The numprint package

Harald Harders
h.harders@tu-bs.de

Version v1.35, 2005/07/11; printed July 11, 2005

Abstract

This package prints numbers with a separator every three digits and convert numbers given as `12345.6e789` to $12\,345{,}6 \cdot 10^{789}$. Numbers are printed in the current mode (text or math) in order to use the correct font.

Many things, including the decimal sign, the thousand separator, as well as the product sign can be changed by the user, e.g., to reach $12{,}345.6\mathrm{e}789$. If requested, numprint can round numbers to a given number of digits.

If an optional argument is given it is printed upright as unit. Numbers can be rounded to a given number of digits.

The package supports an automatic, language-dependent change of the number format.

Tabular alignment using the `tabular(*)`, `array`, `tabularx`, and `longtable` environments (similar to the `dcolumn` and `rccol` packages) is supported using all features of numprint. Additional text can be added before and after the formatted number.

If you want to print numbers with different bases (octal, hexadecimal, etc.) use the `nbaseprt` package which comes with the `numprint` package.

Contents

Copyright

Remarks

! *The* `\fourdigitsep`*,* `\fourdigitnosep`*,* `\addmissingzero`*,* `\noaddmissingzero`*,* `\digits`*,* `\nodigits`*,* `\exponentsdigits`*, and* `\noexponentdigits` *commands have been renamed to* `\npfourdigitsep`*,* `\npfourdigitnosep`*,* `\npaddmissingzero`*,* `\npnoaddmissingzero`*,* `\npdigits`*,* `\npnodigits`*,* `\npexponentsdigits`*, resp.* `\npnoexponentdigits`*.*

! *From version 1.00 to 1.10, the column types have been changed; see section 6. If you want to preserve the old column types, use the package option* `oldcolumntypes`*.*

For typesetting this documentation, the usage of different font shapes has been reduced as much as possible in order to save disk memory and download times. Thus, for nearly all characters, the design size 10 pt has been used. This reduced the size of the PDF file for version 1.10 from 858 KB to 396 KB. Please excuse that shortcoming in the typography.

1 Load the package

To use this package place

```
\usepackage{numprint}
```

in the preamble of your document. No options are necessary but some are available. They are mentioned where their usage is described and in section B.1.

2 Print numbers in text and math mode

`\numprint` This package provides the command `\numprint[`⟨*unit*⟩`]{`⟨*number*⟩`}` that prints the ⟨*number*⟩ given in the required argument. The number is printed in the current mode (math or text mode) so that an eventual chosen difference between mathematical and text numbers stays visible (e.g., by using the eco package).

Numbers may contain of these characters: "`+-\pm0123456789.,eEdD`". Spaces, "`\,`" as well as "`~`" in the argument are ignored. Either a "`,`" or a "`.`" can be used as decimal sign. By default, no thousand separators are allowed in the argument.[1] "`E`", "`e`", "`D`", or "`d`"[2] is converted to an exponential format (e.g., $x \cdot 10^y$ or $x \times 10^y$, depending on the format settings described later). "`\pm`" and "`+-`" produce a ±.

For example, typing

```
\numprint{-123456}; \numprint{\pm 123456}; \numprint{+-3,1415927e-3.1}
```

[1] Section 8.2 describes how this can be changed.

[2] This is useful for FORTRAN produced numbers.

leads to "−123,456; ±123,456; ±3.141,592,7 × 10^−3.1". Notice that "." and "," can be mixed within one number and are converted to the chosen decimal sign for the output.

The number is printed in the active mode (text mode resp. math mode). This may be important if the digits are different in text and math mode as in this document that uses old-style figures in text and lining figures in math mode.[3] See the difference between "123,456.134 × 10^123" and "123,456.134 × 10^123", produced by

```
``\numprint{123456.134e123}'' and ``$\numprint{123456.134e123}$''
```

If no number is given before the exponential characters "e", "E", "d", resp. "D", a pure exponential format is generated. For example, typing

```
\numprint{e4.3242}
```

leads to "10^4.3242". This also works with given signs, e.g.,

```
\numprint{-e4.3242}, \numprint{+-e4.3242}
```

leads to "−10^4.3242, ±10^4.3242".

Since `\numprint` expands the argument before typesetting it you may also use commands inside the argument, e.g.,

```
\def\numberbefore{1234}%
\def\totalnumber{\numberbefore.5678}%
\numprint{\totalnumber}
```

leads to "1,234.567,8".

If the optional argument is given it is printed as a ⟨*unit*⟩ in math mode with an upright font (`\mathrm`), e.g.,

```
\numprint[N/mm^2]{-123456}
```

leads to "−123,456 N/mm^2".

By default, the space between the number and the unit is `\,`. One exception is the degree symbol which is typeset without a distance to the number, e.g., 360° in contrast to 273.15 °C (only a single degree symbol is typeset without a separator). The `numprint` package detects this automatically if the `\tcdegree` command of the `mathcomp` package, the `\textdegree` command of the `textcomp` package, or the `\degree` command of the `gensymb` package is used, e.g.,

```
\numprint[\tcdegree]{360}, \numprint[\degree]{360}
```

Unfortunately, typing in the "°" sign directly cannot be detected (any help welcome).

If you want to use one of the `textcomp` symbols ℃ (`\textcelsius`), Ω (`\textohm`), µ (`\textmu`), or ‰ (`\textperthousand`) you have to load either the `mathcomp` or the `gensymb` package because the unit is printed in math mode. If you decide to use the `textomp` package use `\tc`⟨*name*⟩ instead of `\text`⟨*name*⟩, e.g., `\tccelsius` instead of `\textcelsius`. If using `gensymb` just use `\`⟨*name*⟩, e.g., `\celsius`.[4]

`\np` Since it is timeconsuming to type in the long command `\numprint` for every

[3] This is only the case if the `eco` package is available on your system.

[4] If you still use `\text`⟨*name*⟩ numprint uses `\tc`⟨*name*⟩ or `\`⟨*name*⟩ and generates a warning if one of these commands is available, and produces an error if not.

number in a text the shortcut `\np` can be defined by specifying the package option `np`.

By default, numbers are written in the format $12\,345{,}123\,45 \cdot 10^{12\,345}$, for instance. This means, decimal sign "`,`", thousand separator "`\,`", and product sign "`{}\cdot{}`". This accords to German number formats and is a result of the history of this package. How you can change this is described in the following sections. Especially, have a look at the `autolanguage` package option in section 5.

2.1 Print counters and lengths

If you want to print a counter or a length, you may of course use

```
\numprint{\arabic{page}}
```

resp.

```
\makeatletter
\numprint[pt]{\strip@pt\textwidth}
\makeatother
```

But these methods are not really nice.

`\cntprint` For printing counters, you may use `\cntprint[`⟨*unit*⟩`]{`⟨*counter*⟩`}` where ⟨*unit*⟩ is an optional unit that is printed as by `\numprint`. ⟨*counter*⟩ is the name of a LaTeX counter, for example,

```
$\cntprint{page}$
```

leads to "5".

`\lenprint` Lengths can be printed using `\lenprint[`⟨*unit*⟩`]{`⟨*length*⟩`}`. ⟨*length*⟩ is the macro containing a LaTeX length or a TeX dimension, e.g., `\textwidth`. ⟨*unit*⟩ is a different thing here. If the unit is not given `\lenprint` uses LaTeX's standard and prints the length in `pt`:

```
$\lenprint{\textwidth}$
```

leads to "355 pt". Giving `pt` does not change anything. But if you use other units, e.g., `mm`, `cm`, or `in`, the length is written using the given unit:

```
$\lenprint[pt]{\textwidth}$,
$\lenprint[in]{\textwidth}$,
$\lenprint[bp]{\textwidth}$,
$\lenprint[mm]{\textwidth}$,
$\lenprint[cm]{3\textwidth}$
```

gives "355 pt, 4.913,1 in, 353.672,87 bp, 124.766,77 mm, 37.425,16 cm".

As you can see in the example, it is also possible to specify a factor in the argument, e.g. `3\textwidth`. When printing lengths, it is often desirable not to print all decimal places. `numprint` can round numbers, described in section 3.4.

`\lenprint` knows the units `pt`, `bp`, `in`, `ft`, `mm`, `cm`, `m`, and `km`. Section 4 describes how to add or change units.

3 Customization

3.1 Eliminate separators for four-digit numbers

At least in German it is common not to add a separator to four-digit numbers in non-technical texts, e.g., to typeset "1234" instead of "1,234", but longer numbers are separated: "12,345". If a number, in contrast, has less than five digits on one side of the decimal sign but five or more digits on the other side, separators are inserted on both sides, e.g., "1234.1234" but "1,234.123,45".

\npfourdigitsep
\npfourdigitnosep

This behaviour can be achieved using the command `\npfourdigitnosep`. If using this switch inside a group the change is local. You can switch back to separating with `\npfourdigitsep`. An example:

```
\npfourdigitnosep$\numprint{1234.1234}$, $\numprint{12345.12345}$ --
\npfourdigitsep$\numprint{1234.1234}$, $\numprint{12345.12345}$
```

Leads to "1234.1234, 12,345.123,45 – 1,234.123,4, 12,345.123,45". Default values can be set by the package options `sepfour` and `nosepfour`.

3.2 Add missing zeros before or after decimal sign

\npaddmissingzero
\npnoaddmissingzero

Sometimes people let out a leading zero or a zero after the decimalsign, e.g., "`123.`" or "`.123`". Numprint can add the left out zero, when `\addmissingzero` is used. If, however, no decimal sign is given, e.g., "`123`", no decimal sign or zero is appended. Adding zeros can be switched off with `\noaddmissingzero`. The corresponding package options are `addmissingzero` and `noaddmissinzero`. The default is `addmissingzero`.

3.3 Add a plus sign

\npaddplus
\npnoaddplus

Using the `\npaddplus` command or the package option `addplus`, a plus sign can be added to a number that is specified without a sign. This can be switched off using `\npnoaddplus` resp. `noaddplus`.

\npaddplusexponent
\npnoaddplusexponent

The commands `\npaddplusexponent` and `\npnoaddplusexponent` resp. the package options `addplusexponent` and `noaddplusexponent` do the same for the exponents that the commands/options described above do for the number.

3.4 Rounding numbers

\nprounddigits
\nproundexpdigits
\npnoround
\npnoroundexp

By default, as many digits are printed after the decimal sign, as the `\numprint` command gets as argument. This behaviour can be changed to print a given number of digits where the number is rounded resp. filled with zeros.

This can be switched on using the `\nprounddigits{`⟨*digits*⟩`}` command for ordinary numbers and the `\nproundexpdigits{`⟨*digits*⟩`}` command for exponents. Rounding is switched off with `\npnoround` resp. `\npnoroundexp`. For example,

```
\nprounddigits{2}$\numprint{1.123}$, $\numprint{1.149}$,
$\numprint{1}$, $\numprint{9.999}$ $\numprint{-9.999}$ --
\npnoround$\numprint{1.123}$, $\numprint{1.149}$,
$\numprint{1}$, $\numprint{9.999}$, $\numprint{-9.999}$
```

leads to "1.12, 1.15, 1.00, 10.00 −10.00 – 1.123, 1.149, 1, 9.999, −9.999"

3.5 Padding a number on the left side

\nplpadding
\npnolpadding

Sometimes it is desireable to have a number of a fixed length with the missing digits filled with a character (mostly the character "o", so this is the default). This can be achieved calling `\nplpadding[`⟨*character*⟩`]{`⟨*digits*⟩`}`. This affects only the mantissa, the part after the decimal sign and signs are not counted! If the orginal number has more digits than ⟨*digits*⟩ no characters will be inserted. For example,

```
\nplpadding{6}%
$\numprint{1234}$, $\numprint{-1234}$,
$\numprint{12345678}$, $\numprint{1234.5678}$ --
\nplpadding[x]{6}%
$\numprint{1234}$, $\numprint{-1234}$,
$\numprint{12345678}$, $\numprint{1234.5678}$
```

leads to "001,234, −001,234, 12,345,678, 001,234.567,8 – xx1,234, −xx1,234, 12,345,678, xx1,234.567,8"

`\npnolpadding` switches padding off.

3.6 Replace zeros by other characters

\npreplacenull
\npprintnull

For amounts of money, sometimes a zero after the decimal sign is replaced by different symbols, as for example "—". This can be done by calling the command `\npreplacenull{`⟨*replacement*⟩`}`, e.g.,

```
\npreplacenull{\mbox{---}}
```

Here, `\mbox` guarantees that "—" is printed in text mode.[5]

It can be switched off using `\npprintnull`.

3.7 Change the format of units

\npunitcommand

By default, the unit is printed in math mode with an upright font. This is reached by using the `\npunitcommand` macro which by default is defined as follows:

```
\newcommand*\npunitcommand[1]{\ensuremath{\mathrm{#1}}}
```

If you want to change this, redefine the `\npunitcommand`. You really should use either `\ensuremath` or `\text`[6] to ensure math or text mode respectively. `\npunitcommand` has to take one mandatory argument.

For example, a blue unit is reached by

```
\renewcommand*\npunitcommand[1]{\ensuremath{\color{blue}\mathrm{#1}}}
```

And here is the result: 300 N/mm^2

4 Add more units to `\lenprint`

\npdefunit

The command `\npdefunit{`⟨*unitname*⟩`}{`⟨*unit*⟩`}{`⟨*scale*⟩`}` can be used to define new units or to redefine existing ones. For example, `mm` is defined as follows:

```
\npdefunit{mm}{mm}{0.35145980351}
```

[5] You should better use `amsmath` and the command `\text` which preserves the correct text size, too.

[6] Provided by `amsmath`.

The first argument ⟨*unitname*⟩ is the LaTeX internal name that will be given as unit in the `\lenprint` command. The second argument ⟨*unit*⟩ is the text that will be printed out.

The ⟨*scale*⟩ arises from the fact that the default unit pt is defined as 1 pt = 1/72.27 in = 0.013,837,000,13 in and 1 in = 25.4 mm. Thus, 1 pt = 0.013,837,000,13 × 25.4 mm = 0.351,459,803,51 mm. For defining a new unit, the factor from pt to the desired unit has to be calculated and given as third argument of `\npdefunit`. To redefine a unit with unchanged scale, use `*` instead. For example,

```
\npdefunit{in}{\!''}{*}
```

redefines `in` to print ″ as unit instead of the default in. The `\!` is used to remove the separator between number and unit (`\,`) again. This only works correctly if `\,` and `\!` use the same measure.

```
$\lenprint[in]{\textwidth}$,
\npdefunit{in}{\!''}{*}%
$\lenprint[in]{\textwidth}$
```

leads to "4.913,1 in, 4.913,1″".

5 International support

As mentioned above, `numprint` uses German settings for numbers: thousand separator "`\,`", decimal sign "`,`", product sign "`\cdot`", unit separator `\,`, and no degree separator, by default. This will stay stable for compatibility with older versions even if its unlogical since the default language of LaTeX is English.

Using the package option `autolanguage` this can be fixed. If you are using this option without the `babel` package the settings are switched to English at `\begin{document}`: thousand separator "`,`", decimal sign ".", product sign "`\times`", unit separator `\,`, and no degree separator,

`\selectlanguage` If you are using the `babel` package in conjunction with the `autolanguage` package option, the behaviour of `\numprint` alters with the active language. If you, for instance, use

```
\selectlanguage{ngerman}
```

the German settings are selected. If you then switch back to English, the English settings are active again.

The current version supports English, German, and Portuguese. Unfortunately, I don't really know how to write numbers in other languages than German. I am quite sure that the English version also is correct. But please help me to add other languages.

As long as `numprint` does not support your language you may add the definitions by yourself. How this can be done is described in section 8.1.2.

6 Print aligned numbers in tabulars

Aligning numbers in tabulars is providede by the `dcolumn` and `rccol` packages. But they have two disadvantages. First, they do not support typesetting numbers

in the same way as \numprint does. Second, they force the numbers to be typeset in math mode. Thus, this packages provides own mechanisms to gain aligned numbers.

In former versions up to 1.00, the align mechanism in tabulars has been somehow weak because the author had to repeat the \numprint call in every table cell. This has been improved in version 1.10. For compatibility reasons, the old column types n and N have been preserved if you specify the oldcolumntypes package option; they will be discussed in section sec:column:old.

6.1 The new column types

column type n

The numprint package provides the column type n that takes two mandatory arguments. They define the number of digits before and the number of digits after the decimal sign. The results can be seen in the left column of the tabular below. You can use this column type as the normal column types, e.g.,

```
\begin{tabular}{n{3}{4}n{4}{2}}
```

The numbers are printed in a reserved space with the necessary width for the specified numbers, aligned at the decimal sign. If a column contains numbers that have an exponent it is appended left-aligned while the width of the column is extended by the required space. This is shown in the first column in the example below.

If you, in addition, want to reserve space for digits in the exponent you can insert one (specify numbers of digits before the decimal sign) or two (number of digits before and after the decimal sign) optional arguments, as can be seen in columns 2 and 3 in the example below. If you reserve space for the exponent, too long exponents may exceed the tabular cell (as can be seen in the second column).

This example tabular[7]

123.45×10^{12}	123.45×10^{12}	123.45×10^{12}	123.45×10^{12}
$12{,}345.678 \times 10^{123}$	$12{,}345.678 \times 10^{123}$	$12{,}345.678 \times 10^{123}$	$12{,}345.678 \times 10^{123}$
$123.45 \times 10^{12.3}$	$123.45 \times 10^{12.3}$	$123.45 \times 10^{12.3}$	$123.45 \times 10^{12\cdot3}$
$12{,}345.678 \times 10^{123.3}$	$12{,}345.678 \times 10^{123.3}$	$12{,}345.678 \times 10^{123.3}$	$12{,}345.678 \times 10^{123\cdot3}$

is produced by the following code:

```
\tabcolsep0mm\small
\begin{tabular}{|n{5}{3}|n[3]{5}{3}|n[3][1]{5}{3}|N{5}{3}|}
  \hline
  123.45e12& 123.45e12& 123.45e12 & 123.45e12 \\
  12345.678e123& 12345.678e123& 12345.678e123& 12345.678e123 \\
  123.45e12.3& 123.45e12.3& 123.45e12.3& 123.45e12.3 \\
  12345.678e123.3& 12345.678e123.3& 12345.678e123.3& 12345.678e123.3 \\
  \hline
\end{tabular}
```

column type N

The n column type prints the number in math mode. Thus, the first three columns in the example are printed with lining figures. To print numbers in text mode, you can use the N column type as shown in column four of the tabular.[8] It takes the same arguments as the n column type.

[7] The tabulars are ugly here. But the only important thing is to show the effects of the alignments.

[8] If you use a tabular environment that prints its arguments in math mode, e.g., the array environment, also N prints the numbers in math mode.

You may put additional text into tabular cells. The numprint package uses an algorithm to determine which part is the number and which is additional text. In order to preserve spaces and to use characters as text that could also be part of a number (e.g., digits or the characters “e”, “E”, “d”, or “D”) you have to enclose the text in braces, for example “`{hello} 1234`” instead of “`hello 1234`”. (In some rare cases, even two pairs of braces is necessary. This is the case if you want to use a single character that may also be used in a number, e.g., “`{{3}} 1234`”.[9]) If you don’t use these enclosing braces strange results may appear. For preceding text, you have to ensure that it has the same width for all lines of the same column. Have a look at the following example:

```
\begin{tabular}{n{2}{1}n{2}{1}n{2}{1}n[1]{2}{1}}
  \toprule
  {without braces}&
  {with braces}&
  {with braces and box}&
  \multicolumn{1}{l}{with braces, exp, and box}
  \\
  \midrule
  abc def 12,3e3 rt&
  {abc def } 12,3e3 { rt}&
  {\npmakebox[abc def ][l]{abc def }} 12,3e3 { rt}&
  {\npmakebox[abc def ][l]{abc def }} 12,3e3 { rt}
  \\
  more 45,1 txt&
  {more } 45,1 { txt}&
  {\npmakebox[abc def ][l]{more }} 45,1 { txt}&
  {\npmakebox[abc def ][l]{more }} 45,1 { txt}
  \\
  \midrule
  not blue 45,1 txt&
  {\color{blue}blue } 45,1 { txt}&
  {\color{blue}\npmakebox[abc def ][l]{blue }} 45,1 { txt}&
  {\color{blue}\npmakebox[abc def ][l]{blue }} 45,1 { txt}
  \\
  \bottomrule
\end{tabular}
```

The result looks as follows:

without braces	with braces	with braces and box	with braces, exp, and box
abc*def*12,3e3rt mor10$^{45.1}$txt	abc def 12.3×10^3 rt more 45.1 txt	abc def 12.3×10^3 rt more 45.1 txt	abc def 12.3×10^3 rt more 45.1 txt
notblu10$^{45.1}$txt	blue 45.1 txt	blue 45.1 txt	blue 45.1 txt

In the first column, the texts before and after the number are not enlosed by braces. Then, strange results appear: In the first line, “`de`” is interpreted as number. Thus, it is printed in math mode, and the rest of the cell, “`f 12,3e3 rt`” is printed as text, again. In the second line, the part “`e 45,1`” is printed as number.

This strange behaviour can be avoided by enclosing the texts by braces, as mentioned above and shown in the second column. Thus, the texts are printed

[9] It is the same if the text starts with one of some special commands that normally don’t occur at the beginning of tabular cells, `\end`, `\tabularnewline`, `\nprt@end`, `\endtabular`, `\csname`, and `\relax`.

correctly. A space in the enclosed texts can separate the text from the number, as done in the example. Since the left texts have different widths the alignment of the numbers is broken, here.

\npmakebox

This can be fixed by putting the contents in boxes, using the `\makebox` or `\npmakebox` command, as shown in the third column. The `\npmakebox` has a similar syntax as the `\makebox` command but uses a text instead of a length to determine the width of the box: `\npmakebox[`⟨*text 1*⟩`][`⟨*justification*⟩`]{`⟨*text 2*⟩`}`. The command determines which width ⟨*text 1*⟩ would have and typesets ⟨*text 2*⟩ into a box of this width.

If you put the preceding text in each line in such a box the number will be aligned as desired.

Still, the text on the right-hand side of the numbers is not aligned. This can be fixed by specifying the number of digits of the exponent, as shown in the fourth column.

The third line is a little bit different from the others. Here, not only text is inserted but also a command that changes the output of the numbers. Even such commands have to be enclosed by braces.

\multicolumn

For producing tabular cells that don't contain a number, you either have to enclose the cell contents by braces or to use the `\multicolumn` command as shown in the first line of the example above.

If you want to print a line in bold letters using the `\boldmath` command in math resp. `\bfseries` in text mode the alignment is not correct anymore. This is due to the fact that the bold letters are wider than the normal ones. You can avoid that problem if the font family provides a bold font shape that has the same width as the normal one. For the Computer Modern fonts, such a font shape exists. In text mode, you can access it by using

```
\fontseries{b}\selectfont
```

\mathversion
\npboldmath

If you also want to use that font in math mode, you may use the math version `npbold` by using `\mathversion{npbold}` or `\npboldmath`. In order to save memory, these commands and the `npbold` math version are only available if you call `numprint.sty` using the `boldmath` package option.

An example:

```
\begin{tabular}{lN{12}{3}n{12}{3}}
  \toprule
  normal:&
  123456123456.123e12&
  123456123456.123e12
  \\
  bold:&
  {\fontseries{b}\selectfont} 123456123456.123e12&
  {\npboldmath} 123456123456.123e12
  \\
  bold extended:&
  {\bfseries} 123456123456.123e12&
  {\boldmath} 123456123456.123e12
  \\
  \bottomrule
\end{tabular}
```

This produces:

normal:	$123{,}456{,}123{,}456.123 \times 10^{12}$	$123{,}456{,}123{,}456.123 \times 10^{12}$
bold:	$\mathbf{123{,}456{,}123{,}456.123 \times 10^{12}}$	$\mathbf{123{,}456{,}123{,}456.123 \times 10^{12}}$
bold extended:	$\mathbf{123{,}456{,}123{,}456.123 \times 10^{12}}$	$\mathbf{123{,}456{,}123{,}456.123 \times 10^{12}}$

If you want to add the same text or commands to all lines of a tabular column, you can use the "`>`" specifier in the declaration of the tabular column, as usual. You have to enclose its argument in an additional pair of braces, as shown in the example below. Unfortunately, the "`<`" specifier does not work properly. Therefore, the `\npafternum` command is defined that takes one argument which is printed after the number. The following example shows text before and after the number:

`\npafternum`

```
\begin{tabular}{%
  >{{before \npafternum{ after}}}n[2]{12}{3}%
  >{{\nprounddigits{4}}}n{3}{4}%
  >{{\color{blue}}}n{12}{3}}
  \toprule
  123456123456.123e12&
  12.12345&
  123456.23e1
  \\
  12345.123e12&
  12.1&
  14561234.562e12
  \\
  \bottomrule
\end{tabular}
```

This produces:

before	$123{,}456{,}123{,}456.123 \times 10^{12}$ after	12.124	$123{,}456.23 \times 10^{1}$
before	$12{,}345.123 \times 10^{12}$ after	12.100	$14{,}561{,}234.562 \times 10^{12}$

As can be seen in the second column, numbers can be rounded in special columns by inserting `\nproundigits` into a "`>`" specification. This can be said for all other commands that influence the format of numbers, too.

`\npunit`

Normally, units should not be typeset in the cells of a tabular. But if this is needed, it may be realized using the `\npunit` command, as shown here:

```
\begin{tabular}{>{{\npunit{N/mm^2}}}n{5}{3}}
  \toprule
  12345.123\\
  12.12\\
  {\npunit{psi}} 234.4\\
  4.3\\
  \bottomrule
\end{tabular}
```

This produces:

12,345.123	$\mathrm{N/mm^2}$
12.12	$\mathrm{N/mm^2}$
234.4	psi
4.3	$\mathrm{N/mm^2}$

The tabular alignment of the `numprint` package has been tested with the `tabular`, `tabular*`, `array`, `tabularx` [1], and `longtable` [2] environments. It may or may not run with other packages and environments.

6.2 The old column types

column type N
column type n

If the `oldcolumntype` package option is specified the column types `n` and `N` are defined differently than described in the previous section. The `n` column type alignes the base number to the decimal sign, the `N` column type additionaly alignes the exponent.

Use these commands as follows:

```
\tabcolsep0mm
\begin{tabular}{|n{5}{3}|N{5}{3}{3}|}
  \hline
  \numprint{123.45e12}& \numprint{123.45e12}\\
  \numprint{12345.678e123}& \numprint{12345.678e123}\\
  \numprint{123.45e12.3}& \numprint{123.45e12.3}\\
  \numprint{12345.678e123.3}& \numprint{12345.678e123.3}\\
  \hline
\end{tabular}
```

This leads to

123.45×10^{12}	123.45×10^{12}
$12{,}345.678 \times 10^{123}$	$12{,}345.678 \times 10^{123}$
$123.45 \times 10^{12.3}$	$123.45 \times 10^{12.3}$
$12{,}345.678 \times 10^{123.3}$	$12{,}345.678 \times 10^{123.3}$

The first argument defines the number of digits before the decimal sign, the second the number after. In case of the type `N` the third option defines the number of digits before the decimal sign in the exponent. It is not possible to define the numbers of digits after the decimal sign in the exponent; they are set to zero. Notice that the command `\numprint` has to be written again in each tabular entry, using the old column types.

6.3 Alignment in normal text

The alignment of the numbers in tabulars is realized by writing the number inside a box with the specified width. This functionality can also be used outside tabular environments. The `\npdigits{`⟨*before*⟩`}{`⟨*after*⟩`}` command switches on the alignment of numbers printed by `\numprint`. The first argument defines the number of digits before the decimal sign while the second argument defines the number of digits after it for the mantissa. Since exponents are normally integer numbers the syntax of the corresponding `\npexponentdigits` command is slightly different. Its syntax is `\npexponentdigits[`⟨*after*⟩`]{`⟨*before*⟩`}`. The mandatory argument defines the number of digits before the decimal sign of the exponent. If no optional argument is given, the number of digits after the decimal sign is set to zero. If it is given it defines the number of digits after the decimal sign.

\npdigits
\npexponentdigits

If the `\npdigits` or `\npexponentdigits` commands have been used inside a group the values are reset at the end of the group. Alignment can also be switched off using the `\npnodigits` resp. `\npnoexponentdigits` commands.

\npnodigits
\npnoexponentdigits

7 Error messages etc.

By default, `\numprint` produces an error message if the argument uses some invalid characters or if the number format is invalid. Some people use `\numprint` to do

strange things and thus use invalid arguments designedly. These people may switch off these error messages by using the package option `warning`.

If you want some debug messages to be written into the log file, use the package option `debug`.

8 Advanced customization

8.1 Changing the output

Most of the things described in this section are not necessary to be done by hand because the feature "automatic language support", described in section 5 does this automatically.

8.1.1 Without the `autolanguage` option

\npthousandsep
\npthousandthpartsep
\npdecimalsign
\npproductsign

By using the commands `\npdecimalsign{`⟨*Sign*⟩`}`, `\npthousandsep{`⟨*Separator*⟩`}`, `\npthousandthpartsep{`⟨*Separator*⟩`}`, and `\npproductsign{`⟨*Sign*⟩`}`,[10] several separators and symbols can be changed, e.g.,

```
\npdecimalsign{\ensuremath{\cdot}}\npthousandsep{,}\npproductsign{*}%
\numprint{-123456}; \numprint{3,1415927e-3.2}
```

leads to "$-123{,}456$; $3{\cdot}141{,}592{,}7 * 10^{-3{\cdot}2}$".

The `\npthousandsep` both changes the separators before and after the decimal sign. If you want to use different separators, you have to call `\npthousandthpartsep` *after* `\npthousandsep`.

The separators as well as the decimal sign are typeset in the same mode as the number itself (math or text). If you want to guarantee a special mode, you have to use `\ensuremath` for math or either `\mbox`, `\textrm`, or `\text`[11] for text mode.

The product sign, in contrast, is always printed in math mode. Thus, you don't have to add `\ensuremath` to use math commands.

\global

If using these commands inside a group (`{...}`, `\begingroup...\endgroup`, or an environment) the behaviour of the `\numprint` command is changed only locally (inside the current group). By preceding `\global` the change can be made global inside a group. For example:

```
Local:
\numprint{123e4},
{\npproductsign{\cdot}\numprint{123e4}},
\numprint{123e4}.
Global:
\numprint{123e4},
{\global\npproductsign{\cdot}\numprint{123e4}},
\numprint{123e4}
```

leads to the following:
Local: 123×10^4, $123 \cdot 10^4$, 123×10^4. Global: 123×10^4, $123 \cdot 10^4$, $123 \cdot 10^4$

The current version has the following defaults:

[10] These command did not have the "np" in older versions. This had to be changed in order to avoid an incompatibiliby with the french language of babel.

[11] Provided by `amsmath`.

```
\npthousandsep{\,}
\npdecimalsign{,}
\npproductsign{\cdot}
\npunitseparator{\,}
```

\npunitseparator

The space between the number and the unit is "\," by default. It can be changed using the command \npunitseparator{⟨*Separator*⟩}, e.g.,

```
\npunitseparator{~}
```

\npdegreeseparator

By default, no space is added between the number an a degree symbol. You may specify a separator for that using \npdegreeseparator{⟨*Separator*⟩}.

\nppercentseparator

By default, the same space as for normal units is added between the number an a percent sign. You may specify a different separator for that using \nppercentseparator{⟨*Separator*⟩}.

8.1.2 With the autolanguage option

If you are using the autolanguage option changes made with the commands described in the previous section get lost at the next change of the language or at \begin{document}. Thus, they cannot be used with this option in the way described there.

Thus, you have to redefine the commands that set the language-dependent numprint settings. For each known language, a command \npstyle⟨*language*⟩ is

\npstyleenglish

defined that does the changes, e.g., \npstyleenglish for English. This command is defined as follows:

```
\newcommand*\npstyleenglish{%
  \npthousandsep{,}%
  \npdecimalsign{.}%
  \npproductsign{\times}%
  \npunitseparator{\,}%
  \npdegreeseparator{}%
  \nppercentseparator{\nprt@unitsep}%
}
```

If you want to use different settings for this language, you have two possibilities:

1. Copy the definition for \npstyle⟨*language*⟩ from the style file and change it according to your wishes, for example:[12]

   ```
   \renewcommand*\npstyleenglish{%
     \npthousandsep{,}%
     \npdecimalsign{{\cdot}}%
     \npproductsign{\times}%
     \npunitseparator{\,}%
     \npdegreeseparator{}%
     \nppercentseparator{\nprt@unitsep}%
   }
   ```

2. Another way to add different settings to a language is to use \g@addto@macro to append commands to an existing \npstyle⟨*language*⟩ command, e.g.,

[12] Notice that you have to use \renewcommand* instead of \newcommand*.

```
\makeatletter
\g@addto@macro\npstyleenglish{%
  \nppercentseparator{}%
}%
\makeatother
```

This has the advantage that changes in the original command are not lost by using a copy. The disadvantage is that some commands may be called twice which is slightly slower.

The changes take effect when the style command is called the next time; this is when the language is changed the next time or at `\begin{document}`.

If the language you are using is not yet supported by `numprint` you may add support for it in the preamble of your document.

`\npaddtolanguage` The simplest case is a language that uses the same settings as one of the languages, already supported. If, for instance, you want to use Danish with the same settings as German, you just have to add

```
\npaddtolanguage{danish}{german}
```

to the preamble of your document.

If you, instead, want to use different settings, define a corresponding style command. Let's take Danish as an example, again. Define a command `\npstyledanish` which defines everything you want to change against the default (I choose some strange values for clearness):

```
\newcommand*\npstyledanish{%
  \npthousandsep{.}%
  \npdecimalsign{\ensuremath{\cdot}}%
  \npproductsign{*}%
  \npunitseparator{~}%
  \npdegreeseparator{}%
  \nppercentseparator{\nprt@unitsep}%
}
```

In addition, append the call of this command to the language-switching command for Danish:

```
\npaddtolanguage{danish}{danish}
```

8.2 Changing the argument parsing

It has been said above that thousand separators are not allowed in the argument of the `\numprint` command. This can be customized by the user.

`\nprt@dotlist` For most elements in the input, `\numprint` uses lists that contain the corresponding characters. The macro `\nprt@dotlist` contains the characters interpreted as decimal signs. It is defined as followed:

```
\newcommand*\nprt@dotlist{.,}
```

If you, for example, only want to allow the dot as decimal sign, redefined the list:

```
\renewcommand*\nprt@dotlist{.}
```

If you want to do this in your document rather than in the configureation file `numprint.cfg`—see section 9.3— you have to enclose this by `\makeatletter` and `\makeatother`.

`\nprt@explist` The `\nprt@explist` command contains the characters interpreted as delimiter between mantissa and exponent. By default it contains "`eEdD`". Redefine it as `\nprt@dotlist`.

`\nprt@ignorelist` The `\nprt@ignorelist` command contains a list of characters that are ignored in the input (in addition to spaces, "`\,`", and "`~`"). It is empty by default. If you, for example, only want to allow dots as decimal sign and commas may occur as thousand separators in the input, you may use following redefinition:

```
\renewcommand*\nprt@dotlist{.}
\renewcommand*\nprt@ignorelist{,}
```

Then, "`,`" is ignored in the input and "`.`" is interpreted as decimal sign. For example,

```
\makeatletter
{\renewcommand*\nprt@dotlist{.}%
\renewcommand*\nprt@ignorelist{,}%
\numprint{12,234.123,45e1,2,3.0}}
\makeatother
```

leads to "12,234.123,45 $\times$ $10^{123.0}$".

`\nprt@signlist` The `\nprt@signlist` command contains the list of known signs. By default, it is set to "`+-\pm`". You may change the list of accepted signs by redefining `\nprt@signlist`. If, for instance, the letter "`*`" is intended to be a sign, just type in

```
\renewcommand*\nprt@signlist{+-\pm *}
```

This character is typeset when using it as sign. But this indicates one problem: The sign might differ between text and math mode, as show in the following example: In text mode, `\numprint{*1234}` occurs as "*1,234" while is is typeset as "$*$1,234" in math mode. To avoid that, you may define a command that prints

`\nprt@sign@*` the sign. This command must have a name according to `\nprt@list@`⟨*sign*⟩. In this case, you have to define `\nprt@list@*`. Since the sign might be arbitrary characters, you should define the command as follows:

```
\expandafter\newcommand\csname nprt@sign@*\endcsname{\ensuremath{*}}
```

With this command, `\numprint{*1234}` occurs as "$*$1,234" in text mode and as "$*$1,234" in math mode.

Because of this mechanism to print a sign, it is not possible to use other macro names than `\pm` for signs.[13] You have to use single characters as the shown "`*`".

9 Some tricks

9.1 Let the signs depend on the mode

`\nprt@sign@+` The default signs are typeset in math mode independently of the mode the number

`\nprt@sign@-` ist printed. If you are using a font in which the sings of text and math mode differ

`\nprt@sign@+-`

[13] `\pm` is handled separately.

much, this may be unsatisfactory. Then, you can typeset different signs for text and math mode. Therefore, it is used that the default signs use the same macros as user signs, described in section 8.2. They are defined as follows:

```
\expandafter\newcommand\csname nprt@sign@+\endcsname{\ensuremath{+}}
\expandafter\newcommand\csname nprt@sign@-\endcsname{\ensuremath{-}}
\expandafter\newcommand\csname nprt@sign@+-\endcsname{\ensuremath{\pm}}
```

If you, for instance, don't want to use the math minus for numbers in text mode but another character, you may redefine `\nprt@sign@-`:

```
\expandafter\renewcommand\csname nprt@sign@-\endcsname{%
  \ifmmode -\else ---\fi}
```

With this definition, `\numprint{-1234}` leads to "—1,234" resp. "−1,234" in text resp. math mode.

9.2 Typing negative numbers in red

If you want to print negative number in red colors, you can use the `\nprt@sign@-` command, too. The following example shows how to do it:

```
\usepackage{color}
\makeatletter
\expandafter\renewcommand\csname nprt@sign@-\endcsname{%
  \color{red}\ensuremath{-}}
\makeatother
```

With this definition,

```
\numprint{1234}, \numprint{-1234},
\numprint{1234e-123}, \numprint{-1234e123}.
```

leads to "1,234, −1,234, $1{,}234 \times 10^{-123}$, $-1{,}234 \times 10^{123}$."[14] To avoid a negative exponent beeing printed in red for a positive mantissa, a hack is included in the page which is described in appendix F.9, page 53.

9.3 Configuration file

If your LaTeX installation provides a file `numprint.cfg` in the TeX search path, it is loaded by `numprint.sty` as last action. Thus, you may add all changes and extensions, new languages for instance, into this file.

A Compatiblity to older versions

In most cases, the user macros of this package (the macros not containing a "`@`" in their name) should be compatible to older versions. The parsing of the argument has been improved that some arguments of `\numprint` may be accepted or not in contrast to the older version.

The spacing of aligned numbers has also been corrected. Thus, this is be incompatible to the older version if you are using alignment in the exponent or math environmens other than `\textstyle`.

[14] Wheather you can see the effect in the output depends on the viewer; in PostScript and PDF the red color works, in many dvi viewers, it doesn't.

B Lists of options and commands

This section contains lists of all package options resp. available commands. Items that belong together and may be exclusive are printed in groups together.

B.1 Package options

The default values are marked by *.

`warning`	Produce warnings rather than error messages.
`error`*	Produce warnings rather than error messages.
`autolanguage`	Switch the settings language dependent.
`noautolanguage`*	Fixed settings.
`sepfour`*	Separator for four-digit numbers.
`nosepfour`	No separator for four-digit numbers.
`addmissingzero`*	Add missings zeros before or after the decimal sign.
`noaddmissingzero`	Don't do that.
`addplus`	Add a plus to a number without a sign.
`noaddplus`*	Don't do that.
`addplusexponent`	Add a plus to a number without a sign.
`noaddplusexponent`*	Don't do that.
`oldcolumntypes`	Define the old column types that need to use the `\numprint` inside the tabular.
`newcolumntypes`*	Use the new column types.
`boldmath`	Define the `npbold` math version.
`np`	Define the shortcut `\np` for `\numprint`.
`debug`	Produce debug information in the log file.

B.2 Commands

`\npfourdigitsep`	Switch on separating four-digit numbers.
`\npfourdigitnosep`	Switch off separating four-digit numbers.
`\npaddmissingzero`	Switch on adding missings zeros before or after the decimal sign.
`\npnoaddmissingzero`	Switch off adding missings zeros before or after the decimal sign.
`\npaddplus`	Add a plus to a number without a sign.
`\npnoaddplus`	Don't do that.
`\npaddplusexponent`	Add a plus to the exponent when it has no sign.
`\npnoaddplusexponent`	Don't do that.
`\np`	Shortcut for `\numprint` (only available with package option `np`).
`\numprint`	Typesets a number (the package's main command).

`\npdecimalsign`	Change the decimal sign.
`\npthousandsep`	Change the thousand separator (before and after the decimal sign).
`\npthousandspartsep`	Change the thousand separator (only after the decimal sign).
`\npproductsign`	Change the product sign.
`\npunitseparator`	Change the separator between a number and a unit.
`\npdegreeseparator`	Change the separator between a number and a degree symbol.
`\nppercentseparator`	Change the separator between a number and a percent sign.
`\nprounddigits`	Declare how many digits will be printed after the decimal sign.
`\npnoround`	Switch off rounding and print numbers as given.
`\nproundexpdigits`	Declare how many digits will be printed after the decimal sign in the exponent.
`\npnoroundexp`	Switch off rounding for the exponent.
`\nplpadding`	Declare up to how many digits the number will be padded at the lefthand side.
`\npnolpadding`	Switch off padding.
`\npreplacenull`	Replace the after-decimal-sign part by another text if it is zero.
`\npprintnull`	Print zeros after the decimal sign.
`\npdigits`	Switch on aligned number printing with given digits before and after the decimal sign.
`\npnodigits`	Print numbers in a box that is as wide as needed by the number.
`\npexponentdigits`	Switch on aligned printing of the exponent.
`\npnoexponentdigits`	Switch alignment off for the exponent.
`\npaddtolanguage`	Adds language definitions to the extras section of a `babel` language.
`\npstyledefault`	Defines the settings in standard format.
`\npstylegerman`	Defines the German number format.
`\npstyleenglish`	Defines the English number format.
`\npmakebox`	Command similar to `\makebox` but with text instead of length in first optional argument.
`\npboldmath`	Bold math version with digits with the same width as normal digits.
`\npafternum`	Puts text after the number in tabulars.
`\npunit`	Sets the unit in tabulars.

C Known bugs

- When aligning the exponent for tabulars, the distance between the "10" and the exponent is too small.

D To do

- Add more languages to the automatic international support.
- Add support for "<" in tabular definitions.
- Avoid many of the temporary variables.
- Add support for numbers in tabulars that are right aligned or centered.

E Acknowledgements

- Tilman Finke, tfinke@it-and-law.de, had the idea of rounding numbers.
- Stephan Helma, s.p.helma@gmx.net, has implemented padding numbers on the left side. This function has been slightly changed by me.
- Portuguese support by Vilar Camara Neto and Luis.

References

[1] Carlisle, David: *The* ***tabularx*** *package*, version 2.07, 1999. CTAN:macros/latex/contrib/tabularx/.

[2] Carlisle, David: *The* ***longtable*** *package*, version 4.10, 2000. CTAN:macros/latex/contrib/longtable/.

[3] Guthöhrlein, Eckhart: *The* ***rccol*** *package*, version 1.1a, 2000. CTAN:macros/latex/contrib/rccol/.

F The implementation

Heading of the package:

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{numprint}
  [2005/07/11  v1.35  Print numbers (HH)]
```

F.1 Load packages

Load package array.sty for the new column types.

```
\RequirePackage{array}
```

F.2 Package options and settings

Define a boolean if the first digit is separated in a four digit number. Default is true for compatibility with older versions.

```
\newif\ifnprt@numsepfour
```

Show warnings or errors?

```
\newif\ifnprt@errormessage
```

Add missing zeros?

```
\newif\ifnprt@addmissingzero
```

Add missing plus signs?

```
\newif\ifnprt@addplus@mantissa
\newif\ifnprt@addplus@exponent
```

Switch the style depending on the language automatically?

```
\newif\ifnprt@autolanguage
```

Old or new column types?

```
\newif\ifnprt@newcolumntype
```

Declare new math type?

```
\newif\ifnprt@npbold
```

`\npfourdigitsep` Switch separating of the fourth digit on.

```
\newcommand*\npfourdigitsep{\nprt@numsepfourtrue}%
```

`\npfourdigitnosep` Switch separating of the fourth digit off.

```
\newcommand*\npfourdigitnosep{\nprt@numsepfourfalse}%
```

`\npaddmissingzero` Add a missing zero before or after decimalsign.

```
\newcommand*\npaddmissingzero{\nprt@addmissingzerotrue}%
```

`\npnoaddmissingzero` Don't add a missing zero before or after decimalsign.

```
\newcommand*\npnoaddmissingzero{\nprt@addmissingzerofalse}%
```

`\npaddplus` Add a plus to the mantissa if no sign is given.

```
\newcommand*\npaddplus{\nprt@addplus@mantissatrue}%
```

`\npnoaddplus` Don't add a plus to the mantissa if no sign is given.

```
\newcommand*\npnoaddplus{\nprt@addplus@mantissafalse}%
```

`\npaddplusexponent` Add a plus to the exponent if no sign is given.

```
\newcommand*\npaddplusexponent{\nprt@addplus@exponenttrue}%
```

`\npnoaddplusexponent` Don't add a plus to the exponent if no sign is given.

```
\newcommand*\npnoaddplusexponent{\nprt@addplus@exponentfalse}%
```

Declare options. `nosepfour` switches seperating of the fourth digit off, `sepfour` switches it on. Default is on four compatibility reasons with older versions.

```
\DeclareOption{nosepfour}{\npfourdigitnosep}
\DeclareOption{sepfour}{\npfourdigitsep}
```

Add missing zeros after decimal sign?

```
\DeclareOption{noaddmissingzero}{\npnoaddmissingzero}
\DeclareOption{addmissingzero}{\npaddmissingzero}
```

Add a plus sign to the mantissa if no sign is given?

```
\DeclareOption{noaddplus}{\npnoaddplus}
\DeclareOption{addplus}{\npaddplus}
```

Add a plus sign to the exponent if no sign is given?

```
\DeclareOption{noaddplusexponent}{\npnoaddplusexponent}
\DeclareOption{addplusexponent}{\npaddplusexponent}
```

Add missing zeros after decimal sign?

```
\DeclareOption{noautolanguage}{\nprt@autolanguagefalse}
\DeclareOption{autolanguage}{\nprt@autolanguagetrue}
```

`\np` Define a shortcut for the `\numprint` command?

```
\DeclareOption{np}{\newcommand*\np{\numprint}}
```

Produce warnings or errors?

```
\DeclareOption{oldcolumntypes}{\nprt@newcolumntypefalse}
\DeclareOption{newcolumntypes}{\nprt@newcolumntypetrue}
```

Declare new math type?

```
\DeclareOption{boldmath}{\nprt@npboldtrue}
```

Produce warnings or errors?

```
\DeclareOption{warning}{\nprt@errormessagefalse}
\DeclareOption{error}{\nprt@errormessagetrue}
```

Generate some debug information to the log file?

```
\newcommand*\nprt@debug[1]{}
\DeclareOption{debug}{%
  \renewcommand*\nprt@debug[1]{\PackageInfo{numprint}{#1}}%
}
```

Execute default options and the given options.

```
\ExecuteOptions{sepfour,addmissingzero,error,noautolanguage,newcolumntypes}
\ProcessOptions\relax
```

Define commands to change the output of the `\numprint` command.

`\npdecimalsign` Change the decimal sign. In English it is normally ".", in German ",". The additional pair of braces `{}` prevents from inserting additional space in math mode, e.g., $1,2$ versus 1,2.

```
\newcommand*\npdecimalsign[1]{\def\nprt@decimal{{#1}}}
```

`\npthousandsep` Change the thousand separator. In English it often is ",", in German "." or "" (`\,`). Here again the additional `{}` is used as above. This command changes both the sign before and after the decimal sign. If you want a different sign after the comma you have to call `\npthousandthpartsep` after `\npthousandsep`.

```
\newcommand*\npthousandsep[1]{\def\nprt@separator@before{{#1}}%
  \def\nprt@separator@after{{#1}}}
```

`\npthousandthpartsep` Change the thousand separator after the decismal sign.

```
\newcommand*\npthousandthpartsep[1]{\def\nprt@separator@after{{#1}}}
```

`\npproductsign` Change the product sign that is printed in numbers with exponent like 123×10^4. Normally it is `\cdot`. In American texts sometimes `\times`. The two pairs of braces `{}` ensure the correct spacing on the left and right side of the product sign, e.g., $3{\cdot}4$ versus $3 \cdot 4$.

```
\newcommand*\npproductsign[1]{\def\nprt@prod{\ensuremath{{}#1{}}}}
```

`\npunitseparator` Change the separator between number an unit. Here again the additional `{}` is used as above.

```
\newcommand*\npunitseparator[1]{\def\nprt@unitsep{{#1}}}
```

`\npdegreeseparator` Change the separator between number an unit. Here again the additional `{}` is used as above.

```
\newcommand*\npdegreeseparator[1]{\def\nprt@degreesep{{#1}}}
```

`\nppercentseparator` Change the separator between number an unit. Here again the additional `{}` is used as above.

```
\newcommand*\nppercentseparator[1]{\def\nprt@percentsep{{#1}}}
```

`\nprt@fillnull` Defines command `#1` with a number of `#2` zeros filled. This should reside in Sect. F.7 but is called before.

```
\def\nprt@fillnull#1#2{%
  \@tempcnta=-1
  \loop
    \g@addto@macro{#1}{0}%
    \advance\@tempcnta by 1
  \ifnum\@tempcnta<#2
  \repeat
}
```

`\nprounddigits` Define a command that sets a count of digits after which the number is rounded. This command defines `\nprt@rounddigits` that stores the count and `\nprt@roundnull` that contains of `\nprt@rounddigits` "0" digits.

```
\newcommand\nprounddigits[1]{%
  \def\nprt@rounddigits{#1}%
  \def\nprt@roundnull{}%
  \nprt@fillnull{\nprt@roundnull}{#1}%
}
```

`\npnoround` A command to switch off rounding.

```
\newcommand\npnoround{\nprounddigits{-1}}
```

Don't round by default.

```
\npnoround
```

`\nproundexpdigits` `\npnoroundexp` The same for exponents.

```
\newcommand\nproundexpdigits[1]{%
  \def\nprt@roundexpdigits{#1}%
  \def\nprt@roundexpnull{}%
  \nprt@fillnull{\nprt@roundexpnull}{#1}%
  }
\newcommand\npnoroundexp{\nproundexpdigits{-1}}
\npnoroundexp
```

`\nplpadding` This command sets the total counts of digits a number should contain. The missing digits are filled with the character from the second argument (which is 0 by default).

```
\newcommand\nplpadding[2][0]{%
  \def\nprt@lpaddigits{#2}%
  \def\nprt@lpadchar{#1}%
}
```

`\npnolpadding` The command to switch of padding.

```
\newcommand*\npnolpadding{\nplpadding[\@empty]{-1}}
```

Don't pad by default.

```
\npnolpadding
```

`\npreplacenull` Optionally, one or more zeros after the decimal sign can be replaced by other text, e.g., "—"'.

```
\newcommand*\npreplacenull[1]{\def\nprt@replacenull{#1}}
```

`\npprintnull` Switch off replacement, again.

```
\newcommand*\npprintnull{\let\nprt@replacenull=\@empty}
```

Print the zero by default.

```
\npprintnull
```

`\npunitcommand` This command is used to typeset the unit.

```
\newcommand*\npunitcommand[1]{\ensuremath{\mathrm{#1}}}
```

`\npdigits` With this command the user can switch to aligned printing. The first parameter stands for the digits before the decimal sign and the second for the digits after it.

```
\newif\ifnprt@mantissa@fixeddigits
\newcommand*\npdigits[2]{%
  \edef\nprt@mantissa@fixeddigits@before{#1}%
  \edef\nprt@mantissa@fixeddigits@after{#2}%
  \nprt@mantissa@fixeddigitstrue
}
```

Initialize the numbers.

```
\def\nprt@mantissa@fixeddigits@before{-1}%
\def\nprt@mantissa@fixeddigits@after{-1}%
```

`\npnodigits` Switch back to normal printing.

```
\newcommand*\npnodigits{\nprt@mantissa@fixeddigitsfalse}
```

`\npexponentdigits` With this command the user can switch to aligned printing of the exponent. The parameter stands for the digits before the decimal sign.

```
\newif\ifnprt@exponent@fixeddigits
\newcommand*\npexponentdigits[2][0]{%
  \edef\nprt@exponent@fixeddigits@before{#2}%
  \edef\nprt@exponent@fixeddigits@after{#1}%
  \nprt@exponent@fixeddigitstrue
}
```

Initialize the numbers.

```
\def\nprt@exponent@fixeddigits@before{-1}%
\def\nprt@exponent@fixeddigits@after{-1}%
```

`\npnoexponentdigits` Switch back to normal printing.

```
\newcommand*\npnoexponentdigits{\nprt@exponent@fixeddigitsfalse}
```

F.3 Error and warning messages

Define a boolean which helps `\numprint` to detect errors.

```
\newif\ifnprt@argumenterror
```

`\nprt@error` Define `\nprt@error{`⟨*Message*⟩`}{`⟨*Help text*⟩`}` which prints a warning resp. an error message, depending on the package options `warning` and `error`.

```
\newcommand\nprt@error[2]{%
  \ifnprt@errormessage
    \PackageError{numprint}{#1}{#2}%
  \else
    \PackageWarning{numprint}{#1}%
  \fi
```

The boolean is set to "true". Then, `\numprint` knows that an error occured.

```
  \nprt@argumenterrortrue
}
```

F.4 String parsing

The `\IfCharInString` package does not work in the tabular alignment context. Thus, define an own command `\nprt@IfCharInString` that does the same.

`\nprt@charfound`

```
\newif\ifnprt@charfound
```

`\nprt@IfCharInString` ????

```
\newcommand*\nprt@IfCharInString[2]{%
  \nprt@charfoundfalse
  \begingroup
    \def\nprt@searchfor{#1}%
    \edef\nprt@argtwo{#2}%
    \expandafter\nprt@@IfCharInString\nprt@argtwo\@empty\@empty
  \endgroup
  \ifnprt@charfound
    \expandafter\@firstoftwo
  \else
    \expandafter\@secondoftwo
  \fi
}
```

`\nprt@@IfCharInString` ????

```
  \def\nprt@@IfCharInString#1#2\@empty{%
    \def\nprt@argone{#1}%
    \edef\nprt@argtwo{#2}%
    \ifx\nprt@searchfor\nprt@argone
```

```
      \global\nprt@charfoundtrue
    \else
      \ifx\nprt@argtwo\@empty
      \else
        \nprt@@IfCharInString#2\@empty
      \fi
    \fi
  }
```

F.5 Parsing of the \numprint argument

\nprt@plus@test
\nprt@minus@test
\nprt@plusminus@test

Define the signs as commands. This is necessary to be able to compare them with other characters.

```
\newcommand*\nprt@plus@test{+}
\newcommand*\nprt@minus@test{-}
\newcommand*\nprt@plusminus@test{\pm}
```

\nprt@numberlist
\nprt@dotlist
\nprt@epxlist
\nprt@signlist

Define lists of valid characters for different elements of numbers for parsing the mandatory argument of the \numprint command. It has nothing to do with the output. \nprt@numberlist contains digits, \nprt@dotlist valid decimal signs, \nprt@explist the characters that start the exponent, and \nprt@signlist the valid signs, where "+-" as alias for "\pm" does not have to be specified seperately.

```
\newcommand*\nprt@numberlist{0123456789}
\newcommand*\nprt@dotlist{.,}
\newcommand*\nprt@explist{eEdD}
\newcommand*\nprt@signlist{+-\pm}
\newcommand*\nprt@ignorelist{}
```

Counters for the number of digits before and after the decimal sign of the mantissa.

```
\newcounter{nprt@mantissa@digitsbefore}%
\newcounter{nprt@mantissa@digitsafter}%
```

Counters for the number of digits before and after the decimal sign of the exponent.

```
\newcounter{nprt@exponent@digitsbefore}%
\newcounter{nprt@exponent@digitsafter}%
```

Boolean to store if an exponent will be printed.

```
\newif\ifnprt@expfound
```

Boolean to store if mantissa resp. exponent contain a decimal sign.

```
\newif\ifnprt@mantissa@decimalfound
\newif\ifnprt@exponent@decimalfound
```

\nprt@testsign

Define \nprt@testsign{⟨*Number type*⟩}{⟨*Number*⟩} which tests wheather a sign is given and then starts \nprt@testnumber (the call actually is done by \nprt@@testsign).

```
\newcommand*\nprt@testsign[2]{%
```

First, store the expanded arguments in macros.

```
  \edef\nprt@commandname{#1}%
  \edef\nprt@tmp{#2}%
```

Call the working command `\nprt@@testsign`. The large number of `\expandafter` calls is necessary to ensure that the second to fourth argument are already expanded (thus, `#2` and `#3` are single characters). Append enough `\@empty` to ensure the argument-end marker is found even for an empty number.

```
\expandafter\nprt@@testsign\expandafter{%
  \expandafter\nprt@commandname\expandafter}%
  \nprt@tmp\@empty\@empty\@empty\@empty
}
```

`\nprt@@testsign` The first argument is the Number type ("mantissa" or "exponent"). Because it is longer than one character, it has to be enclosed in braces when calling this function (see the previous code line). The arguments `#2` to `#4` are the given number, where `#2` and `#3` contain of the first resp. second character of the number, while `#4` contains the rest.

```
\def\nprt@@testsign#1#2#3#4\@empty{%
```

Store the first argument to a macro.

```
  \edef\nprt@commandname{#1}%
```

Define the macros that store the digits before respectivly after the decimal sign. They are filled digit by digit and start empty.

```
  \expandafter\xdef\csname nprt@#1@before\endcsname{\@empty}%
  \expandafter\xdef\csname nprt@#1@after\endcsname{\@empty}%
```

Yet, no digits are stored.

```
  \setcounter{nprt@#1@digitsbefore}{0}%
  \setcounter{nprt@#1@digitsafter}{0}%
```

Test wheather the first character of the number, `#2`, contains a sign symbol which is listed in `\nprt@signlist`.

```
  \nprt@IfCharInString{#2}{\nprt@signlist}{%
```

If yes, store that sign in the command `\nprt@`⟨*#2*⟩`@sign`.

```
    \expandafter\xdef\csname nprt@#1@sign\endcsname{#2}%
```

If the sign is a "+" the second character of the number may be a "-" to replace that combination by `\pm`. Thus, do an extra handling of that case.

```
    \expandafter\ifx\csname nprt@#1@sign\endcsname\nprt@plus@test
      \def\nprt@tmp{#3}%
      \ifx\nprt@tmp\nprt@minus@test
```

The second character, `#3`, *is* a "-", thus redefine `\nprt@`⟨*#2*⟩`@sign` to `\pm`.

```
        \expandafter\xdef\csname nprt@#1@sign\endcsname{+-}%
```

The digits start at the third character of the number string which is `#4`. Start `\nprt@testnumber` to parse the digits of the number if a number is given after the given sign "+-". If not, empty `\nprt@`⟨*#1*⟩ that `\numprint` can determine that only a sign has been given.

```
        \def\nprt@tmp{#4}%
        \ifx\nprt@tmp\@empty
          \expandafter\edef\csname nprt@#1\endcsname{\@empty}%
        \else
          \expandafter\nprt@testnumber\expandafter\nprt@commandname#4\@empty
        \fi
      \else
```

If there is a single sign character "+", the digits start at #3. Start \nprt@testnumber to parse the digits of the number if a number is given after the given sign "+". If not, empty \nprt@⟨#1⟩ that \numprint can determine that only a sign has been given.

```
        \ifx#3\@empty
          \expandafter\edef\csname nprt@#1\endcsname{\@empty}%
        \else
          \expandafter\nprt@testnumber\expandafter\nprt@commandname#3#4\@empty
        \fi
      \fi
    \else
```

If the sign is a "\pm" store "+- as sign.

```
      \expandafter\ifx\csname nprt@#1@sign\endcsname\nprt@plusminus@test
        \expandafter\xdef\csname nprt@#1@sign\endcsname{+-}%
      \fi
```

If there is a single sign character other than "+", the digits start at #3. Start \nprt@testnumber to parse the digits of the number if a number is given after the given sign. If not, empty \nprt@⟨#1⟩ that \numprint can determine that only a sign has been given.

```
      \ifx#3\@empty
        \expandafter\edef\csname nprt@#1\endcsname{\@empty}%
      \else
        \expandafter\nprt@testnumber\expandafter\nprt@commandname#3#4\@empty
      \fi
    \fi
  }{%
```

If there is no sign, set \nprt@⟨#2⟩@sign empty.

```
    \expandafter\xdef\csname nprt@#1@sign\endcsname{\@empty}%
```

The digits start at #2. Start \nprt@testnumber to parse the digits of the number.

```
    \expandafter\nprt@testnumber\expandafter\nprt@commandname#2#3#4\@empty
  }%
}
```

\nprt@testnumber As with \nprt@@testsign, the first argument is the Number type, while the second and third arguments contain the number. #2 contains the first character of the remaining number string, #3 the rest.

```
\def\nprt@testnumber#1#2#3\@empty{%
```

Store the arguments in macros.

```
  \edef\nprt@commandname{#1}%
  \edef\nprt@argthree{#3}%
```

Test wheather the current character is a valid character for a real number (say a digit or a decimal sign).

```
  \nprt@IfCharInString{#2}{\nprt@numberlist\nprt@dotlist}{%
```

If this is the case, continue testing. If the current character is a decimal sign, set the boolean \ifnprt@⟨*Number type*⟩@decimalfound that a decimal sign has been found. If this has been done before, the number contains two decimal signs which is not allowed, thus, generate an error message.

```
    \nprt@IfCharInString{#2}{\nprt@dotlist}{%
```

```
      \csname ifnprt@#1@decimalfound\endcsname
        \nprt@error{More than one decimal sign used}{The mantissa
          or the exponent may only contain a maximum of one decimal
          sign (one of the list `\nprt@dotlist')}%
      \else
        \csname nprt@#1@decimalfoundtrue\endcsname
      \fi
    }{%
```

If the current character is no decimal sign it has to be a digit. If the decimal sign has been found before, this digit is in the real part of the number. Then, add it at the end of the after-decimal-sign part. Also, increase the number of found digits.

```
      \csname ifnprt@#1@decimalfound\endcsname
        \expandafter\g@addto@macro\csname nprt@#1@after\endcsname{#2}%
        \stepcounter{nprt@#1@digitsafter}%
      \else
```

If the decimal sign has not been found before, this digit is in the integer part of the number. Then, add it at the end of the before-decimal-sign part.

```
        \expandafter\g@addto@macro\csname nprt@#1@before\endcsname{#2}%
        \stepcounter{nprt@#1@digitsbefore}%
      \fi
    }%
```

If the next character is not `\@empty` and thus, the end of the number is not reached, start `\nprt@testnumber` resursively to parse the next character.

```
    \ifx\nprt@argthree\@empty
    \else
      \expandafter\nprt@testnumber\expandafter\nprt@commandname#3\@empty
    \fi
  }{%
```

The current character is neither a digit nor a decimal sign. Thus, it is a invalid character; produce an error message.

```
    \nprt@error{Invalid number format. Printing the
      argument\MessageBreak
      verbatim}{Something is wrong in the format of the number}%
  }%
}
```

`\nprt@testcharacter` This macro parses the whole mandatory argument of `\numprint`. This means it is tested on invalid characters and on a mantissa and an exponent. The first argument is the current character while `#2` is the rest of the argument, not parsed yet.

```
\def\nprt@testcharacter#1#2\@empty{%
```

Store the second argument to a macro.

```
  \edef\nprt@argtwo{#2}%
```

Test wheather the current character is a valid one.

```
  \nprt@IfCharInString{#1}{%
    \nprt@numberlist\nprt@dotlist\nprt@explist\nprt@signlist\nprt@ignorelist}{%
```

Yes, it is valid.

Now, test wheather it is one of the ignored characters.

```
    \nprt@IfCharInString{#1}{\nprt@ignorelist}{%
```

```
      \nprt@debug{Character `\noexpand#1' ignored}%
  }{%
```

Yes, it is valid.

Now, test wheather it is one of the characters that start the exponent. If yes, set `\ifnprt@expfound` to "true". If in addition, this has been done before, you have used more than one exponent starting character; produce an error message.

```
      \nprt@IfCharInString{#1}{\nprt@explist}{%
        \ifnprt@expfound
          \nprt@error{Character for exponent (`\nprt@explist') used
            more than once}{The argument of \string\numprint\space may
            only contain one of following characters: `\nprt@explist'}%
        \fi
        \nprt@expfoundtrue
      }{%
```

If the current character is not an exponent-starting character it is either a part of the mantissa or the exponent, depending on wheather the exponent has been started before. Add the current character to the corresponding command that stores the mantissa resp. the exponent.

```
        \ifnprt@expfound
          \g@addto@macro\nprt@exponent{#1}%
        \else
          \g@addto@macro\nprt@mantissa{#1}%
        \fi
      }%
    }%
```

If we have not reached the end of the argument, call `\nprt@testcharacter` recursively.

```
    \ifx\nprt@argtwo\@empty
    \else
      \nprt@testcharacter#2\@empty\@empty\@empty
    \fi
  }{%
```

If the character is not valid produce an error message.

```
    \nprt@error{Invalid characters `#1' in mandatory argument
      of\MessageBreak
      \string\numprint. Allowed are\MessageBreak
      `\nprt@numberlist\nprt@dotlist\nprt@explist\nprt@signlist\nprt@ignorelist'}{%
      You may only use the specified characters in the argument.}%
  }%
}
```

F.6 Table alignment

F.6.1 Aligned numbers, also for ordinary text

Define some lengths that help to calculate the width of a number.

```
\newlength{\nprt@digitwidth}%
\newlength{\nprt@sepwidth}%
\newlength{\nprt@decimalwidth}%
\newlength{\nprt@blockwidth}%
```

`\nprt@calcblockwidth` Define `\nprt@calcblockwidth{`⟨*Number type*⟩`}{`⟨*Position*⟩`}{`⟨*Math mode*⟩`}`, where ⟨*Number type*⟩ is either "mantissa" or "exponent", ⟨*Position*⟩ is the position relative to the decimal sign ("before" or "after"), ⟨*Math mode*⟩ is a math mode command (`\displaystyle`, `\textstyle`, `\scriptstyle`, or `\scriptscriptstyle`).

This macro calculates the width of a block if aligned output is requested. The resulted width is stored in the length `\nprt@blockwidth`.

```
\newcommand*\nprt@calcblockwidth[3]{%
```

Store the arguments in macros.

```
  \edef\nprt@argone{#1}%
  \edef\nprt@argtwo{#2}%
  \edef\nprt@argthree{#3}%
```

Define a macro with the contents "mantissa" to be able to compare it.

```
  \edef\nprt@mantissaname{mantissa}%
```

Define a macro with the contents "after" to be able to compare it.

```
  \edef\nprt@aftername{after}%
```

If the width for the mantissa is to be calculated, enter this code part.

```
  \ifx\nprt@argone\nprt@mantissaname
```

The width of the digits and separators changes between text and math mode. If math mode is active, proceed here.

```
    \ifmmode
```

Calculate the width of digits. It is assumed that all digits have the same width as a zero. First, execute parameter `#3` to switch to the current math style. This is be done that way since it is not possible to export length values from `\mathchoice`.

```
      \settowidth{\nprt@digitwidth}{$#3%
        0$}%
```

Calculate the width of the separators. This may differ from before and after the decimal sign.

```
      \settowidth{\nprt@sepwidth}{$#3%
        \csname nprt@separator@#2\endcsname$}%
```

Calculate the width of the decimal sign.

```
      \settowidth{\nprt@decimalwidth}{$#3%
        \nprt@decimal$}%
    \else
```

Do the same for text mode.

```
      \settowidth{\nprt@digitwidth}{0}%
      \settowidth{\nprt@sepwidth}{\csname nprt@separator@#2\endcsname}%
      \settowidth{\nprt@decimalwidth}{\nprt@decimal}%
    \fi
```

The same for the exponent.

```
  \else
    \ifmmode
      \settowidth{\nprt@digitwidth}{$#3%
        {}^{0}$}%
      \settowidth{\nprt@sepwidth}{$#3%
        {}^{\csname nprt@separator@#2\endcsname}$}%
      \settowidth{\nprt@decimalwidth}{$#3%
        {}^{\nprt@decimal}$}%
```

```
    \else
      \settowidth{\nprt@digitwidth}{\textsuperscript{0}}%
      \settowidth{\nprt@sepwidth}{%
        \textsuperscript{\csname nprt@separator@#2\endcsname}}%
      \settowidth{\nprt@decimalwidth}{\textsuperscript{\nprt@decimal}}%
    \fi
  \fi
```

Output to the log file.

```
  \nprt@debug{Widths for #1 #2 decimal sign
    (\ifx\nprt@argthree\@empty text mode\else math mode #3\fi):\MessageBreak
    digits \the\nprt@digitwidth,
    separators \the\nprt@sepwidth,\MessageBreak
    decimal sign \the\nprt@decimalwidth}%
```

Produce a warning if the current number exceeds the reserved space. Signs ($+-\pm$) are not taken into account.

```
  \ifnum\csname nprt@#1@fixeddigits@#2\endcsname<%
    \csname thenprt@#1@digits#2\endcsname
    \PackageWarning{numprint}{#1 exceeds reserved space
      #2\MessageBreak
      decimal sign}%
  \fi
```

Calculate the width of the given number of digits without separators.

```
  \setlength{\nprt@blockwidth}{%
    \csname nprt@#1@fixeddigits@#2\endcsname\nprt@digitwidth}%
```

Calculate how many separators are put into the number.

```
  \setcounter{nprt@blockcnt}{\csname nprt@#1@fixeddigits@#2\endcsname}%
  \addtocounter{nprt@blockcnt}{-1}%
  \divide\c@nprt@blockcnt 3%
```

If four-digit length numbers are not separated, delete the number of separators again.

```
  \ifnprt@numsepfour
  \else
    \ifnum\csname nprt@#1@fixeddigits@before\endcsname<5
      \ifnum\csname nprt@#1@fixeddigits@after\endcsname<5
        \setcounter{nprt@blockcnt}{0}%
      \fi
    \fi
  \fi
```

Add the width of the separators to the width.

```
  \addtolength{\nprt@blockwidth}{\thenprt@blockcnt\nprt@sepwidth}%
```

Add the width of the decimal sign to the width if it is after the decimal sign and there shall be digits after the decimal sign.

```
  \ifx\nprt@argtwo\nprt@aftername
    \expandafter\ifnum\csname nprt@#1@fixeddigits@after\endcsname>0
      \addtolength{\nprt@blockwidth}{\the\nprt@decimalwidth}%
    \fi
  \fi
}
```

\npunit The \npunit command takes as argument a unit that is printed in every cell of a table when using the (new) n or N column types.

```
\newcommand*\npunit[1]{\def\nprt@unit{#1}}
```

\npnounit Initialize \nprt@unit.

```
\edef\nprt@unit{\@empty}
```

\npafternum The \npunit command takes as argument some text that is printed after the number in every cell of a table when using the (new) n or N column types.

```
\newcommand*\npafternum[1]{\def\nprt@afternum{#1}}
```

\nprt@afternum Initialize \nprt@afternum.

```
\edef\nprt@afternum{\@empty}
```

\npmakebox The \npmakebox is similar to the \makebox command but it takes a text as first optional argument instead of a length. The width of the box is calculated by the width of this text.

```
\DeclareRobustCommand*\npmakebox{%
  \@ifnextchar[% ]
  {\nprt@makebox}{\makebox}%
}
```

\nprt@makebox The internal part of the \npmakebox command.

```
\newcommand*\nprt@makebox{}
\def\nprt@makebox[#1]{%
  \settowidth\@tempdima{#1}%
  \makebox[\@tempdima]%
}
```

Declare a bold math alphabet npbold that aligns with normal digits.

```
\ifnprt@npbold
  \DeclareMathVersion{npbold}
  \SetSymbolFont{operators}{npbold}{OT1}{cmr} {b}{n}
  \SetSymbolFont{letters}  {npbold}{OML}{cmm} {b}{it}
  \SetSymbolFont{symbols}  {npbold}{OMS}{cmsy}{b}{n}
  \SetMathAlphabet\mathsf{npbold}{OT1}{cmss}{b}{n}
  \SetMathAlphabet\mathit{npbold}{OT1}{cmr}{b}{it}
```

\npboldmath Switch to that bold math alphabet.

```
  \def\npboldmath{\@nomath\npboldmath
                  \mathversion{npbold}}
\fi
```

F.6.2 Auxilliary routines for the new column types

This code has been developed starting from the rccol package by Eckhart Guthöhrlein [3]. Some small bugs of that package have been corrected here, too.

\nprt@digittoks Token list that will contain all characters of a tabular cell that are allowed for \numprint.

```
\newtoks\nprt@digittoks
```

`\nprt@pretoks` Token list with all tokens before the number itself.

```
\newtoks\nprt@pretoks
```

`\nprt@posttoks` Token list with all tokens after the number itself.

```
\newtoks\nprt@posttoks
```

`\ifnprt@numfound` Has the number already been found in the parsing of the tabular cell?

```
\newif\ifnprt@numfound
```

`\nprt@begin` This macro is executed at the begin of each tabular cell.

```
\def\nprt@begin{%
```

Initialize the tokens and macros.

```
  \nprt@digittoks={}%
  \nprt@pretoks={}%
  \nprt@posttoks={}%
  \edef\nprt@unit{\@empty}%
  \edef\nprt@afternum{\@empty}%
  \nprt@numfoundfalse
```

Set the allowed characters. This macro is made empty when the number itself is read-in totally.

```
  \edef\nprt@allowedchars{\nprt@numberlist\nprt@dotlist\nprt@explist
    \nprt@signlist\nprt@ignorelist}%
```

Start to parse the tabular cell.

```
  \nprt@getnexttok
}
```

`\nprt@saveothertok` Adds the current token to the list of tokens before or after the number.

```
\def\nprt@saveothertok#1{%
  \ifnprt@numfound
```

If the number has already been found it is ended now. This is marked by clearing the list of allowed characters.

```
    \def\nprt@allowedchars{}%
    \nprt@posttoks=\expandafter{\the\nprt@posttoks#1}%
  \else
    \nprt@pretoks=\expandafter{\the\nprt@pretoks#1}%
  \fi
}
```

`\nprt@getnexttok` Parse the tabular cell. The argument is the next token of the tabular cell. Inside this command, the end of the tabular cell is detected by different possibilities to end a cell.

```
\def\nprt@getnexttok#1{%
```

If the current token is `\tabularnewline` or `\\` (which is the same) the tabular cell is finished.

```
  \ifx\tabularnewline#1%
```

Redefine the `\nprt@next` command that is called at the end of this command to execute the found `\tabularnewline` command.

```
    \let\nprt@next\tabularnewline
  \else
```

If this tabular cell is not in the last column and a `&` is found, the `\nprt@end` command is found that is inserted into this cell using the `<` specifier by the columnn types.

```
    \ifx\end#1%
```

Redefine `\nprt@next` to execute `\nprt@end` at the end of this command.

```
      \let\nprt@next\end
    \else
```

If this is the last cell of the tabular, the `\end` part of `\end{tabular}` or `\end{tabular*}` is found.

```
      \ifx\nprt@end#1%
```

Test wheather it is the normal or the star version of the environment. Is this really necessary?

```
        \let\nprt@next\nprt@end
      \else
```

If this is the last cell of the tabular and the tabular has been called using `\tabular` ... `\endtabular` instead of using the environment call, `\endtabular` is found.

```
        \ifx\endtabular#1%
          \let\nprt@next\endtabular
        \else
```

For `tabularx`, a test on `\csname` has to be added.

```
          \ifx\csname#1%
            \let\nprt@next\csname
          \else
```

If the last cell of a line is empty, `\relax` will be found.

```
            \ifx\relax#1%
              \let\nprt@next\relax
            \else
```

If no command is found that ends the tabular cell, we are not yet at the end of the cell. Redefine `\nprt@next` to start this command recursively for parsing the next token.

```
              \let\nprt@next\nprt@getnexttok
```

Test if this token is one of the allowed characters of a number.

```
              \nprt@IfCharInString{#1}{\nprt@allowedchars}{%
```

If yes, append this character to the token list of the number and set the flag that the number has been found.

```
                \nprt@numfoundtrue
                \nprt@digittoks=\expandafter{\the\nprt@digittoks#1}%
              }{%
```

If it is no character of a number, store it for the tokens before or after the number.

```
                \nprt@saveothertok{#1}%
              }%
            \fi % \relax
          \fi % \csname
        \fi % \endtabular
      \fi % \nprt@end
    \fi % \end
  \fi % \tabularnewline
```

Call the previously saved command (recursion or end of the tabular cell).

```
  \nprt@next
}
```

Boolean to decide if math mode is active outside the tabular cell. This is true for the `array` environment.

```
\newif\ifnprt@mathtabular
```

`\nprt@end` This macro is called at the end of each tabular cell. The arguments are used as follows: ⟨*digits before*⟩, ⟨*digits after*⟩ the decimal sign for the mantissa; ⟨*digits before*⟩, ⟨*digits after*⟩ the decimal sign for the exponent; commands inserted ⟨*before*⟩ and ⟨*after*⟩ the `\numprint` command. The arguments five and six are empty for printing the number in text mode and contain "$" both for printing the number in math mode.

```
\def\nprt@end#1#2#3#4#5#6{%
```

First, print the tokens before the number.

```
  \the\nprt@pretoks
```

Print the number in a group in order to save the tokens before and after the number to be influenced by its settings.

```
  \begingroup
```

Set the digits for the alignment of the mantissa.

```
%    \npnodigits
    \npdigits{#1}{#2}%
```

Set the digits for the alignment of the exponent if given. If no number of digits is given, they are set negativ.

```
    \npnoexponentdigits
    \ifnum#3<0
      \nprt@debug{no exponent alignment in tabular}%
    \else
      \ifnum#4<0
        \nprt@debug{exponent alignment in tabular with #3 digits}%
        \npexponentdigits{#3}%
      \else
        \nprt@debug{exponent alignment in tabular with #3.#4 digits}%
        \npexponentdigits[#4]{#3}%
      \fi
    \fi
```

Omit the section that prints the number if no number has been given.

```
    \ifnprt@numfound
```

Print the pre-command if defined. Since the pre-command is fixed to nothing or "$" to switch to math mode, this is only done if the math mode is not active, before. Set the boolean `\ifnprt@mathtabular` according to the previous status in order to close the math mode if necessary.

```
      \ifmmode
        \nprt@mathtabulartrue
      \else
        \nprt@mathtabularfalse
        #5%
      \fi
```

Print the number, with unit if one has been given.

```
      \ifx\nprt@unit\@empty
        \numprint{\the\nprt@digittoks}%
      \else
        \numprint[\nprt@unit]{\the\nprt@digittoks}%
      \fi
```

Switch off math mode if it has not been active before.

```
      \ifnprt@mathtabular
      \else
        #6%
      \fi
```

Print a message to the log file if no number has been specified.

```
    \else
      \PackageInfo{numprint}{No number in tabular cell}%
    \fi
```

Do the rest outside the group in order to prevent the post-texts from being formatted as the number.

```
  \endgroup
```

Print the tokens after the number.

```
  \the\nprt@posttoks
```

Print the contents defined with `\npafternum`.

```
  \ifx\nprt@afternum\@empty
  \else
    \nprt@afternum
  \fi
}
```

F.6.3 New column types

Define the new column types.

```
\ifnprt@newcolumntype
```

Declare a new column type `N` which prints a number in text mode and does not need to repeat `\numprint` in each tabular cell. This is declared empty because all executing macros are redefined anyway.

```
  \newcolumntype{N}{}
```

`\NC@rewrite@N` Redefine this command to parse the declaration of the `N` column type to look for an optional argument. If none is given, set both optional arguments to "`-1`".

```
  \def\NC@rewrite@N{%
    \nprt@digittoks{}%
    \nprt@pretoks{}%
```

If no optional argument is given, set both optional arguments to "`-1`".

```
    \@ifnextchar[{% ]
      \nprt@rewrite@{}{}%
    }{%
      \nprt@rewrite@@{}{}{-1}[-1]%
    }%
  }
```

Declare a new column type `n` which prints a number and does not need to repeat `\numprint` in each tabular cell. This is declared empty because all executing macros are redefined anyway.

```
\newcolumntype{n}{}
```

`\NC@rewrite@n` Same as `\NC@rewrite@N`, but give "$" twice for math mode.

```
\def\NC@rewrite@n{%
  \nprt@digittoks{}%
  \nprt@pretoks{}%
  \@ifnextchar[{% ]
    \nprt@rewrite@{$}{$}%
  }{%
    \nprt@rewrite@@{$}{$}{-1}[-1]%
  }%
}
```

`\nprt@rewrite@` Look for the second optional argument.

```
\def\nprt@rewrite@#1#2[#3]{%
  \@ifnextchar[{%]
    \nprt@rewrite@@{#1}{#2}{#3}%
  }{%
    \nprt@rewrite@@{#1}{#2}{#3}[-1]%
  }%
}
```

`\nprt@rewrite@@` The arguments are used as follows: commands inserted ⟨*before*⟩ and ⟨*after*⟩ the `\numprint` command; ⟨*digits before*⟩, ⟨*digits after*⟩ the decimal sign for the mantissa; ⟨*digits before*⟩, ⟨*digits after*⟩ the decimal sign for the exponent.

```
\def\nprt@rewrite@@#1#2#3[#4]#5#6{%
```

Add the definition for the current column to the already made column definitions that are stored in `\@temptokena`. Before the column itself, the starting command for the parsing, `\nprt@begin`, is inserted. At the end of the column, start the final work, command `\nprt@end`. These commands shall not be expanded, yet.

```
  \edef\nprt@rewrite@scratch{\the\@temptokena
    >{\noexpand\nprt@begin\noexpand\ignorespaces}l%
    <{\noexpand\nprt@end{#5}{#6}{#3}{#4}{#1}{#2}}%
  }%
```

Set `\@temptokena` to the preceding and the current column definition. This is implicitely used by the `array` package.

```
  \@temptokena\expandafter{\nprt@rewrite@scratch}%
```

Parse for next column in the definition.

```
  \NC@find
}
```

F.6.4 Old column types for compatibility

Define the old column types.

```
\else
```

The column type `n` aligns the base number but not the exponent.

```
  \newcolumntype{n}[2]{>{\npdigits{#1}{#2}$}l<{$}}
```

The column type `N` aligns the base number as well as the exponent.

```
\newcolumntype{N}[3]{%
  >{\npdigits{#1}{#2}\npexponentdigits{#3}$}l<{$}}
```

End of column type section.

```
\fi
```

F.7 Round numbers

Stores if the currenct digit has to be rounded up.

```
\newif\ifnprt@roundup
```

Declare the counter used inside `\nprt@round@after` and `\nprt@round@before`.

```
\newcount\nprt@thisdigit
```

`\nprt@round@after` Does the rounding of the number after the decimal sign. The first argument contains the current digit, the second the rest of the number.

This routine is a little bit complicated. It is called recursively. In the forward rund, it just counts the number of digits, it has parsed already. In the backward rund the rounding is done.

```
\def\nprt@round@after#1#2\@empty{%
```

Store the arguments in macros.

```
  \edef\nprt@argone{#1}%
  \edef\nprt@argtwo{#2}%
```

Count the parsed digits.

```
  \advance \nprt@curpos by 1
```

If the end of the number is reached an internal error has been occured since enough zeros are appended to the number before.

```
  \ifx\nprt@argone\@empty
    \nprt@error{Rounding: End of number has been reached}{This may
      not happen}%
  \else
```

If the current digit is one behind the last digit to be printed it decides if the last printed digit is rounded or not. This decision is stored in `\ifnprt@roundup`. If this position is reached, stop the recursion.

```
    \ifnum\nprt@curpos>\nprt@rndpos
      \ifnum\nprt@argone>4
        \nprt@rounduptrue
      \fi
    \else
```

The position has not been reached, yet. Thus, call this routine recursively.

```
      \expandafter\nprt@round@after#2\@empty\@empty
```

The following lines are executed in the backward run when rounding.

Store the current digit in a number in order to be able to calculate with it.

```
      \nprt@thisdigit=#1
```

If this number is to be rounded up do it by advancing it by one.

```
      \ifnprt@roundup
        \advance \nprt@thisdigit by 1
```

Reset `\ifnprt@roundup` since the preceeding digit is not to be rounded, normally.

```
        \nprt@roundupfalse
```

If the digit has been rounded from 9 to 10, it has to be a "0" and the preceeding digit has to be rounded up.

```
        \ifnum\nprt@thisdigit=10
          \nprt@thisdigit=0
          \nprt@rounduptrue
        \fi
      \fi
```

Store the modificated current digit to the new number by putting it into `\nprt@newnum` before all digits that have been stored before.

```
      \expandafter\xdef\expandafter\nprt@newnum{%
        \the\nprt@thisdigit\nprt@newnum}%
    \fi
  \fi
}
```

`\nprt@round@before` Does the rounding of the number after the decimal sign. The first argument contains the current digit, the second the rest of the number. This routine works as `\nprt@round@after` but stops at its end in contrast to a given number.

```
\def\nprt@round@before#1#2\@empty{%
```

Store the arguments.

```
  \edef\nprt@argone{#1}%
  \edef\nprt@argtwo{#2}%
```

Do the recursion until the end of the number. This routine does not have to decide wheather the number has to be rounded since it knows that by `\ifnprt@roundup`, set by `\nprt@round@after`.

```
  \ifx\nprt@argtwo\@empty
  \else
    \expandafter\nprt@round@before#2\@empty
  \fi
```

Store the current digit into a counter and use zero if the number before the decimal sign is empty.

```
  \ifx\nprt@argone\@empty
    \nprt@thisdigit=0
  \else
    \nprt@thisdigit=#1
  \fi
```

Add one to the number if it has to be rounded.

```
  \ifnprt@roundup
    \advance \nprt@thisdigit by 1
```

Reset the rounding of the next number.

```
    \nprt@roundupfalse
```

If rounded from 9 to 10, set this digit to "0" and give the information to the next digit.

```
    \ifnum\nprt@thisdigit=10
      \nprt@thisdigit=0
      \nprt@rounduptrue
```

```
    \fi
  \fi
```

If the number is empty, the new number has to be added and the counter adjusted.

```
  \ifx\nprt@argone\@empty
    \xdef\nprt@newnum{\the\nprt@thisdigit}%
    \stepcounter{nprt@\nprt@numname @digitsbefore}%
  \else
```

Insert the current digit before the already stored digits in `\nprt@newnum`.

```
    \expandafter\xdef\expandafter\nprt@newnum{%
      \the\nprt@thisdigit\nprt@newnum}%
  \fi
}
```

`\nprt@round` Round a number. The first argument is the Number type ("mantissa" resp. "exponent"), the second is the number of digits to be printed after the decimal sign.

```
\newcommand*\nprt@round[2]{%
  \begingroup
```

Store the Number type for use in `\nprt@round@before` and `\nprt@round@after`.

```
    \edef\nprt@numname{#1}%
```

If the number of printed digits after the decimal sign is negative, no rounding will be performed.

```
    \ifnum#2<0
    \else
```

Print a debug message.

```
      \nprt@debug{\string\nprt@round: Round after #2 digits for #1}%
```

Set the number of digits after the decimal sign to the given value since this number of digits will be printed later.

```
      \setcounter{nprt@#1@digitsafter}{#2}%
```

Append enough zeros to the after-decimal-sign part in order to have enough digits that `\nprt@round@after` will not reach the end of the number.

```
      \expandafter\g@addto@macro\csname nprt@#1@after\endcsname{%
        \nprt@roundnull}%
```

Two new counters for the round position and the current postion in `\nprt@round@after`.

```
      \newcount\nprt@curpos
      \newcount\nprt@rndpos
```

Set the number of digits after the decimal sign.

```
      \nprt@rndpos=#2
```

Default not to round.

```
      \nprt@roundupfalse
```

Define the "working" number for the subroutines.

```
      \edef\nprt@tmpnum{\csname nprt@#1@after\endcsname}%
```

The new number starts empty and will be filled by `\nprt@round@after`.

```
      \edef\nprt@newnum{}%
```

Do the rounding after the decimal sign.

```
      \expandafter\nprt@round@after\nprt@tmpnum\@empty\@empty
```

Copy the new number after the decimal sign to the "official" command storing it.

```
      \expandafter\xdef\csname nprt@#1@after\endcsname{\nprt@newnum}%
```

If the integer part has to be modificated, too, do it.

```
      \ifnprt@roundup
```

Copy the number to the working number.

```
        \edef\nprt@tmpnum{\csname nprt@#1@before\endcsname}%
```

Clear the new number.

```
        \edef\nprt@newnum{}%
```

Do the rounding before the decimal sign.

```
        \expandafter\nprt@round@before\nprt@tmpnum\@empty\@empty
```

If the first digit has been rounded up from 9 a new digit "1" has to be inserted before the number.

```
        \ifnprt@roundup
          \expandafter\xdef\expandafter\nprt@newnum{1\nprt@newnum}%
          \stepcounter{nprt@#1@digitsbefore}%
        \fi
```

Copy the new number before the decimal sign to the "official" command storing it.

```
        \expandafter\xdef\csname nprt@#1@before\endcsname{\nprt@newnum}%
      \fi
    \fi
  \endgroup
```

Set the boolean for a found decimal sign according to the number of printed decimals.

```
  \ifnum#2<0
  \else
```

If rounded to no digits after the decimal sign, switch off printing of it.

```
    \ifnum#2=0
      \csname nprt@#1@decimalfoundfalse\endcsname
    \else
```

If one or more digits are printed, a decimal sign has to be printed.

```
      \csname nprt@#1@decimalfoundtrue\endcsname
    \fi
  \fi
}%
```

`\nprt@lpad` Pad a number with a character on the left side. The first argument contains the number, the second is the desired length and the third the padding character.

```
\newcommand*\nprt@lpad[3]{%
```

If the count of digits is negative, no padding will be performed.

```
  \ifnum#2<0
  \else
```

Print a debug message.

```
    \nprt@debug{\string\nprt@lpad: Padding #1 with #3 to a length of #2}%
```

If the number has less than the desired length, add the padding character to the left and call this function recursively.

```
    \ifnum\csname thenprt@#1@digitsbefore\endcsname<#2
      \expandafter\xdef\csname nprt@#1@before\endcsname{%
        #3\csname nprt@#1@before\endcsname}%
      \stepcounter{nprt@#1@digitsbefore}%
      \nprt@lpad{#1}{#2}{#3}%
    \fi
  \fi
}%
```

F.8 Print the numbers

`\nprt@sign@+` `\nprt@sign@-` `\nprt@sign@+-`

Define commands for printing the signs in math mode. This ensures that the printed signs really are signs and not hyphens. Compare "-" to "−".

```
\expandafter\newcommand\csname nprt@sign@+\endcsname{\ensuremath{+}}
\expandafter\newcommand\csname nprt@sign@-\endcsname{\ensuremath{-}}
\expandafter\newcommand\csname nprt@sign@+-\endcsname{\ensuremath{\pm}}
```

`\nprt@printsign`

Print a sign. The first argument contains the number type ("mantissa" or "exponent"). The second argument contains the sign as in source code.

```
\newcommand*\nprt@printsign[2]{%
```

Write sign to log file.

```
  \nprt@debug{\string\nprt@printsign: `#2'}%
```

Set command in a group to prevent the defined macros from being global.

```
    \edef\nprt@marg{#2}%
```

If a plus is to be added do it if no sign given.

```
    \csname ifnprt@addplus@#1\endcsname
      \ifx\nprt@marg\@empty
        \edef\nprt@marg{+}%
      \fi
    \fi
```

Do nothing if still no sign given.

```
    \ifx\nprt@marg\@empty
    \else
```

If a macro `\nprt@sign@`⟨*sign*⟩ is defined for the given sign, e.g., `\nprt@sign@+`, print it; if not, print the sign itself.

```
      \@ifundefined{nprt@sign@\nprt@marg}{%
        \PackageWarning{numprint}{%
          Unknown sign `\nprt@marg'. Print as typed in}%
        \nprt@marg
      }{%
        {\csname nprt@sign@\nprt@marg\endcsname}%
      }%
    \fi
}
```

Internal counters for printing.

```
\newcounter{nprt@digitsfirstblock}
\newcounter{nprt@blockcnt}
```

Internal boolean.

```
\newif\ifnprt@shortnumber
```

`\nprt@printbefore` Print the number before the decimal sign. The argument is the Number type ("mantissa" or "exponent"). When this macro is called, everything is parsed already. Thus, it is known wheather a decimal sign has been found, which and how many digits are before resp. after the decimal sign etc.

```
\newcommand*\nprt@printbefore[1]{%
```

If missing zeros shall be added and there is no digit before the decimal sign store that "0" into the corresponding command and store in the counter that the number of digits before the decimal sign is one, know.

```
  \ifnprt@addmissingzero
    \ifnum\csname thenprt@#1@digitsbefore\endcsname=0
      \expandafter\edef\csname nprt@#1@before\endcsname{0}%
      \stepcounter{nprt@#1@digitsbefore}%
    \fi
  \fi
```

I'm not sure why I have added the group here. But it works and I won't change it, therefore.

```
  \begingroup
```

Store the number to be printed in `\nprt@numbertoprint` in order to have simpler calls in this routine than using `\csname`...

```
    \edef\nprt@numbertoprint{\csname nprt@#1@before\endcsname}%
```

If four-digit numbers are not to be separated and both, the integer and the real parts, are shorter than 5 digits, set the boolean `\ifnprt@shortnumber` to "true" that the number is printed without separators, later.

```
    \ifnprt@numsepfour
    \else
      \ifnum\csname thenprt@#1@digitsbefore\endcsname<5
        \ifnum\csname thenprt@#1@digitsafter\endcsname<5
          \nprt@shortnumbertrue
        \fi
      \fi
    \fi
```

If the number is short according to the preceeding code, just print that number by calling `\nprt@numbertoprint`.

```
    \ifnprt@shortnumber
      \nprt@numbertoprint
    \else
```

If the number will get separators, calculate how many separators will be inserted.

```
      % ganze Bloecke
      \setcounter{nprt@blockcnt}{%
        \csname thenprt@#1@digitsbefore\endcsname}%
      \addtocounter{nprt@blockcnt}{-1}%
      \divide\c@nprt@blockcnt 3%
```

Then, calculate how many digits are in the first block (one, two, or three). Use `\c@nprt@cntprint` as temporary variable.

```
      \setcounter{nprt@digitsfirstblock}{%
```

```
      \csname thenprt@#1@digitsbefore\endcsname}%
    \setcounter{nprt@cntprint}{\thenprt@blockcnt}%
    \multiply \c@nprt@cntprint 3%
    \addtocounter{nprt@digitsfirstblock}{-\thenprt@cntprint}%
```

Depending on that number, call `\nprt@printone`, `\nprt@printtwo`, resp. `\nprt@printthree` which do what you may expect with that names.

```
    \ifnum\thenprt@digitsfirstblock=1
      \expandafter\nprt@printone\nprt@numbertoprint\@empty
    \else
      \ifnum\thenprt@digitsfirstblock=2
        \expandafter\nprt@printtwo\nprt@numbertoprint\@empty
      \else
        \ifnum\thenprt@digitsfirstblock=3
          \expandafter\nprt@printthree\nprt@numbertoprint\@empty
        \else
          \ifnum\thenprt@digitsfirstblock=0
          \else
            \PackageError{numprint}{internal error}{}%
          \fi
        \fi
      \fi
    \fi
  \fi
  \endgroup
}
```

`\nprt@printthree` Print three digits. If the command has not reached the end of the string, print a separator `\nprt@separator@before` and call this routine recursively.

```
\def\nprt@printthree#1#2#3#4\@empty{%
  #1#2#3%
  \def\nprt@tmp{#4}%
  \ifx\nprt@tmp\empty
  \else
    \nprt@separator@before%
    \nprt@printthree#4\@empty\@empty\@empty
  \fi
}
```

`\nprt@printtwo` The same but start with two instead of three digits.

```
\def\nprt@printtwo#1#2#3\@empty{%
  #1#2%
  \def\nprt@tmp{#3}%
  \ifx\nprt@tmp\empty
  \else
    \nprt@separator@before%
    \nprt@printthree#3\@empty\@empty\@empty
  \fi
}
```

`\nprt@printone` The same but start with one instead of three digits.

```
\def\nprt@printone#1#2\@empty{%
  #1%
  \def\nprt@tmp{#2}%
```

```
  \ifx\nprt@tmp\empty
  \else
    \nprt@separator@before%
    \nprt@printthree#2\@empty\@empty\@empty
  \fi
}
```

`\nprt@printafter` Print the number after the decimal sign. The argument is the Number type ("mantissa" or "exponent"). This macro works similarly as `\nprt@printbefore`.

```
\newcommand*\nprt@printafter[1]{%
```

If a missing zero shall be added do it if no digits are given after the decimal sign if a decimal sign has been given. If no decimal sign has been given, the number is pure integer and does not get a real part.

```
  \csname ifnprt@#1@decimalfound\endcsname
    \ifnprt@addmissingzero
      \ifnum\csname thenprt@#1@digitsafter\endcsname=0
        \expandafter\edef\csname nprt@#1@after\endcsname{0}%
        \stepcounter{nprt@#1@digitsafter}%
      \fi
    \fi
```

If a after-decimal zero will be replaced by another command but the real part is empty, put a "0" after the comma (same as "addmissingzero", but here it is just a hack in order to enable the replacement command to take effect later).

```
    \ifx\nprt@replacenull\@empty
    \else
      \ifnum\csname thenprt@#1@digitsafter\endcsname=0
        \expandafter\edef\csname nprt@#1@after\endcsname{0}%
        \stepcounter{nprt@#1@digitsafter}%
      \fi
    \fi
  \fi
```

Store the number in `\nprt@numbertoprint` and continue only if it is not empty.

```
  \begingroup
    \edef\nprt@numbertoprint{\csname nprt@#1@after\endcsname}%
    \ifx\nprt@numbertoprint\@empty
    \else
```

Find out wheather separators have to be inserted.

```
      \ifnprt@numsepfour
      \else
        \ifnum\csname thenprt@#1@digitsbefore\endcsname<5
          \ifnum\csname thenprt@#1@digitsafter\endcsname<5
            \nprt@shortnumbertrue
          \fi
        \fi
      \fi
```

If a zero has to be replaced by a replacement text, and if the after comma part has the numerical value "0" (= it contains of zeros only), do the replacement.

```
      \ifx\nprt@replacenull\@empty
      \else
        \ifnum\nprt@numbertoprint=0
```

```
737           \nprt@shortnumbertrue
738           \edef\nprt@numbertoprint{\nprt@replacenull}%
739         \fi
740       \fi
```

If the number is short (without separators) just print it.

```
741       \ifnprt@shortnumber
742         \nprt@numbertoprint
743       \else
```

Print the number with separators. The choice between different block sizes does not have to be done because the after-decimal-sign part starts with three-digit block from the left end.

```
744         \expandafter\nprt@printthree@after%
745           \nprt@numbertoprint\@empty\@empty\@empty
746       \fi
747     \fi
748   \endgroup
749 }
```

`\nprt@printthree@after` The same as `\nprt@printthree` but with another separator.

```
750 \def\nprt@printthree@after#1#2#3#4\@empty{%
751   #1#2#3%
752   \def\nprt@tmp{#4}%
753   \ifx\nprt@tmp\empty
754   \else
755     \nprt@separator@after
756     \nprt@printthree@after#4\@empty\@empty\@empty
757   \fi
758 }
```

F.9 The main command

`\numprint` The main macro of the package. The mandatory argument takes a number and prints it as described above. The optional argument may contain a unit which then is printed, too.

```
759 \DeclareRobustCommand*\numprint[2][\@empty]{%
```

Switch off the error flag. This should not be necessary but is done for stability reasons.

```
760   \nprt@argumenterrorfalse
```

Clear the mantissa and the exponent.

```
761   \xdef\nprt@exponent{\@empty}%
762   \xdef\nprt@mantissa{\@empty}%
```

Do everything inside a group to avoid defining too many temporary macros that are not deleted after the macro.

```
763   \begingroup
```

Store the mandatory argument into a macro. Redefine `\,` and `~` to do nothing as to ignore them. Because the argument is expanded this does not work with the ignore list for characters. This again has to be done inside a group to preserve the two macros for later usage.

```
764     \begingroup
```

```
      \def\,{}%
      \catcode`\~=\active % tilde is active
      \def~{}%
      \xdef\nprt@marg{#2}%
    \endgroup
```

Don't expand the unit because that may cause trouble.

```
    \def\nprt@oarg{#1}%
```

Declare some commands to detect empty arguments.

```
    \def\nprt@@empty{\@empty}%
    \def\nprt@nix{}%
    \def\nprt@nixleer{ }%
```

Some debug information.

```
    \ifx\nprt@oarg\nprt@@empty
      \nprt@debug{\string\numprint{\protect#2}}%
    \else
      \nprt@debug{\string\numprint[\protect#1]{\protect#2}}%
    \fi
```

Test for an empty mandatory argument.

```
    \ifx\nprt@marg\nprt@nix
      \nprt@error{empty argument}{You have to specify a number in
        the argument of \string\numprint}%
    \fi
    \ifx\nprt@marg\nprt@nixleer
      \nprt@error{empty argument}{You have to specify a number in
        the argument of \string\numprint}%
    \fi
```

Test wheather only valid characters have been used and devide the argument in the mantissa and the exponent.

```
    \expandafter\nprt@testcharacter\nprt@marg\@empty\@empty
```

If there are invalid characters in the argument, just print the argument without formatting it. Redefine `\pm` to avoid an additional error in text mode.

```
    \ifnprt@argumenterror
      \begingroup\def\pm{+-}#2\endgroup
    \else
```

If everything is okay, proceed with parsing.

If the mantissa is empty don't work on it but reset the sign of the mantissa to avoid to reprint the old sign when using `\numprint{e123}`. If it is not empty, parse it for a sign and a number.

```
      \ifx\nprt@mantissa\@empty
        \def\nprt@mantissa@sign{\@empty}%
      \else
        \nprt@testsign{mantissa}{\nprt@mantissa}%
      \fi
```

If the mantissa contains only a sign, `\nprt@mantissa` is set to `\@empty` and thus empty even if it wasn't 4 lines above. If the mantissa only contains of a sign and an exponent is given, everything is fine. If no exponent is given, the input format is invalid.

```
      \ifx\nprt@mantissa\@empty
```

```
      \ifx\nprt@exponent\@empty
        \nprt@error{Invalid number format. Printing the
          argument\MessageBreak
          verbatim}{Something is wrong in the format of the number}%
      \else
        \nprt@printsign{mantissa}{\nprt@mantissa@sign}%
      \fi
    \else
```

Round the mantissa if necessary.

```
      \nprt@round{mantissa}{\nprt@rounddigits}%
```

Pad the mantissa if necessary.

```
      \nprt@lpad{mantissa}{\nprt@lpaddigits}{\nprt@lpadchar}%
    \fi
```

If an exponent has been found, parse this like the mantissa.

```
    \ifnprt@expfound
```

Test wheather an exponent character was given but no exponent.

```
      \ifx\nprt@exponent\@empty
        \nprt@error{Empty exponent}{If you specify an exponent
          using one of the characters `\nprt@explist' you
          have\MessageBreak
          to give an exponent, too.}%
      \else
        \nprt@testsign{exponent}{\nprt@exponent}%
        \nprt@round{exponent}{\nprt@roundexpdigits}%
      \fi
    \fi
```

If either the mantissa or the exponent produces an error, just print the argument as is. Redefine `\pm` to avoid an additional error in text mode.

```
    \ifnprt@argumenterror
      \begingroup\def\pm{+-}#2\endgroup
    \else
```

Print the mantissa if present.

```
      \ifx\nprt@mantissa\@empty
      \else
```

Print the part before the decimal sign.

If the number shall be printed in a box (table alignment) some special things have to be done.

```
        \ifnprt@mantissa@fixeddigits
          \ifmmode
```

In math mode, it must be decided which math style is active because the width of the box depends on that. The `\mathchoice` command does the formatting for all styles and typesets the active one then.

```
            \mathchoice{%
```

Calculate the width of the block for the `\displaystyle`.

```
              \nprt@calcblockwidth{mantissa}{before}{\displaystyle}%
```

Generate a box with the calculated width and the corresponding math style.

```
              \makebox[\the\nprt@blockwidth][r]{$\displaystyle
```

Print the sign if present.

```
                \nprt@printsign{mantissa}{\nprt@mantissa@sign}%
```

Print the integer part into the box.

```
                \nprt@printbefore{mantissa}$}%
              }{%
```

Do the same for `\textstyle`.

```
                \nprt@calcblockwidth{mantissa}{before}{\textstyle}%
                \makebox[\the\nprt@blockwidth][r]{$\textstyle
                  \nprt@printsign{mantissa}{\nprt@mantissa@sign}%
                  \nprt@printbefore{mantissa}$}%
              }{%
```

Do the same for `\scriptstyle`.

```
                \nprt@calcblockwidth{mantissa}{before}{\scriptstyle}%
                \makebox[\the\nprt@blockwidth][r]{$\scriptstyle
                  \nprt@printsign{mantissa}{\nprt@mantissa@sign}%
                  \nprt@printbefore{mantissa}$}%
              }{%
```

Do the same for `\scriptscriptstyle`.

```
                \nprt@calcblockwidth{mantissa}{before}{\scriptscriptstyle}%
                \makebox[\the\nprt@blockwidth][r]{$\scriptscriptstyle
                  \nprt@printsign{mantissa}{\nprt@mantissa@sign}%
                  \nprt@printbefore{mantissa}$}%
              }%
            \else
```

If the number is printed in text mode, the size is preserved inside the box. Thus, no hack as for math mode is necessary.

```
              \nprt@calcblockwidth{mantissa}{before}{\@empty}%
              \makebox[\the\nprt@blockwidth][r]{%
                \nprt@printsign{mantissa}{\nprt@mantissa@sign}%
                \nprt@printbefore{mantissa}%
              }%
            \fi
          \else
```

If the number is printed without fixed width, just print it.

```
            \nprt@printsign{mantissa}{\nprt@mantissa@sign}%
            \nprt@printbefore{mantissa}%
          \fi
```

Print the after-decimal-sign digits. This works exactly as the integer part.

```
          \ifnprt@mantissa@fixeddigits
            \ifmmode
              \mathchoice{%
                \nprt@calcblockwidth{mantissa}{after}{\displaystyle}%
                \makebox[\the\nprt@blockwidth][l]{$\displaystyle
                  \ifnprt@mantissa@decimalfound
                    \nprt@decimal
                  \fi
                  \nprt@printafter{mantissa}$}%
              }{%
                \nprt@calcblockwidth{mantissa}{after}{\textstyle}%
```

```
              \makebox[\the\nprt@blockwidth][l]{$\textstyle
                \ifnprt@mantissa@decimalfound
                  \nprt@decimal
                \fi
                \nprt@printafter{mantissa}$}%
            }{%
              \nprt@calcblockwidth{mantissa}{after}{\scriptstyle}%
              \makebox[\the\nprt@blockwidth][l]{$\scriptstyle
                \ifnprt@mantissa@decimalfound
                  \nprt@decimal
                \fi
                \nprt@printafter{mantissa}$}%
            }{%
              \nprt@calcblockwidth{mantissa}{after}{\scriptscriptstyle}%
              \makebox[\the\nprt@blockwidth][l]{$\scriptscriptstyle
                \ifnprt@mantissa@decimalfound
                  \nprt@decimal
                \fi
                \nprt@printafter{mantissa}$}%
            }%
          \else
            \nprt@calcblockwidth{mantissa}{after}{\@empty}%
            \makebox[\the\nprt@blockwidth][l]{%
              \ifnprt@mantissa@decimalfound
                \nprt@decimal
              \fi
              \nprt@printafter{mantissa}%
            }%
          \fi
        \else
          \ifnprt@mantissa@decimalfound
            \nprt@decimal
          \fi
          \nprt@printafter{mantissa}%
        \fi
      \fi
```

If an exponent is defined it has to be printed later. Therefore, define the command `\nprt@printexp` that just prints its argument.

```
      \ifnprt@expfound
        \def\nprt@printexp##1{##1}%
```

If a sign but no number has been given for the exponent `\nprt@exponent` is empty but `\ifnprt@expfound` true. Test for this case here.

```
        \ifx\nprt@exponent\@empty
          \nprt@error{Invalid number format in
            exponent. Printing\MessageBreak
            garbage}{Something is wrong in the format of the exponent}%
        \fi
      \else
```

If there is no exponent but digits reserved for it, define the `\nprt@printexp` so that is just needs that space but does not print anything.

```
        \ifnprt@exponent@fixeddigits
          \def\nprt@printexp##1{\hphantom{##1}}%
```

```
      \else
```

If there is no exponent and no reserved digits, don't print the exponent.

```
        \def\nprt@printexp##1{}%
      \fi
    \fi
```

If negative numbers are printed in red (as shown in section 9.2), a positive mantissa with a negative exponent would normally lead to a black mantissa with a red exponent. To avoid that the `\color` command is redefined to do nothing here. Do it in a group to restrain this to the exponent itself.

```
    \begingroup
    \def\color##1{}%
```

Call the command that either print the exponent, reserves space as it were printed or does nothing.

```
    \nprt@printexp{%
```

If not mantissa is specified, print the short version, e.g., 10^{123} and thus leave out the product sign.

```
      \ifx\nprt@mantissa\@empty
      \else
```

Print out the product sign, if there is a mantissa.

```
        \nprt@prod
      \fi
```

Print "10" and the exponent. This is different between text and math mode.

```
      \ifmmode 10^\else 10\expandafter\textsuperscript\fi{%
```

Print the number before the decimal sign. As the mantissa.

```
        \ifnprt@exponent@fixeddigits
          \ifmmode
            \mathchoice{%
              \nprt@calcblockwidth{exponent}{before}{\displaystyle}%
              \makebox[\the\nprt@blockwidth][r]{$\displaystyle
                \nprt@printsign{exponent}{\nprt@exponent@sign}%
                \nprt@printbefore{exponent}$}%
            }{%
              \nprt@calcblockwidth{exponent}{before}{\textstyle}%
              \makebox[\the\nprt@blockwidth][r]{$\textstyle
                \nprt@printsign{exponent}{\nprt@exponent@sign}%
                \nprt@printbefore{exponent}$}%
            }{%
              \nprt@calcblockwidth{exponent}{before}{\scriptstyle}%
              \makebox[\the\nprt@blockwidth][r]{$\scriptstyle
                \nprt@printsign{exponent}{\nprt@exponent@sign}%
                \nprt@printbefore{exponent}$}%
            }{%
              \nprt@calcblockwidth{exponent}{before}{\scriptscriptstyle}%
              \makebox[\the\nprt@blockwidth][r]{$\scriptscriptstyle
                \nprt@printsign{exponent}{\nprt@exponent@sign}%
                \nprt@printbefore{exponent}$}%
            }%
          \else
            \nprt@calcblockwidth{exponent}{before}{\@empty}%
```

```
      \makebox[\the\nprt@blockwidth][r]{%
        \nprt@printsign{exponent}{\nprt@exponent@sign}%
        \nprt@printbefore{exponent}%
      }%
    \fi
  \else
    \nprt@printsign{exponent}{\nprt@exponent@sign}%
    \nprt@printbefore{exponent}%
  \fi
```

Produce a warning since this is uncommon in exponents.

```
  \ifnprt@exponent@decimalfound
    \PackageWarning{numprint}{Non-integer exponent}%
  \fi
```

Print the part after the decimal sign.

```
  \ifnprt@exponent@fixeddigits
    \ifmmode
      \mathchoice{%
        \nprt@calcblockwidth{exponent}{after}{\displaystyle}%
        \makebox[\the\nprt@blockwidth][l]{$\displaystyle
          \ifnprt@exponent@decimalfound
            \nprt@decimal
          \fi
          \nprt@printafter{exponent}$}%
      }{%
        \nprt@calcblockwidth{exponent}{after}{\textstyle}%
        \makebox[\the\nprt@blockwidth][l]{$\textstyle
          \ifnprt@exponent@decimalfound
            \nprt@decimal
          \fi
          \nprt@printafter{exponent}$}%
      }{%
        \nprt@calcblockwidth{exponent}{after}{\scriptstyle}%
        \makebox[\the\nprt@blockwidth][l]{$\scriptstyle
          \ifnprt@exponent@decimalfound
            \nprt@decimal
          \fi
          \nprt@printafter{exponent}$}%
      }{%
        \nprt@calcblockwidth{exponent}{after}{\scriptscriptstyle}%
        \makebox[\the\nprt@blockwidth][l]{$\scriptscriptstyle
          \ifnprt@exponent@decimalfound
            \nprt@decimal
          \fi
          \nprt@printafter{exponent}$}%
      }%
    \else
      \nprt@calcblockwidth{exponent}{after}{\@empty}%
      \makebox[\the\nprt@blockwidth][l]{%
        \ifnprt@exponent@decimalfound
          \nprt@decimal
        \fi
        \nprt@printafter{exponent}%
      }%
```

```
            \fi
          \else
            \ifnprt@exponent@decimalfound
              \nprt@decimal
            \fi
            \nprt@printafter{exponent}%
          \fi
        }% 10^
      }% \nprt@printexp
      \endgroup
    \fi
  \fi
```

If the unit is not empty, print it, too.

```
  \ifx\nprt@oarg\nprt@@empty
  \else
```

All units expect the degree symbol are separated from the number. Detect wheather the degree symbol is used.

The `\textdegree` command is not useable in math mode. Redeclare it to generate a warning and to use `\tcdegree` instead.

```
    \def\textdegree{%
      \PackageWarning{numprint}{The unit is typeset in mathmode. Use
        \string\tcdegree\space of\MessageBreak
        the mathcomp package or \string\degree\space of the
        gensymb\MessageBreak
        package}%
```

Call `\tcdegree` via `\csname` in order to avoid a second error message if `\tcdegree` is not present (because of not loading `mathcomp.sty`.

```
      \csname tcdegree\endcsname
    }%
```

Handle `\textcelsius`, `\textohm`, `\textmu`, and `\textperthousand` in the same way as `\textdegree`.

```
    \def\nprt@PackageError##1##2{\PackageWarning{numprint}{##1}}%
    \def\textcelsius{%
      \begingroup
      \@ifundefined{tccelsius}{\@ifundefined{celsius}{%
          \def\nprt@PackageError####1####2{%
            \PackageError{numprint}{####1}{####2}}%
        }{\celsius}}{\tccelsius}%
      \nprt@PackageError{The unit is typeset in
        mathmode. Use \string\tccelsius\space of\MessageBreak
        the mathcomp package or \string\celsius\space of the
        gensymb\MessageBreak
        package}{If you load the mathcomp package
        \string\textcelsius\space is substituted by
        \string\tccelsius. If you load the gensymb package
        \string\textcelsius\space is substituted by \string\celsius.}%
      \endgroup
    }%
    \def\textohm{%
      \begingroup
      \@ifundefined{tcohm}{\@ifundefined{ohm}{%
```

```
            \def\nprt@PackageError####1####2{%
              \PackageError{numprint}{####1}{####2}}%
          }{\ohm}}{\tcohm}%
        \nprt@PackageError{The unit is typeset in
          mathmode. Use \string\tcohm\space of\MessageBreak
          the mathcomp package or \string\ohm\space of the
          gensymb\MessageBreak
          package}{If you load the mathcomp package
          \string\textohm\space is substituted by
          \string\tcohm. If you load the gensymb package
          \string\textohm\space is substituted by \string\ohm.}%
        \endgroup
      }%
      \def\textmu{%
        \begingroup
        \@ifundefined{tcmu}{%
          \def\nprt@PackageError####1####2{%
            \PackageError{numprint}{####1}{####2}}%
        }{\tcmu}%
        \nprt@PackageError{The unit is typeset in
          mathmode. Use \string\tcmu\space of\MessageBreak
          the mathcomp package}{If you load the mathcomp package
          \string\textmu\space is substituted by \string\tcmu.}%
        \endgroup
      }%
      \def\textperthousand{%
        \begingroup
        \@ifundefined{tcperthousand}{\@ifundefined{perthousand}{%
            \def\nprt@PackageError####1####2{%
              \PackageError{numprint}{####1}{####2}}%
          }{\perthousand}}{\tcperthousand}%
        \nprt@PackageError{The unit is typeset in
          mathmode. Use \string\tcperthousand\MessageBreak
          of the mathcomp package or \string\perthousand\space of
          the\MessageBreak
          gensymb package}{If you load the mathcomp package
          \string\textperthousand\space is substituted by
          \string\tcperthousand. If you load the gensymb package
          \string\textperthousand\space is substituted by
          \string\perthousand.}%
        \endgroup
      }%
```

If the unit is `\tcdegree` from the `mathcomp` package, the unit is a degree sign. Then print the separator `\nprt@degreesep` instead of `\nprt@unitsep`.

```
      \def\nprt@tmpunit{\tcdegree}%
      \ifx\nprt@oarg\nprt@tmpunit
        \ensuremath{\nprt@degreesep}%
      \else
```

If the unit is `\degree` from the `gensymb` package, the unit is a degree sign. Then print the separator `\nprt@degreesep` instead of `\nprt@unitsep`.

```
        \def\nprt@tmpunit{\degree}%
        \ifx\nprt@oarg\nprt@tmpunit
          \ensuremath{\nprt@degreesep}%
```

```
        \else
```

If the unit is `\textdegree` from the `textcomp` package, the unit is a degree sign. Then print the separator `\nprt@degreesep` instead of `\nprt@unitsep`.

```
          \def\nprt@tmpunit{\textdegree}%
          \ifx\nprt@oarg\nprt@tmpunit
            \ensuremath{\nprt@degreesep}%
          \else
```

If the unit is `\%`, the unit is a percent sign. Then print the separator `\nprt@percentsep` instead of `\nprt@unitsep`.

```
            \def\nprt@tmpunit{\%}%
            \ifx\nprt@oarg\nprt@tmpunit
              \ensuremath{\nprt@percentsep}%
            \else
```

Else, print `\nprt@unitsep`.

```
              \ensuremath{\nprt@unitsep}%
            \fi
          \fi
        \fi
      \fi
```

Finally, print the unit.

```
      \npunitcommand{\nprt@oarg}%
    \fi
  \endgroup
}
```

F.10 Print lengths and counters

`\cntprint` The macro for printing counters. The mandatory argument takes a LaTeX counter. The optional argument may contain a unit which is printed, too.

```
\newcounter{nprt@cntprint}
\DeclareRobustCommand*\cntprint[2][\@empty]{%
  \@ifundefined{c@#2}{\@nocounterr{#2}}{%
    \numprint[#1]{\arabic{#2}}%
  }%
}
```

`\lenprint` The macro for printing lengths. The mandatory argument takes a TeX dimension or a counter. The optional argument may contain a unit that is used to convert the unit from `pt` to the given value and which then is printed, too.

```
\DeclareRobustCommand*\lenprint[2][\@empty]{%
```

Do everything in a group to avoid macros to resist outside.

```
  \begingroup
```

Save the optional argument in `\nprt@oarg` and expand it.

```
    \edef\nprt@oarg{#1}%
```

If no optional argument has been given, change it to `pt`. This cannot be done as default argument in the `\DeclareRobustCommand*` definition since printing of counters does not have `pt` as default.

```
    \edef\nprt@tmp{\@empty}%
```

```
    \ifx\nprt@oarg\nprt@tmp
      \def\nprt@oarg{pt}%
    \fi
```

If the chosen unit is not defined produce an error message and fall back to `pt`.

```
    \@ifundefined{nprt@factor@\nprt@oarg}{%
      \PackageError{numprint}{Invalid unit `#1'}{%
        The requested unit is not defined using \string\npdefunit.^^J%
        Using `pt' instead.}%
      \def\nprt@oarg{pt}%
    }{}%
```

For later printing, a length is used that is fixed to `pt` as length. If another unit has been chosen a fake length has to be calculated that is as many `pt` long as the original length is in the given unit. Thus, multiply the mandatory argument by the specific factor for the given unit.

```
    \setlength{\@tempdima}{#2}%
    \setlength{\@tempdima}{\csname nprt@factor@\nprt@oarg\endcsname\@tempdima}%
```

If the unit name for the given unit is not defined but the factor is an internal error has occured. Produce an error message.

```
    \@ifundefined{nprt@unit@\nprt@oarg}{%
      \PackageError{numprint}{Unknown unit name `\nprt@oarg'}{%
        Send a bug report to harald.harders@tu-bs.de with a short
        example showing this bug.}%
    }{%
```

Store the unit name in `\nprt@oarg`.

```
      \edef\nprt@oarg{\csname nprt@unit@\nprt@oarg\endcsname}%
    }%
```

Finally, call `\numprint`. The two `\expandafter` are necessary since `\numprint` also uses `\nprt@oarg` and produces an infinite loop otherwise.

```
    \expandafter\numprint\expandafter[\nprt@oarg]{\strip@pt\@tempdima}%
  \endgroup
}
```

`\npdefunit` Define a new unit for usage with `\lenprint`. First argument: name in the source-code. Second argument: printed unit. Third argument: factor from `pt` to the new unit or `*` to preserve old factor.

```
\newcommand*\npdefunit[3]{%
```

If the third argument is not `*` define the factor for this unit in `\nprt@factor@`⟨#1⟩.

```
  \if#3*
  \else
    \expandafter\def\csname nprt@factor@#1\endcsname{#3}%
  \fi
```

Define the unit name for this unit in `\nprt@unit@`⟨#1⟩.

```
  \expandafter\def\csname nprt@unit@#1\endcsname{#2}%
}
```

Declare the standard units. `\!` is used to delete the unit separator again. This is a hack and does not work in all cases correctly.

```
\npdefunit{pt}{pt}{1.00000000000}
\npdefunit{bp}{bp}{0.99626400996}
```

```
\npdefunit{in}{in}{0.01383700013}
\npdefunit{ft}{ft}{0.00115308334}
\npdefunit{mm}{mm}{0.35145980351}
\npdefunit{cm}{cm}{0.03514598035}
\npdefunit{m}{m}{0.0003514598035}
\npdefunit{km}{km}{0.0000003514598035}
```

F.11 Internationalization

`\nprt@ifundefined` The normal `\@ifundefined` command defines the tested macro as side effect. This command is better. Sometimes it will hopefully be unnecessary and supported by the standard (with a different name, of course).

```
\newcommand*\nprt@ifundefined[1]{%
  \begingroup\expandafter\expandafter\expandafter\endgroup
  \expandafter\ifx\csname #1\endcsname\relax
    \expandafter\@firstoftwo
  \else
    \expandafter\@secondoftwo
  \fi
}
```

`\nprt@addto` Adds the code of the second argument to the macro defined in the first argument. This argument has to be specified without the preceeding backslash. The code is added only if the command is defined, already.

```
\newcommand\nprt@addto[2]{%
  \expandafter\nprt@ifundefined{#1}{}{%
    \expandafter\addto\expandafter{\csname #1\endcsname}{#2}%
  }%
}
```

`\npaddtolanguage` Define `\npaddtolanguage{`⟨*Language*⟩`}{`⟨*Language definitions*⟩`}`. Adds the ⟨*Language definitions*⟩ specified in argument two to the ⟨*Language*⟩ of argument one. The language definitions have to be provided in a command with the name `\npstyle`⟨*Language definitions*⟩. The switching-on command is added to `\extras`⟨*Language*⟩ while the switching-off command `\npstyledefault` is added to `\noextras`⟨*Language*⟩.

For example, `\npaddtolanguage{UKenglish}{english}` adds the command `\npstyleenglish` to `\extrasUKenglish` and `\npstyledefault` to `\noextrasUKenglish`.

The `\npstyle`⟨*Language definitions*⟩ commands are described below.

```
\newcommand\npaddtolanguage[2]{%
  \nprt@addto{extras#1}{\csname npstyle#2\endcsname}%
  \nprt@addto{noextras#1}{\npstyledefault}%
}
```

`\npstyledefault` Set the default values for the separators. They are set to the German language because of compatibility with older versions.

```
\newcommand*\npstyledefault{%
  \npthousandsep{\,}%
  \npdecimalsign{,}%
  \npproductsign{\cdot}%
  \npunitseparator{\,}%
  \npdegreeseparator{}%
  \nppercentseparator{\nprt@unitsep}%
}
```

Set the default settings (that are actually the same as German).

```
    \npstyledefault
```

`\npstylegerman` Sets the parameters to German habit.

```
\newcommand*\npstylegerman{%
  \npthousandsep{\,}%
  \npdecimalsign{,}%
  \npproductsign{\cdot}%
  \npunitseparator{\,}%
  \npdegreeseparator{}%
  \nppercentseparator{\nprt@unitsep}%
}
```

`\npstyleenglish` Sets the parameters to English habit.

```
\newcommand*\npstyleenglish{%
  \npthousandsep{,}%
  \npdecimalsign{.}%
  \npproductsign{\times}%
  \npunitseparator{\,}%
  \npdegreeseparator{}%
  \nppercentseparator{\nprt@unitsep}%
}
```

`\npstyleportuguese` Sets the parameters to German habit.

```
\newcommand*\npstyleportuguese{%
  \npthousandsep{\,}%
  \npdecimalsign{,}%
  \npproductsign{\cdot}%
  \npunitseparator{\,}%
  \npdegreeseparator{}%
  \nppercentseparator{\nprt@unitsep}%
}
```

Do the following actions at `\begin{document}` to ensure that it is done after loading `babel.sty` if it is loaded at all.

```
\AtBeginDocument{%
```

By default, automatic language support is switched off for compatiblity reasons. Proceed only if it is switched on.

```
  \ifnprt@autolanguage
```

Automatic language support only works with babel.

```
    \@ifpackageloaded{babel}{%
```

Adds the language settings to the known languages if they are provided by babel. The current version knows all German, English, and Portuguese dialects.

```
      \npaddtolanguage{UKenglish}{english}%
      \npaddtolanguage{USenglish}{english}%
      \npaddtolanguage{american}{english}%
      \npaddtolanguage{austrian}{german}%
      \npaddtolanguage{british}{english}%
      \npaddtolanguage{canadian}{english}%
      \npaddtolanguage{english}{english}%
      \npaddtolanguage{german}{german}%
```

```
      \npaddtolanguage{naustrian}{german}%
      \npaddtolanguage{ngerman}{german}%
      \npaddtolanguage{brazil}{portuguese}
      \npaddtolanguage{brazilian}{portuguese}
      \npaddtolanguage{portuges}{portuguese}
      \npaddtolanguage{portuguese}{portuguese}
```

Set the active language again to activate the extras section.

```
      \expandafter\selectlanguage\expandafter{\languagename}%
    }{%
```

If `babel` is not loaded but automatic language support is activated, switch to English as default:

```
      \npstyleenglish
    }%
  \fi
}
```

Load configuration file if present.

```
\InputIfFileExists{numprint.cfg}{%
  \message{Configuration file `numprint.cfg' loaded.}%
}{%
  \message{No configuration file `numprint.cfg' found.}%
}
```

`\nprt@renameerror` A command for producing an error message for redefined macros.

```
\newcommand*\nprt@renameerror[1]{%
  \expandafter\def\csname #1\endcsname{%
    \PackageError{numprint}{This command has been renamed
      to\MessageBreak
      \string\np #1}{In order to avoid problems with other
      packages and for consistency, this\MessageBreak
      command has been renamed in this version.}%
  }%
}
```

`\fourdigitsep` `\fourdigitnosep` `\addmissingzero` `\noaddmissingzero` `\digits` `\nodigits` `\exponentdigits` `\noexponentdigits` Define replacements for the old commands that produce error messages.

```
\nprt@renameerror{fourdigitsep}
\nprt@renameerror{fourdigitnosep}
\nprt@renameerror{addmissingzero}
\nprt@renameerror{noaddmissingzero}
\nprt@renameerror{digits}
\nprt@renameerror{nodigits}
\nprt@renameerror{exponentdigits}
\nprt@renameerror{noexponentdigits}
```

Change History

Since the version 1.00 is an entirely new implementation, the Change History prior version 1.00 has been lost in this document. Have a look to `numprint032.dtx` or `README` to get it.

1.00

General: Automatic support for different number formats in differ-

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.

O

P

R

S

T