchardef\ion@two@original="7032
```

```
86 \mathchardef\ion@three@original="7033
87 \mathchardef\ion@four@original="7034
88 \mathchardef\ion@five@original="7035
89 \mathchardef\ion@six@original="7036
90 \mathchardef\ion@seven@original="7037
91 \mathchardef\ion@eight@original="7038
92 \mathchardef\ion@nine@original="7039
93 \mathchardef\ion@e@original="7165
```

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

```
94 \def\ion@comma@ignore{}
95 \def\ion@comma@decimal{\ion@decimal@curr}
96 \def\ion@comma@thousands{\ion@thousands@curr}
97 \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).

```
98 \def\ion@aftercomma@ignore{}
99 \def\ion@aftercomma@decimal{\ion@beforedecimalfalse}
100 \def\ion@aftercomma@thousands{\ion@noexplicitthousandsfalse}
101 \def\ion@aftercomma@default{\ion@aftercomma@thousands}
```

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

```
102 \def\ion@point@ignore{}
103 \def\ion@point@decimal{\ion@decimal@curr}
104 \def\ion@point@thousands{\ion@thousands@curr}
105 \def\ion@point@default{\ion@point@decimal}
```

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

```
106 \def\ion@afterpoint@ignore{}
107 \def\ion@afterpoint@decimal{\ion@beforedecimalfalse}
108 \def\ion@afterpoint@thousands{\ion@noexplicitthousandsfalse}
109 \def\ion@afterpoint@default{\ion@afterpoint@decimal}
```

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

```
110 \mathchardef\ion@decimal@point="013A
111 \mathchardef\ion@decimal@comma="013B
112 \mathchardef\ion@decimal@punctpoint="613A
113 \mathchardef\ion@decimal@punctcomma="613B
114 \def\ion@decimal@default{\ion@decimal@point}
```

. . . the thousands output separator, . . .

```
115 \def\ion@thousands@none{}
116 \mathchardef\ion@thousands@comma="013B
117 \mathchardef\ion@thousands@point="013A
118 \mathchardef\ion@thousands@punctcomma="613B
119 \mathchardef\ion@thousands@punctpoint="613A
120 \def\ion@thousands@apostrophe{^\prime}
```

```
121 \def\ion@thousands@phantom{\phantom{\ion@point@original}}
122 \def\ion@thousands@space{\,}
123 \def\ion@thousands@default{\ion@thousands@punctcomma}
```

... the thousandths output separator, ...

```
124 \def\ion@thousandths@none{}
125 \mathchardef\ion@thousandths@comma="013B
126 \mathchardef\ion@thousandths@point="013A
127 \mathchardef\ion@thousandths@punctcomma="613B
128 \mathchardef\ion@thousandths@punctpoint="613A
129 \def\ion@thousandths@apostrophe{^\prime}
130 \def\ion@thousandths@phantom{\phantom{\ion@point@original}}
131 \def\ion@thousandths@space{\,}
132 \def\ion@thousandths@default{\ion@thousandths@space}
```

... and the exponent output separator are given.

```
133 \def\ion@exponent@none{}
134 \def\ion@exponent@original{\ion@e@original}
135 \def\ion@exponent@ite{\mathchar"7165\ion@currnum@exponenttrue}
136 \def\ion@exponent@itE{\mathchar"7145\ion@currnum@exponenttrue}
137 \def\ion@exponent@rme{\mathchar"7065\ion@currnum@exponenttrue}
138 \def\ion@exponent@rmE{\mathchar"7045\ion@currnum@exponenttrue}
139 \def\ion@exponent@timestento{\times10\,\ion@currnum@exponenttrue%
140   \ion@exponent@superscripttrue}
141 \def\ion@exponent@cdottento{\cdot10\,\ion@currnum@exponenttrue%
142   \ion@exponent@superscripttrue}
143 \def\ion@exponent@wedge{^\wedge\ion@currnum@exponenttrue}
144 \def\ion@exponent@default{\ion@exponent@original}
```

7.5   Enabling and disabling features

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

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

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

```
152 \def\ion@separators@math@active{\mathcode`,="8000\mathcode`.="8000\relax}
153 \def\ion@separators@math@inactive{\mathcode`,="613B\mathcode`.="013A\relax}
154 \def\ion@signs@math@active{\mathcode`+="8000\mathcode`-="8000\relax}
155 \def\ion@signs@math@inactive{\mathcode`+="202B\mathcode`-="2200\relax}
156 \def\ion@digits@math@active{\mathcode`0="8000\mathcode`1="8000\mathcode`2="8000%
157   \mathcode`3="8000\mathcode`4="8000\mathcode`5="8000\mathcode`6="8000%
158   \mathcode`7="8000\mathcode`8="8000\mathcode`9="8000\relax}
159 \def\ion@digits@math@inactive{\mathcode`0="7030\mathcode`1="7031%
160   \mathcode`2="7032\mathcode`3="7033\mathcode`4="7034\mathcode`5="7035%
161   \mathcode`6="7036\mathcode`7="7037\mathcode`8="7038\mathcode`9="7039\relax}
```

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

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

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

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

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

167 `\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.

168 `\def\ion@comma{%`
169   `\ion@ifnextdigit{%`
170   `\ion@currnum@append*{\ion@comma@curr}\ion@aftercomma@curr%`
171   `}{%`
172   `\ion@ifnextseparator{%`
173   `\ion@currnum@append*{\ion@comma@curr}\ion@aftercomma@curr%`
174   `\@warning{Too many separators}%`
175   `}{%`
176   `\ion@ifnextchar e{%`
177   `\ion@currnum@append*{\ion@comma@curr}\ion@aftercomma@curr%`
178   `\ion@currnum@output\ion@exponent@curr\@gobble%`
179   `}{%`
180   `\ion@currnum@output\ion@comma@original%`
181   `}%`
182   `}%`
183   `}%`
184 `}`
185 `\def\ion@point{%`

```
186   \ion@ifnextdigit{%
187     \ion@currnum@append*{\ion@point@curr}\ion@afterpoint@curr%
188   }{%
189     \ion@ifnextseparator{%
190       \ion@currnum@append*{\ion@point@curr}\ion@afterpoint@curr%
191         \@warning{Too many separators}%
192     }{%
193     \ion@ifnextchar e{%
194       \ion@currnum@append*{\ion@point@curr}\ion@afterpoint@curr%
195         \ion@currnum@output\ion@exponent@curr\@gobble%
196     }{%
197       \ion@currnum@output\ion@point@original%
198     }%
199   }%
200   }%
201 }
202 \def\ion@plus{%
203   \ion@iffirstchar{%
204     \ion@plus@original%
205   }{%
206     \ion@currnum@append*{\ion@plus@original}%
207   }%
208   \ion@ifnextdigit{%
209     %% nothing
210   }{%
211     \ion@ifnextseparator{%
212       %% nothing
213     }{%
214     \ion@ifnextsign{%
215       \@warning{Too many signs}%
216     }{%
217       \ion@currnum@output%
218     }%
219   }%
220   }%
221 }
222 \def\ion@minus{%
223   \ion@iffirstchar{%
224     \ion@minus@original%
225   }{%
226     \ion@currnum@append*{\ion@minus@original}%
227   }%
228   \ion@ifnextdigit{%
229     %% nothing
230   }{%
231     \ion@ifnextseparator{%
232       %% nothing
233     }{%
234     \ion@ifnextsign{%
235       \@warning{Too many signs}%
236     }{%
237       \ion@currnum@output%
238     }%
239   }%
```

```
240     }%
241 }
242 \def\ion@zero{%
243   \ion@iffirstchar{%
244     \ion@zero@original\ion@currnum@append{}%
245   }{%
246     \ion@currnum@append{\ion@zero@original}%
247   }%
248   \ion@ifnextdigit{%
249     %% nothing
250   }{%
251     \ion@ifnextseparator{%
252       %% nothing
253     }{%
254       \ion@ifnextchar e{%
255         \ion@currnum@output\ion@exponent@curr\@gobble%
256       }{%
257         \ion@currnum@output%
258       }%
259     }%
260   }%
261 }
262 \def\ion@one{%
263   \ion@iffirstchar{%
264     \ion@one@original\ion@currnum@append{}%
265   }{%
266     \ion@currnum@append{\ion@one@original}%
267   }%
268   \ion@ifnextdigit{%
269     %% nothing
270   }{%
271     \ion@ifnextseparator{%
272       %% nothing
273     }{%
274       \ion@ifnextchar e{%
275         \ion@currnum@output\ion@exponent@curr\@gobble%
276       }{%
277         \ion@currnum@output%
278       }%
279     }%
280   }%
281 }
282 \def\ion@two{%
283   \ion@iffirstchar{%
284     \ion@two@original\ion@currnum@append{}%
285   }{%
286     \ion@currnum@append{\ion@two@original}%
287   }%
288   \ion@ifnextdigit{%
289     %% nothing
290   }{%
291     \ion@ifnextseparator{%
292       %% nothing
293     }{%
```

```
294      \ion@ifnextchar e{%
295          \ion@currnum@output\ion@exponent@curr\@gobble%
296      }{%
297          \ion@currnum@output%
298      }%
299    }%
300  }%
301 }
302 \def\ion@three{%
303    \ion@iffirstchar{%
304      \ion@three@original\ion@currnum@append{}%
305    }{%
306      \ion@currnum@append{\ion@three@original}%
307    }%
308    \ion@ifnextdigit{%
309      %% nothing
310    }{%
311      \ion@ifnextseparator{%
312        %% nothing
313      }{%
314        \ion@ifnextchar e{%
315          \ion@currnum@output\ion@exponent@curr\@gobble%
316        }{%
317          \ion@currnum@output%
318        }%
319      }%
320    }%
321 }
322 \def\ion@four{%
323    \ion@iffirstchar{%
324      \ion@four@original\ion@currnum@append{}%
325    }{%
326      \ion@currnum@append{\ion@four@original}%
327    }%
328    \ion@ifnextdigit{%
329      %% nothing
330    }{%
331      \ion@ifnextseparator{%
332        %% nothing
333      }{%
334        \ion@ifnextchar e{%
335          \ion@currnum@output\ion@exponent@curr\@gobble%
336        }{%
337          \ion@currnum@output%
338        }%
339      }%
340    }%
341 }
342 \def\ion@five{%
343    \ion@iffirstchar{%
344      \ion@five@original\ion@currnum@append{}%
345    }{%
346      \ion@currnum@append{\ion@five@original}%
347    }%
```

```
348    \ion@ifnextdigit{%
349      %% nothing
350    }{%
351      \ion@ifnextseparator{%
352        %% nothing
353      }{%
354        \ion@ifnextchar e{%
355          \ion@currnum@output\ion@exponent@curr\@gobble%
356        }{%
357          \ion@currnum@output%
358        }%
359      }%
360    }%
361 }
362 \def\ion@six{%
363    \ion@iffirstchar{%
364      \ion@six@original\ion@currnum@append{}%
365    }{%
366      \ion@currnum@append{\ion@six@original}%
367    }%
368    \ion@ifnextdigit{%
369      %% nothing
370    }{%
371      \ion@ifnextseparator{%
372        %% nothing
373      }{%
374        \ion@ifnextchar e{%
375          \ion@currnum@output\ion@exponent@curr\@gobble%
376        }{%
377          \ion@currnum@output%
378        }%
379      }%
380    }%
381 }
382 \def\ion@seven{%
383    \ion@iffirstchar{%
384      \ion@seven@original\ion@currnum@append{}%
385    }{%
386      \ion@currnum@append{\ion@seven@original}%
387    }%
388    \ion@ifnextdigit{%
389      %% nothing
390    }{%
391      \ion@ifnextseparator{%
392        %% nothing
393      }{%
394        \ion@ifnextchar e{%
395          \ion@currnum@output\ion@exponent@curr\@gobble%
396        }{%
397          \ion@currnum@output%
398        }%
399      }%
400    }%
401 }
```

```
402 \def\ion@eight{%
403   \ion@iffirstchar{%
404     \ion@eight@original\ion@currnum@append{}%
405   }{%
406     \ion@currnum@append{\ion@eight@original}%
407   }%
408   \ion@ifnextdigit{%
409     %% nothing
410   }{%
411     \ion@ifnextseparator{%
412       %% nothing
413     }{%
414       \ion@ifnextchar e{%
415         \ion@currnum@output\ion@exponent@curr\@gobble%
416       }{%
417         \ion@currnum@output%
418       }%
419     }%
420   }%
421 }
422 \def\ion@nine{%
423   \ion@iffirstchar{%
424     \ion@nine@original\ion@currnum@append{}%
425   }{%
426     \ion@currnum@append{\ion@nine@original}%
427   }%
428   \ion@ifnextdigit{%
429     %% nothing
430   }{%
431     \ion@ifnextseparator{%
432       %% nothing
433     }{%
434       \ion@ifnextchar e{%
435         \ion@currnum@output\ion@exponent@curr\@gobble%
436       }{%
437         \ion@currnum@output%
438       }%
439     }%
440   }%
441 }
```

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.

```
442 \begingroup
443   \ion@separators@active\ion@signs@active\ion@digits@active
444   \gdef\ion@define@charmacros{%
445     \global\let,=\ion@comma%
446     \global\let.=\ion@point%
447     \global\let+=\ion@plus%
448     \global\let-=\ion@minus%
449     \global\let0=\ion@zero%
450     \global\let1=\ion@one%
451     \global\let2=\ion@two%
```

```
452     \global\let3=\ion@three%
453     \global\let4=\ion@four%
454     \global\let5=\ion@five%
455     \global\let6=\ion@six%
456     \global\let7=\ion@seven%
457     \global\let8=\ion@eight%
458     \global\let9=\ion@nine%
459   }
460 \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.

```
461 \def\ion@iffirstchar#1#2{%
462   \ifion@currnum@exponent%
463     #2%
464   \else%
465     \ifion@currnum@firstchar%
466       #1%
467     \else
468       #2%
469     \fi%
470   \fi%
471   \ion@currnum@firstcharfalse%
472 }
```

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

```
473 \long\def\ion@ifnextdigit#1#2{%
474   \def\reserved@a{#1}%
475   \def\reserved@b{#2}%
476   \futurelet\@let@token\ion@ifnextdigit@}
477 \def\ion@ifnextdigit@{%
478   \ifx\@let@token1\let\reserved@c\reserved@a\else%
479     \ifx\@let@token2\let\reserved@c\reserved@a\else%
480       \ifx\@let@token3\let\reserved@c\reserved@a\else%
481         \ifx\@let@token4\let\reserved@c\reserved@a\else%
482           \ifx\@let@token5\let\reserved@c\reserved@a\else%
483             \ifx\@let@token6\let\reserved@c\reserved@a\else%
484               \ifx\@let@token7\let\reserved@c\reserved@a\else%
485                 \ifx\@let@token8\let\reserved@c\reserved@a\else%
486                   \ifx\@let@token9\let\reserved@c\reserved@a\else%
487                     \ifx\@let@token0\let\reserved@c\reserved@a\else%
488                       \let\reserved@c\reserved@b%
489                     \fi%
490                   \fi%
491                 \fi%
492               \fi%
493             \fi%
494           \fi%
495         \fi%
496       \fi%
497     \fi%
498   \fi%
```

```
499    \reserved@c}
```

... for a separator ., ...

```
500 \long\def\ion@ifnextseparator#1#2{%
501    \def\reserved@a{#1}%
502    \def\reserved@b{#2}%
503    \futurelet\@let@token\ion@ifnextseparator@}
504 \def\ion@ifnextseparator@{%
505    \ifx\@let@token,\let\reserved@c\reserved@a\else%
506      \ifx\@let@token.\let\reserved@c\reserved@a\else%
507        \let\reserved@c\reserved@b%
508      \fi%
509    \fi%
510    \reserved@c}
```

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

```
511 \long\def\ion@ifnextsign#1#2{%
512    \def\reserved@a{#1}%
513    \def\reserved@b{#2}%
514    \futurelet\@let@token\ion@ifnextsign@}
515 \def\ion@ifnextsign@{%
516    \ifx\@let@token+\let\reserved@c\reserved@a\else%
517      \ifx\@let@token-\let\reserved@c\reserved@a\else%
518        \let\reserved@c\reserved@b%
519      \fi%
520    \fi%
521    \reserved@c}
```

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

```
522 \long\def\ion@ifnextchar#1#2#3{%
523    \let\reserved@d=#1%
524    \def\reserved@a{#2}%
525    \def\reserved@b{#3}%
526    \futurelet\@let@token\ion@ifnextchar@}
527 \def\ion@ifnextchar@{%
528    \ifx\@let@token\reserved@d%
529      \let\reserved@c\reserved@a%
530    \else%
531      \let\reserved@c\reserved@b%
532    \fi%
533    \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.

```
534 \newcommand*{\ion@conflict@package}[1]{%
535    \@ifpackageloaded{#1}{%
536      \PackageError{ionumbers}%
537      {Packages #1 and ionumbers conflict!\MessageBreak%
538      Do not load both packages in the same document}{}%
```

```
539    }{}%
540 }
541 \newcommand*{\ion@problem@package}[2]{%
542    \@ifpackageloaded{#1}{%
543      \PackageWarning{ionumbers}%
544      {Loading #1 and ionumbers is problematic!\MessageBreak#2}%
545    }{}%
546 }
547
548 \AtBeginDocument{%
549    \ion@conflict@package{ziffer}%
550    \ion@problem@package{dcolumn}{Use 'tabular's inside \string\ionumbersoff}%
551    \ion@problem@package{amsmath}{Load ionumbers after amsmath}%
552    \ion@problem@package{amsmath}{Use \string\operatorname\space inside
553      \string\ionumbersoff}%
554    \ion@problem@package{amsopn}{Use \string\operatorname\space inside
555      \string\ionumbersoff}%
556 }
```

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).

```
557 \newcommand*\ion@conflict@definedtest[1]{%
558    \ifx#1\@undefined\else\PackageWarning{ionumbers}%
559      {Potential conflict with other package(s) detected.\MessageBreak%
560      '\string#1' has already been defined. I will redefine it.\MessageBreak%
561      This might break other package(s)!\MessageBreak}\fi}
562 \begingroup
563    \ion@separators@active\ion@signs@active\ion@digits@active
564    \ion@conflict@definedtest{,}
565    \ion@conflict@definedtest{.}
566    \ion@conflict@definedtest{+}
567    \ion@conflict@definedtest{-}
568    \ion@conflict@definedtest{0}
569    \ion@conflict@definedtest{1}
570    \ion@conflict@definedtest{2}
571    \ion@conflict@definedtest{3}
572    \ion@conflict@definedtest{4}
573    \ion@conflict@definedtest{5}
574    \ion@conflict@definedtest{6}
575    \ion@conflict@definedtest{7}
576    \ion@conflict@definedtest{8}
577    \ion@conflict@definedtest{9}
578 \endgroup
```

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

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

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

```
580 \newcommand*{\ion@conflict@redefinedtest}[2]{%
581    \ifx#1#2\else\PackageWarning{ionumbers}%
582      {Potential conflict with other package(s) detected.\MessageBreak%
583      '\string#1' has been redefined. This might break ionumbers!\MessageBreak}%
```

```
584    \fi}
585  \begingroup
586    \ion@separators@active\ion@signs@active\ion@digits@active
587    \gdef\ion@conflict@redefinedtest@macro{%
588      \ion@conflict@redefinedtest{,}{\ion@comma}%
589      \ion@conflict@redefinedtest{.}{\ion@point}%
590      \ion@conflict@redefinedtest{+}{\ion@plus}%
591      \ion@conflict@redefinedtest{-}{\ion@minus}%
592      \ion@conflict@redefinedtest{0}{\ion@zero}%
593      \ion@conflict@redefinedtest{1}{\ion@one}%
594      \ion@conflict@redefinedtest{2}{\ion@two}%
595      \ion@conflict@redefinedtest{3}{\ion@three}%
596      \ion@conflict@redefinedtest{4}{\ion@four}%
597      \ion@conflict@redefinedtest{5}{\ion@five}%
598      \ion@conflict@redefinedtest{6}{\ion@six}%
599      \ion@conflict@redefinedtest{7}{\ion@seven}%
600      \ion@conflict@redefinedtest{8}{\ion@eight}%
601      \ion@conflict@redefinedtest{9}{\ion@nine}%
602    }
603  \endgroup
604  \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, and `\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 `ifs`, counters and empty separator macros are initialized.

```
605  \newif\ifion@currnum@firstchar\ion@currnum@firstchartrue
606  \newif\ifion@beforedecimal\ion@beforedecimaltrue
607  \newif\ifion@noexplicitthousands\ion@noexplicitthousandstrue
608  \newif\ifion@currnum@exponent\ion@currnum@exponentfalse
609  \newif\ifion@exponent@superscript\ion@exponent@superscriptfalse
610  \newcount\ion@thousands@currpos\ion@thousands@currpos=0
611  \newcount\ion@thousandths@currpos\ion@thousandths@currpos=0
612  \def\ion@currnum{}
```

```
613 \def\ion@thousands@sepa{}
614 \def\ion@thousands@sepb{}
615 \def\ion@thousands@sepc{}
616 \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.

```
617 \newcommand{\ion@currnum@append}{%
618   \ion@currnum@firstcharfalse%
619   \@ifstar{\ion@currnum@append@@}{\ion@currnum@append@}%
620 }
621 \newcommand*{\ion@currnum@append@@}[1]{%
622   \ion@addto@macro{\ion@currnum}{#1}%
623 }
624 \newcommand*{\ion@currnum@append@}[1]{%
625   \ifion@beforedecimal%
626     %% push back (empty) separator and character
627     \ifcase\ion@thousands@currpos%
628       \ion@addto@macro{\ion@currnum}{#1}%
629     \or%
630       \ion@addto@macro{\ion@currnum}{\ion@thousands@sepa#1}%
631     \or%
632       \ion@addto@macro{\ion@currnum}{\ion@thousands@sepb#1}%
633     \or%
634       \ion@addto@macro{\ion@currnum}{\ion@thousands@sepc#1}%
635     \fi%
636     %% advance thousands counter
637     \advance\ion@thousands@currpos by1\relax%
638     \ifnum\ion@thousands@currpos>3%
639       \ion@thousands@currpos=1%
640     \fi%
641   \else%
642     %% push back (empty) separator and character
643     \ifnum\ion@thousandths@currpos=3%
644       \ion@addto@macro{\ion@currnum}{\ion@thousandths@sep#1}%
645     \else%
646       \ion@addto@macro{\ion@currnum}{#1}%
647     \fi%
648     %% advance thousandths counter
649     \advance\ion@thousandths@currpos by1\relax%
650     \ifnum\ion@thousandths@currpos>3%
651       \ion@thousandths@currpos=1%
652     \fi%
653   \fi%
654 }
```

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.

```
655 \newcommand*{\ion@currnum@output}{%
656   \begingroup%
657     %% set automatic thousands separator
658     \ifion@autothousands%
659       \ifion@noexplicitthousands%
```

```
660        \ifcase\ion@thousands@currpos%
661            %% do nothing
662        \or%
663            \def\ion@thousands@sepa{\ion@thousands@curr}%
664        \or%
665            \def\ion@thousands@sepb{\ion@thousands@curr}%
666        \or%
667            \def\ion@thousands@sepc{\ion@thousands@curr}%
668        \fi%
669      \fi%
670    \fi%
671    %% set automatic thousandths separator
672    \ifion@autothousandths%
673        \def\ion@thousandths@sep{\ion@thousandths@curr}%
674    \fi%
675    %% output number
676    \ifion@currnum@exponent%
677      \ifion@exponent@superscript%
678        ^{\ion@currnum}%
679      \else%
680        {\ion@currnum}%
681      \fi%
682    \else
683      \ion@currnum%
684    \fi
685    \endgroup%
686    %% reset stuff for next number
687    \ion@thousands@currpos=0%
688    \ion@thousandths@currpos=0%
689    \def\ion@currnum{}%
690    \ion@currnum@firstchartrue%
691    \ion@beforedecimaltrue%
692    \ion@noexplicitthousandstrue%
693    \ion@currnum@exponentfalse%
694    \ion@exponent@superscriptfalse%
695 }
```

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

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

Change History

v0.2.0-alpha

General: initial .dtx version 1

v0.2.1-alpha

General: replaced website by e-mail address in all fields containing contact information 1

v0.2.2-alpha

General: fixed problem with single digit numbers without {}-grouping; conflict with ams-math/amsopn documented and handled with warning messages; thanks to Robert Nürnberg for

reporting these two problems . . 1
v0.2.3-alpha
    General: saved one \if and made
        sign/digit macros a bit clearer . 1
v0.2.4-alpha
    General: fixed bug with curly braces

in   \ion@problem@package
macro; thanks to Lars for re-
porting this problem 1

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.