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\,345.6e789. 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 \texttt{tabular}(*), \texttt{array}, \texttt{tabularx}, and \texttt{longtable} environments (similar to the \texttt{dcolumn} and \texttt{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 \texttt{nbaseprt} package which comes with the \texttt{numprint} package.

Contents

1 Load the package 3

2 Print numbers in text and math mode 3
   2.1 Print counters and lengths 5

3 Customization 6
   3.1 Eliminate separators for four-digit numbers 6
   3.2 Add missing zeros before or after decimal sign 6
   3.3 Add a plus sign 6
   3.4 Rounding numbers 6
   3.5 Padding a number on the left side 7
   3.6 Replace zeros by other characters 7
   3.7 Change the format of units 7

4 Add more units to \texttt{lenprint} 7

5 International support 8
Print aligned numbers in tabulars

6.1 The new column types
6.2 The old column types
6.3 Alignment in normal text

Error messages etc.

Advanced customization

8.1 Changing the output
  8.1.1 Without the autolanguage option
  8.1.2 With the autolanguage option
8.2 Changing the argument parsing

Some tricks

9.1 Let the signs depend on the mode
9.2 Typing negative numbers in red
9.3 Configuration file

Compatibility to older versions

Lists of options and commands

B.1 Package options
B.2 Commands

Known bugs

To do

Acknowledgements

The implementation

F.1 Load packages
F.2 Package options and settings
F.3 Error and warning messages
F.4 String parsing
F.5 Parsing of the \numprint argument
F.6 Table alignment
  F.6.1 Aligned numbers, also for ordinary text
  F.6.2 Auxiliary routines for the new column types
  F.6.3 New column types
  F.6.4 Old column types for compatibility
F.7 Round numbers
F.8 Print the numbers
F.9 The main command
F.10 Print lengths and counters
F.11 Internationalization
Copyright

Copyright 2000–2005 Harald Harders.
This program can be redistributed and/or modified under the terms of the LaTeX Project Public License Distributed from CTAN archives in directory macros/latex/base/lppl.txt; either version 1 of the License, or any later version.

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.\footnote{Section 8.2 describes how this can be changed.} “E\,”, “e\,”, “D\,”, or “d\,\textsuperscript{2}” is converted to an exponential format (e.g., x \cdot 10\textsuperscript{y} or x \times 10\textsuperscript{y}, depending on the format settings described later). “\pm” and “\+\-” produce a ±.

For example, typing
\numprint{-123456}; \numprint{\pm 123456}; \numprint{\pm1415927e-3.1}
\numprint{-123456}; \numprint{-3,1415927e-3.1}
leads to “−123.456; ±123.456; \pm 3.141.592.7 \times 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. See the difference between “123.456.134 \times 10^{123}” and “123.456.134 \times 10^{123}”, produced by

\ensuremath{\numprint{123456.134e123}} and \ensuremath{\$\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 \textit{unit} in math mode with an upright font (\texttt{\textup{mathrm}}), e.g.,

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

leads to “−123.456 \text{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 \texttt{\tcdegree} command of the mathcomp package, the \texttt{\textdegree} command of the textcomp package, or the \texttt{\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 °C (\texttt{\textcelsius}), Ω (\texttt{\textohm}), μ (\texttt{\textmu}), or % (\texttt{\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 textcomp package use \texttt{\tc{name}} instead of \texttt{\text{name}}, e.g., \texttt{\tcelsius} instead of \texttt{\textcelsius}. If using gensymb just use \texttt{name}, e.g., \texttt{\celsius}.

\numprint

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

\textsuperscript{3}This is only the case if the eco package is available on your system.

\textsuperscript{4}If you still use \texttt{name} \numprint uses \texttt{\tc{name}} or \texttt{(\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 123.45·10^{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.

\section{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]{\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., “1.234,1234” but “1,234.123,45”.

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, 12,345.12345 – 1.234.123,45, 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

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 `noaddmissingzero`. The default is `addmissingzero`.

3.3 Add a plus sign

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 `\noaddplus` resp. `noaddplus`.

The commands `\npaddplusexponent` and `\noaddplusexponent` 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

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 `\nrounddigits` command for ordinary numbers and the `\nroundexpdigits` command for exponents. Rounding is switched off with `\nroround` resp. `\nroroundexp`. For example,

```
\nrounddigits(2)\numprint{1.123}$, \numprint{1.149}$, \numprint{-9.999}$, \numprint{9.999}$$\--$
\nroround\numprint{1.123}$, \numprint{1.149}$, \numprint{-9.999}$, \numprint{9.999}$
```

leads to “1.12, 1.15, 1.00, 10.00 – 1.123, 1.149, 1, 9.999, –9.999”
3.5 Padding a number on the left side

\texttt{\nplpadding} Sometimes it is desirable to have a number of a fixed length with the missing digits filled with a character (mostly the character “0”, so this is the default). This can be achieved calling \texttt{\nplpadding[⟨character⟩]{⟨digits⟩}}. This affects only the mantissa, the part after the decimal sign and signs are not counted! If the original number has more digits than ⟨digits⟩ no characters will be inserted. For example,

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

leads to “001,234, –001,234, 12,345,678, 001,234,567,8 – xx1,234, –xx1,234, 12,345,678, xx1,234,567,8”
\texttt{\nplpadding} switches padding off.

3.6 Replace zeros by other characters

\texttt{\npreplacenull} 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 \texttt{\npreplacenull{⟨replacement⟩}}, e.g.,

\begin{verbatim}
\npreplacenull{\text{---}}
\end{verbatim}

Here, \texttt{\text} guarantees that “—” is printed in text mode.\footnote{You should better use \texttt{amsmath} and the command \texttt{\text} which preserves the correct text size, too.}

It can be switched off using \texttt{\npprintnull}.

3.7 Change the format of units

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

\begin{verbatim}
\newcommand*{\npunitcommand}[1]{\ensuremath{\mathrm{#1}}}
\end{verbatim}

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

For example, a blue unit is reached by

\begin{verbatim}
\renewcommand*{\npunitcommand}[1]{\ensuremath{\color{blue}\mathrm{#1}}}
\end{verbatim}

And here is the result: \texttt{300 N/mm²}

4 Add more units to \texttt{\lenprint}

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

\begin{verbatim}
\npdefunit{mm}{mm}{0.35145980351}
\end{verbatim}

\footnote{Provided by \texttt{amsmath}.}
The first argument \texttt{⟨unitname⟩} is the \LaTeX{} internal name that will be given as unit in the \texttt{\lenprint} command. The second argument \texttt{⟨unit⟩} is the text that will be printed out.

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

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

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

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

leads to “4.913,1 in, 4.913,1′′”.

5 International support

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

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

\texttt{\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 \texttt{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 \texttt{dcolumn} and \texttt{rcol} packages. But they have two disadvantages. First, they do not support typesetting numbers
in the same way as \texttt{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 \texttt{numprint} call in every table cell. This has been improved in version 1.10. For compatibility reasons, the old column types \texttt{n} and \texttt{N} have been preserved if you specify the \texttt{oldcolumntypes} package option; they will be discussed in section sec:column:old.

### 6.1 The new column types

#### column type \texttt{n}

The \texttt{numprint} package provides the column type \texttt{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}}
\end{tabular}

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\footnote{The tabulars are ugly here. But the only important thing is to show the effects of the alignments.} is produced by the following code:

\begin{verbatim}
\begin{tabular}{|n{5}{3}|n[3]{5}{3}|n[3]{1}{5}{3}|N{5}{3}|}
\hline
123.45 \times 10^{14} & 123.45 \times 10^{14} & 123.45 \times 10^{14} & 123.45 \times 10^{14} \\
1234.5678 \times 10^{123} & 1234.5678 \times 10^{123} & 1234.5678 \times 10^{123} & 1234.5678 \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^{12.3} \\
1234.5678 \times 10^{123.3} & 1234.5678 \times 10^{123.3} & 1234.5678 \times 10^{123.3} & 1234.5678 \times 10^{123.3} \\
\hline
\end{tabular}
\end{verbatim}

The \texttt{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 \texttt{N} column type as shown in column four of the tabular.\footnote{If you use a tabular environment that prints its arguments in math mode, e.g., the \texttt{array} environment, also \texttt{N} prints the numbers in math mode.} It takes the same arguments as the \texttt{n} column type.
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 “c”, “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&{}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:

<table>
<thead>
<tr>
<th>without braces</th>
<th>with braces</th>
<th>with braces and box</th>
<th>with braces, exp, and box</th>
</tr>
</thead>
<tbody>
<tr>
<td>abc def 12.3 × 10^3 rt</td>
<td>abc def 12.3 × 10^3 rt</td>
<td>abc def 12.3 × 10^3 rt</td>
<td>abc def 12.3 × 10^3 rt</td>
</tr>
<tr>
<td>more 45.1 txt</td>
<td>more 45.1 txt</td>
<td>more 45.1 txt</td>
<td>more 45.1 txt</td>
</tr>
<tr>
<td>not blue 45.1 txt</td>
<td>blue 45.1 txt</td>
<td>blue 45.1 txt</td>
<td>blue 45.1 txt</td>
</tr>
</tbody>
</table>

In the first column, the texts before and after the number are not enclosed 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

9It 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, \csmame, 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.

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.

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`

If you also want to use that font in math mode, you may use the math version `\npboldmath` 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:

```latex
\begin{tabular}{lN{12}{3}n{12}{3}}
\toprule
normal:& 123456123456.123e12 & 123456123456.123e12 \\
| \textbf{bold} & 123456123456.123e12 & 123456123456.123e12 \\
| \textbf{bold extended} & 123456123456.123e12 & 123456123456.123e12 \\
\bottomrule
\end{tabular}
```

This produces:
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 \nfafternum command is defined that takes one argument which is printed after the number. The following example shows text before and after the number:

\begin{tabular}{>{{before \nfafternum{ after}}}n{2}{12}{3}% >{{\nprounddigits{4}}}n{3}{4}% >{{color{blue}}}n{12}{3}}
| \toprule
| 123456.123456.123 \times 10^{12} & 123.456.123456.123 \times 10^{12} \\
| 123.456.123456.123 \times 10^{12} & 123456.123456.123 \times 10^{12} \\
| \bottomrule
\end{tabular}

This produces:

| before 123,456,123,456,123 \times 10^{12} after | 12.124 | 123,456.23 \times 10^{12} \\
| before \begin{tabular}{>{{\npunit{N/mm^2}}}n{5}{3}}
| \toprule
| 12345.123 \begin{\npunit{N/mm^2}}\end{\npunit{N/mm^2}} & 12.12 \begin{\npunit{psi}}\end{\npunit{psi}} \\
| 234.4 \begin{\npunit{psi}}\end{\npunit{psi}} & 4.3 \begin{\npunit{N/mm^2}}\end{\npunit{N/mm^2}} \\
| \bottomrule
\end{tabular}

This produces:

12345.123 N/mm$^2$
12.12 N/mm$^2$
234.4 psi
4.3 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

\begin{tabular}{|n{5}{3}|N{5}{3}{3}|}
\hline
123.45 & \numprint{123.45e12} \\
12,345.678e123 & \numprint{12,345.678e123} \\
123.45 & \numprint{123.45e12.3} \\
12,345.678e123.3 & \numprint{12,345.678e123.3} \\
\hline
\end{tabular}

This leads to

\begin{align*}
123.45 & \times 10^{12} & 123.45 & \times 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} \\
\end{align*}

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.

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.

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

By using the commands \npdecimalsign\{Sign\}, \npthousandsep\{Separator\},
\npthousandthpartsep\{Separator\}, and \npproductsign\{Sign\}, several
separators and symbols can be changed, e.g.,

```
\npdecimalsign\{\text\{\text\{Sign\}\}\}\npthousandsep\{,\}\npproductsign\{\text\{\text\{Sign\}\}\}\n\numprint{-123\,\text\{\text\{\text\{\text\{Sign\}\}\}\}\,141.592,7 \times 10^{-3}}
```

leads to “−123,456; 3·141.592,7 * 10^{−3}”.

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 \textmath for math or either \mbox, \textrm, or \text for text mode.

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

If using these commands inside a group {\ldots}, \begin{group}...\end{group}, 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{123\,e4},
\{\npproductsign\{\text\{\text\{Sign\}\}\}\numprint{123\,e4}\},
\numprint{123\,e4}.
```

Global:
```
\numprint{123\,e4},
\{\global\npproductsign\{\text\{\text\{Sign\}\}\}\numprint{123\,e4}\},
\numprint{123\,e4}
```

leads to the following:
Local: $123 \times 10^4$, $123 \cdot 10^4$, $123 \times 10^4$. Global: $123 \times 10^3$, $123 \cdot 10^3$, $123 \cdot 10^3$

The current version has the following defaults:

\footnote{These command did not have the “np” in older versions. This had to be changed in order to
avoid an incompatibility with the french language of babel.}

\footnote{Provided by amsmath.}
The space between the number and the unit is \texttt{\,} by default. It can be changed using the command \texttt{\npunitseparator\{Separator\}}, e.g.,
\begin{verbatim}
\npunitseparator{\textasciicolon}
\end{verbatim}

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

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 \texttt{\npercentseparator{\}}.

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 \texttt{\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 \texttt{\npstyle\{language\}} is defined that does the changes, e.g., \texttt{\npstyleenglish} for English. This command is defined as follows:
\begin{verbatim}
\newcommand*{\npstyleenglish}{% 
  \npthousandsep{\,} \npdecimalsign{.} \npproductsign{\times} \npunitseparator{\,} \npdegreeseparator{} \npercentseparator{\nprt@unitsep} %}
\end{verbatim}

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

1. Copy the definition for \texttt{\npstyle\{language\}} from the style file and change it according to your wishes, for example:\footnote{Notice that you have to use \texttt{\renewcommand*} instead of \texttt{\newcommand*}.}
\begin{verbatim}
\renewcommand*{\npstyleenglish}{% 
  \npthousandsep{\,} \npdecimalsign{\cdot} \npproductsign{\times} \npunitseparator{\,} \npdegreeseparator{} \npercentseparator{\nprt@unitsep} %}
\end{verbatim}

2. Another way to add different settings to a language is to use \texttt{\g@addto@macro} to append commands to an existing \texttt{\npstyle\{language\}} command, e.g.,
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 \texttt{numprint} you may add support for it in the preamble of your document.

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

\begin{verbatim}
\nptaddtolanguage{danish}{german}
\end{verbatim}


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 \texttt{nptstyledanish} which defines everything you want to change against the default (I choose some strange values for clearness):

\begin{verbatim}
\newcommand*{nptstyledanish}{%
  nptthousandsep{.}%
  npdecimalsign{\ensuremath{\cdot}}%
  npproductsign{*}%
  npunitseparator{~}%
  npdegreeseparator{}%
  nppercentseparator{\prt@unitsep}%
}%
\end{verbatim}

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

\begin{verbatim}
\nptaddtolanguage{danish}{danish}
\end{verbatim}

### 8.2 Changing the argument parsing

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

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

\begin{verbatim}
\newcommand*{nprt@dotlist}{,.}
\end{verbatim}

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

\begin{verbatim}
\renewcommand*{nprt@dotlist}{.}
\end{verbatim}
If you want to do this in your document rather than in the configuration file `numprint.cfg`—see section 9.3—you have to enclose this by \makeatletter and \makeatother.

The \texttt{\np@explist} command contains the characters interpreted as delimiter between mantissa and exponent. By default it contains “eEdD”. Redefine it as \texttt{\np@dotlist}.

The \texttt{\np@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:

\begin{verbatim}
\renewcommand*{\np@dotlist}{.}
\renewcommand*{\np@ignorelist}{,}
\end{verbatim}

Then, “,” is ignored in the input and “.,” is interpreted as decimal sign. For example,

\begin{verbatim}
\makeatletter
{\renewcommand*{\np@dotlist}{.} \renewcommand*{\np@ignorelist}{,} \numprint{12,234.123,45e1,2,3.0}}
\makeatother
\end{verbatim}

leads to “12,234.123,45 × 10^{123}”.

The \texttt{\np@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 \texttt{\np@signlist}. If, for instance, the letter “*” is intended to be a sign, just type in

\begin{verbatim}
\renewcommand*{\np@signlist}{+-\pm *}
\end{verbatim}

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, \texttt{\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 the sign. This command must have a name according to \texttt{\np@list@\langle\text{sign}\rangle}. In this case, you have to define \texttt{\np@list@*}. Since the sign might be arbitrary characters, you should define the command as follows:

\begin{verbatim}
\expandafter\newcommand\csname nprt@sign@*\endcsname{\ensuremath{*}}
\end{verbatim}

With this command, \texttt{\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 \texttt{\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

The default signs are typeset in math mode independently of the mode the number ist printed. If you are using a font in which the sings of text and math mode differ

¹³\texttt{\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}.” To avoid a negative exponent being 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 Compatibility 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 environments other than $\textstyle$.

\footnote{Whether 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 missing 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 the exponent when it has no 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 missing zeros before or after the decimal sign.
- `\npnoaddmissingzero`: Switch off adding missing 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.
- `\pnoaddplusexponent`: 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.
\pnnoroundexp 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.
\npeponentdigits Switch on aligned printing of the exponent.
\npnexponentdigits 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


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.

5 \newif\ifnprt@numsepfour

Show warnings or errors?

6 \newif\ifnprt@errormessage

Add missing zeros?

7 \newif\ifnprt@addmissingzero

Add missing plus signs?

8 \newif\ifnprt@addplus@mantissa
9 \newif\ifnprt@addplus@exponent

Switch the style depending on the language automatically?

10 \newif\ifnprt@autolanguage

Old or new column types?

11 \newif\ifnprt@newcolumntype

Declare new math type?

12 \newif\ifnprt@npbold

\npfourdigitsep Switch separating of the fourth digit on.
13 \newcommand*{\npfourdigitsep}{\nprt@numsepfourtrue}\

\npfourdigitnosep Switch separating of the fourth digit off.
14 \newcommand*{\npfourdigitnosep}{\nprt@numsepfourfalse}\

\npaddmissingzero Add a missing zero before or after decimalsign.
15 \newcommand*{\npaddmissingzero}{\nprt@addmissingzerotrue}\

\npnoaddmissingzero Don’t add a missing zero before or after decimalsign.
16 \newcommand*{\npnoaddmissingzero}{\nprt@addmissingzerofalse}\

\npaddplus Add a plus to the mantissa if no sign is given.
17 \newcommand*{\npaddplus}{\nprt@addplus@mantissatrue}\

\npnoaddplus Don’t add a plus to the mantissa if no sign is given.
18 \newcommand*{\npnoaddplus}{\nprt@addplus@mantissafalse}\

\npaddplusexponent Add a plus to the exponent if no sign is given.
19 \newcommand*{\npaddplusexponent}{\nprt@addplus@exponenttrue}\

\npnoaddplusexponent Don’t add a plus to the exponent if no sign is given.
20 \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.
21 \DeclareOption{nosepfour}{\npfourdigitnosep}\
22 \DeclareOption{sepfour}{\npfourdigitsep}
Add missing zeros after decimal sign?
23 \DeclareOption{noaddmissingzero}{\npnoaddmissingzero}
24 \DeclareOption{addmissingzero}{\npaddmissingzero}

Add a plus sign to the mantissa if no sign is given?
25 \DeclareOption{noaddplus}{\npnoaddplus}
26 \DeclareOption{addplus}{\npaddplus}

Add a plus sign to the exponent if no sign is given?
27 \DeclareOption{noaddplusexponent}{\npnoaddplusexponent}
28 \DeclareOption{addplusexponent}{\npaddplusexponent}

Add missing zeros after decimal sign?
29 \DeclareOption{noautolanguage}{\nt@autolanguagefalse}
30 \DeclareOption{autolanguage}{\nt@autolanguagetrue}

\np Define a shortcut for the \numprint command?
31 \DeclareOption{np}{\newcommand*{\np}{\numprint}}

Produce warnings or errors?
33 \DeclareOption{oldcolumntypes}{\nt@newcolumntypefalse}
34 \DeclareOption{newcolumntypes}{\nt@newcolumntypetrue}

Declare new math type?
35 \DeclareOption{boldmath}{\nt@npboldtrue}

Produce warnings or errors?
36 \DeclareOption{warning}{\nt@errormessagefalse}
37 \DeclareOption{error}{\nt@errormessagetrue}

Generate some debug information to the log file?
38 \newcommand*{\nt@debug}[1]{\PackageInfo{\numprint}{\np}}
39 \DeclareOption{debug}{\renewcommand*{\nt@debug}[1]{\PackageInfo{\numprint}{\np}}}

Execute default options and the given options.
41 \ExecuteOptions{sepfour,addmissingzero,error,noautolanguage,newcolumntypes}
42 \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.
43 \newcommand*{\npdecimalsign}[1]{\def\nt@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.
44 \newcommand*{\npthousandsep}[1]{\def\nt@separator@before{{#1}}}\def\nt@separator@after{{#1}}
45 \def\nt@separator@before{{#1}}\def\nt@separator@after{{#1}}

\npthousandthpartsep Change the thousand separator after the decimal sign.
46 \newcommand*{\npthousandthpartsep}[1]{\def\nt@separator@after{{#1}}}
Change the product sign that is printed in numbers with exponent like $123 \times 10^4$. 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{}}}}

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

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

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

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

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

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

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}

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}}

A command to switch off rounding.

\newcommand*\pnoround{\nprounddigits{-1}}

Don’t round by default.

\pnoround

The same for exponents.

\newcommand*\nprroundexpdigits[1]{\def\nprt@roundexpdigits{#1}\def\nprt@roundexpnull{}\nprt@fillnull{\nprt@roundexpnull}{#1}}

\newcommand*\pnoroundexp{\nprroundexpdigits{-1}}
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}
}

The command to switch of padding.
\newcommand*\npnolpadding{\nplpadding[@empty][-1]}

Don’t pad by default.
\npnolpadding

Optionally, one or more zeros after the decimal sign can be replaced by other text, e.g., “—”.
\newcommand*\npreplacenull[1]{\def\nprt@replacenull{#1}}

Switch off replacement, again.
\newcommand*\npprintnull{\let\nprt@replacenull=\@empty}

Print the zero by default.
\npprintnull

This command is used to typeset the unit.
\newcommand*\npunitcommand[1]{\ensuremath{\mathrm{#1}}}
Initialize the numbers.
\def\npnoexponentdigits{\number\@ne}%
\def\npnoexponentdigits@after{-1}%
\npnoexponentdigits Switch back to normal printing.
\newcommand*{\npnoexponentdigits}{\npnoexponentdigitsfalse}

F.3 Error and warning messages
Define a boolean which helps \numprint to detect errors.
\newif\nprt@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 occurred.
\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\nprt@charfound

\nprt@IfCharInString Define \nprt@IfCharInString{\{Message\}}{\{Help text\}} which does the same.
\newcommand{\nprt@IfCharInString}[2]{% \begingroup \def\nprt@searchfor{#1} \edef\nprt@argtwo{#2} \expandafter\nprt@@IfCharInString \prt@argtwo\@empty\@empty \ifnprt@charfound \expandafter\@firstoftwo \else \expandafter\@secondoftwo \fi \endgroup}

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

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

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

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

Counters for the number of digits before and after the decimal sign of the mantissa.
\newcounter{\prt@mantissa@digitsbefore}
\newcounter{\prt@mantissa@digitsafter}

Counters for the number of digits before and after the decimal sign of the exponent.
\newcounter{\prt@exponent@digitsbefore}
\newcounter{\prt@exponent@digitsafter}

Boolean to store if an exponent will be printed.
\newif{\nprt@expfound}

Boolean to store if mantissa resp. exponent contain a decimal sign.
\newif{\nprt@mantissa@decimalfound}
\newif{\nprt@exponent@decimalfound}

Define \texttt{\prt@testsign}\{\textit{Number type}\}\{\textit{Number}\} which tests whether a sign is given and then starts \texttt{\prt@testnumber} (the call actually is done by \texttt{\prt@testsign}).

First, store the expanded arguments in macros.
\edef{\prt@commandname}{#1}
\edef{\prt@tmp}{#2}
Call the working command \texttt{\npertestsign}. The large number of \texttt{\expandafter} calls is necessary to ensure that the second to fourth argument are already expanded (thus, \texttt{#2} and \texttt{#3} are single characters). Append enough \texttt{\@empty} to ensure the argument-end marker is found even for an empty number.

\begin{verbatim}
\expandafter\npertestsign\expandafter{% 
\expandafter\npertcommandname\expandafter}%
\nperttmp\@empty\@empty\@empty\@empty}
\end{verbatim}

\texttt{\npertestsign} 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 \texttt{#2} to \texttt{#4} are the given number, where \texttt{#2} and \texttt{#3} contain of the first resp. second character of the number, while \texttt{#4} contains the rest.

\begin{verbatim}
\def\npertestsign#1#2#3#4\@empty{%
Store the first argument to a macro.
\edef\npertcommandname{#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, \texttt{#2}, contains a sign symbol which is listed in \texttt{\npertsignlist}.
\begin{verbatim}
\expandafter\xdef\csname nprt@#1@sign\endcsname{\@empty}\
\expandafter\xdef\csname nprt@#1@sign\endcsname{\@empty}\
\end{verbatim}
\end{verbatim}

If yes, store that sign in the command \texttt{\npert(#2)@sign}. If the sign is a "+" the second character of the number may be a "-" to replace that combination by \texttt{\pm}. Thus, do an extra handling of that case.

\begin{verbatim}
\expandafter\xdef\csname nprt@#1@sign\endcsname{\@empty}\
\expandafter\xdef\csname nprt@#1@sign\endcsname{\@empty}\
\end{verbatim}

The digits start at the third character of the number string which is \texttt{#4}. Start \texttt{\nperttestnumber} to parse the digits of the number if a number is given after the given sign "+-". If not, empty \texttt{\npert(\#1)} that \texttt{\numprint} can determine that only a sign has been given.

\begin{verbatim}
\def\nperttestnumber{}% 
\ifx\npert@tmp\@empty
\expandafter\xdef\csname nprt@#1@endcsname{\@empty}\
\expandafter\xdef\csname nprt@#1@endcsname{\@empty}\
\else
\expandafter\xdef\csname nprt@#1@endcsname{\@empty}\
\expandafter\xdef\csname nprt@#1@endcsname{\@empty}\
\fi
\end{verbatim}
If there is a single sign character “+”, the digits start at \#3. Start \texttt{\npert@testnumber} to parse the digits of the number if a number is given after the given sign “+”. If not, empty \texttt{\npert@(#1)} that \texttt{\numprint} can determine that only a sign has been given.

\begin{verbatim}
177 \ifx#3\@empty
178 \expandafter\edef\csname nprt@#1\endcsname{\@empty}%
179 \else
180 \expandafter\npert@testnumber\expandafter\npert@commandname#3#4\@empty
181 \fi
182 \fi
183 \else
184 If the sign is a “\pm” store “+-” as sign.
185 \expandafter\ifx\csname nprt@#1@sign\endcsname\npert@plusminus@test
186 \expandafter\xdef\csname nprt@#1@sign\endcsname{+-}%
187 \fi
188 If there is a single sign character other than “+”, the digits start at \#3. Start \npert@testnumber to parse the digits of the number if a number is given after the given sign. If not, empty \npert@(#1) that \texttt{\numprint} can determine that only a sign has been given.
189 \ifx#3\@empty
190 \expandafter\edef\csname nprt@#1\endcsname{\@empty}%
191 \else
192 \expandafter\npert@testnumber\expandafter\npert@commandname#3#4\@empty
193 \fi
194 \fi
195 \}
196 \}
\end{verbatim}

\texttt{\npert@testnumber} As with \texttt{\npert@testsing}, 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.

\begin{verbatim}
198 \def\npert@testnumber#1#2#3\@empty{%
199 \edef\npert@commandname(#1)\%
200 \edef\npert@argthree(#3)\%
201 Test whether the current character is a valid character for a real number (say a digit or a decimal sign).
202 \npert@IfCharInString(#2){\npert@numberlist}\npert@dotlist}{}
203 \}
\end{verbatim}

If this is the case, continue testing. If the current character is a decimal sign, set the boolean \texttt{\ifnprt@(#1)\npert@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.

\begin{verbatim}
205 \npert@IfCharInString(#2){\npert@dotlist}{}
\end{verbatim}
\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 recursively 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 an 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}{%
Yes, it is valid.

Now, test whether 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.

\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 whether 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\numberlist\dotlist\explist\signlist\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\{{\langle Number type\rangle}\}\{{\langle Position\rangle}\}\{{\langle Math mode\rangle}\}, where \langle Number type\rangle is either “mantissa” or “exponent”, \langle Position\rangle is the position relative to the decimal sign (“before” or “after”), \langle Math mode\rangle is a math mode command (\text{\textstyle}, \text{\scriptstyle}, or \text{\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.
\if\nprt@argone{\nprt@mantissaname}\
\if\textmode\
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 \text{\mathchoice}.
\settowidth{\nprt@digitwidth}{$#30$}\
\settowidth{\nprt@sepwidth}{$#3\text{\csname nprt@separator@#2\endcsname}$}\
\settowidth{\nprt@decimalwidth}{$#3\nprt@decimal$}\
\else\
Do the same for text mode.
\settowidth{\nprt@digitwidth}{0}\
\settowidth{\nprt@sepwidth}\text{\csname nprt@separator@#2\endcsname}\nprt@sepwidth}\
\settowidth{\nprt@decimalwidth}{\nprt@decimal}\
\fi\
The same for the exponent.
\else\
\fi

\else
\fi

\}{\nprt@calcblockwidth}[3]
\else
  \settowidth{\nprt@digitwidth}{\textsuperscript{0}}\%
  \settowidth{\nprt@sepwidth}{%\textsuperscript{\csname nprt@separator#2\endcsname}}\%
  \settowidth{\nprt@decimalwidth}{\textsuperscript{\nprt@decimal}}\%
\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 (+ − ±) 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
    \addtoafter{\csname nprt@#1@fixeddigits@after\endcsname}\%
    \addtolength{\nprt@blockwidth}{\the\nprt@decimalwidth}\%
  \fi
\fi
\fi
\fi
\fi

33
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.

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

Initialize `\nprt@unit`.

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

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.

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

Initialize `\nprt@afternum`.

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

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.

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

The internal part of the `\npmakebox` command.

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

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

```latex
\if\nprt@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}
\fi
```

Switch to that bold math alphabet.

```latex
\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.

The token list that will contain all characters of a tabular cell that are allowed for `\numprint`.

```latex
\newtoks\nprt@digittoks
```
Token list with all tokens before the number itself.

Token list with all tokens after the number itself.

Has the number already been found in the parsing of the tabular cell?

This macro is executed at the begin of each tabular cell.

Initialize the tokens and macros.

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

Start to parse the tabular cell.

Adds the current token to the list of tokens before or after the number.

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

Start to 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.

If the current token is \tabularnewline or \ (which is the same) the tabular cell is finished. 
Redefine the \nprt@next command that is called at the end of this command to execute the found \tabularnewline command.
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 \tablularx, 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@saveothurtok{#1}%
}\fi % \relax
\fi % \csname
\fi % \endtabular
\fi % \nprt@end
\fi % \end
\fi % \tablularnewline
Call the previously saved command (recursion or end of the tabular cell).

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

This macro is called at the end of each tabular cell. The arguments are used as follows: \langle \text{digits before} \rangle, \langle \text{digits after} \rangle \text{ the decimal sign for the mantissa; } \langle \text{digits before} \rangle, \langle \text{digits after} \rangle \text{ the decimal sign for the exponent; commands inserted } \langle \text{before} \rangle \text{ and } \langle \text{after} \rangle \text{ the } \texttt{\numprint} \text{ command. The arguments five and six are empty for printing the number in text mode and contain } \langle \text{"\$"} \rangle \text{ for printing the number in math mode.}

First, print the tokens before the number.

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

Set the digits for the alignment of the mantissa.

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

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

Print the pre-command if defined. Since the pre-command is fixed to nothing or \langle \text{"\$"} \rangle to switch to math mode, this is only done if the math mode is not active, before. Set the boolean \texttt{\ifnprt@mathtabular} according to the previous status in order to close the math mode if necessary.
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
  \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 for/hyphen.alt
matted 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
}

\section*{F.6.3 New column types}

Define the new column types.
\ifnprt@newcolumntype
  Declare a new column type \texttt{N} which prints a number in text mode and does not need to repeat \texttt{\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 \texttt{N} column type to look for an optional argument. If none is given, set both optional arguments to \texttt{"-1"}.
\def\NC@rewrite@N{\%
  \nprt@digittoks{}\%
  \nprt@pretoks{}\%
  \@ifnextchar[{}{\%
    \nprt@rewrite@@{}{}{-1}{-1}\%
  }{\}
}

\NC@rewrite@N
Declare a new column type \texttt{n} which prints a number and does not need to repeat the \texttt{numprint} in each tabular cell. This is declared empty because all executing macros are redefined anyway.

\begin{verbatim}
467 \newcolumntype{n}{\phantom{0}}
\end{verbatim}

\texttt{\textbackslash NC\textbackslash rewrite@n} Same as \texttt{NC\textbackslash rewrite@N}, but give “\$” twice for math mode.

\begin{verbatim}
468 \def\NC\textbackslash rewrite@n{\%
469 \nprt@digittoks{}
470 \nprt@pretoks{}
471 \@ifnextchar[{{%
472 \nprt@rewritetoks{$}{$}%
473 }{%
474 \nprt@rewritetoks{$}{$}{-1}{-1}%
475 }%
476 }
\end{verbatim}

\texttt{\textbackslash nprt@rewrite@0} Look for the second optional argument.

\begin{verbatim}
477 \def\nprt@rewrite@0#1#2[#3]{%
478 \@ifnextchar[{{%
479 \nprt@rewritetoks[#1]{#2}{#3}%
480 }{%
481 \nprt@rewritetoks[#1]{#2}{#3}{-1}%
482 }%
483 }
\end{verbatim}

\texttt{\textbackslash nprt@rewrite@00} The arguments are used as follows: commands inserted \texttt{(before)} and \texttt{(after)} the \texttt{numprint} command; \texttt{(digits before)}, \texttt{(digits after)} the decimal sign for the mantissa; \texttt{(digits before)}, \texttt{(digits after)} the decimal sign for the exponent.

\begin{verbatim}
484 \def\nprt@rewrite@00#1#2#3[#4][#5][#6]{%
\end{verbatim}

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

\begin{verbatim}
485 \def\nprt@rewrite@scratch{\the@temptokena
486 >{\noexpand\nprt@begin}\noexpand\ignorespaces}]
487 {<\noexpand\nprt@end[#5]{#6}{#3}{#4}{#1}{#2}]}
488 }
\end{verbatim}

Set \texttt{@temptokena} to the preceding and the current column definition. This is implicitly used by the \texttt{array} package.

\begin{verbatim}
489 \@temptokena\expandafter{\nprt@rewrite@scratch}%
\end{verbatim}

Parse for next column in the definition.

\begin{verbatim}
490 \NC\textbackslash find
491 }
\end{verbatim}

\textbf{F.6.4 \ Old column types for compatibility}

Define the old column types.

\begin{verbatim}
492 \else
\end{verbatim}

The column type \texttt{n} aligns the base number but not the exponent.

\begin{verbatim}
493 \newcolumntype{n}[2]{>{\npdigits[#1]}{#2}{$}l<{$}}
\end{verbatim}
The column type \texttt{N} aligns the base number as well as the exponent.

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

End of column type section.

\section*{F.7 Round numbers}

Stores if the current digit has to be rounded up.

\begin{verbatim}
\newif\ifnprt@roundup
\DeclareCounter{nprt@round@after}
\DeclareCounter{nprt@round@before}

\def\nprt@round@after#1#2\@empty{%
\edef\nprt@argone{#1}%
\edef\nprt@argtwo{#2}%
\texttt{Count the parsed digits.}
\texttt{If the end of the number is reached an internal error has been occurred since enough zeros are appended to the number before.}
\texttt{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 \texttt{\ifnprt@roundup}. If this position is reached, stop the recursion.}
\texttt{The position has not been reached, yet. Thus, call this routine recursively.}
\texttt{The following lines are executed in the backward run when rounding.}
\texttt{Store the current digit in a number in order to be able to calculate with it.}
\texttt{If this number is to be rounded up do it by advancing it by one.}
\texttt{\texttt{\expandafter\nprt@round@after\@empty\@empty}}
\end{verbatim}
Reset \ifnprt@roundup since the preceeding digit is not to be rounded, normally.

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.

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

\def\prt@round@before#1#2\@empty{%
Store the arguments.
\edef\prt@argone{#1}%
\edef\prt@argtwo{#2}%
Do the recursion until the end of the number. This routine does not have to decide
whether the number has to be rounded since it knows that by \ifnprt@roundup,
set by \prt@round@after.
\ifx\prt@argone\@empty
\else
\expandafter\prt@round@before\expandafter\prt@argtwo\@empty
\fi
Store the current digit into a counter and use zero if the number before the decimal
sign is empty.
\ifx\prt@argone\@empty
\prt@thisdigit=0
\else
\prt@thisdigit=#1
\fi
Add one to the number if it has to be rounded.
\ifnprt@roundup
\advance\prt@thisdigit by 1
Reset the rounding of the next number.
\prt@roundupfalse
If rounded from 9 to 10, set this digit to “0” and give the information to the next
digit.
\ifnum\prt@thisdigit=10
\prt@thisdigit=0
\prt@rounduptrue
If the number is empty, the new number has to be added and the counter adjusted.
\fi
\fi

\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 \texttt{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 
\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 position 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@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 modified, 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
prt@round@before
prt@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
prt@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.

\begin{verbatim}
\def\framebox#1{\begin{tabular}{|c|}
\hline
#1 \\
\hline\end{tabular}}
\end{verbatim}

F.8 Print the numbers
\begin{verbatim}
define commands for printing the signs in math mode. This ensures that the printed signs really are signs and not hyphens. Compare “-” to “−”.
\newcommand{\nprt@sign@+}{}%\newcommand{\nprt@sign@-}{}%\newcommand{\nprt@sign@+-}{}
\newcommand{\nprt@printsign}{...}
\newcommand{\nprt@debug}{...}
\newcommand{\nprt@marg}{...}
\newcommand{\nprt@addplus}{...}
\newcommand{\nprt@debug}{...}
\end{verbatim}

Internal counters for printing.
\begin{verbatim}
\newcounter{\nprt@digitsfirstblock}
\newcounter{\nprt@blockcnt}
\end{verbatim}

44
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 whether 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 preceding 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}%
\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}{%
Depending on that number, call \texttt{\nprt@printone}, \texttt{\nprt@printtwo}, resp. \texttt{\nprt@printthree} which do what you may expect with that names.

\begin{verbatim}
\def\nprt@printthree#1#2#3#4\@empty{%
  #1#2#3%
  \def\prt@tmp{#4}%
  \ifx\prt@tmp\empty%
    \else
      \nprt@separator@before%
      \nprt@printthree#3\@empty\@empty\@empty
    \fi
  \fi
}
\def\nprt@printtwo#1#2#3\@empty{%
  #1#2%
  \def\prt@tmp{#3}%
  \ifx\prt@tmp\empty%
    \else
      \nprt@separator@before%
      \nprt@printthree#3\@empty\@empty\@empty
    \fi
  \fi
}
\def\nprt@printone#1#2\@empty{%
  #1%
  \def\prt@tmp{#2}%
  \ifx\prt@tmp\empty%
    \else
      \nprt@separator@before%
      \nprt@printthree#3\@empty\@empty\@empty
    \fi
  \fi
}
\end{verbatim}

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

\texttt{\nprt@printtwo} The same but start with two instead of three digits.

\texttt{\nprt@printone} The same but start with one instead of three digits.
\newcommand*{\np@printafter}{\@empty}

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

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.

\ifx\np@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

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\np@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 \np@numbertoprint and continue only if it is not empty.

\begingroup
\edef\np@numbertoprint{\csname nprt@#1@after\endcsname}
\ifx\np@numbertoprint\@empty
\else
\ifnum\csname thenprt@#1@digitsafter\endcsname=0
\expandafter\edef\csname nprt@#1@after\endcsname{0}
\stepcounter{nprt@#1@digitsafter}
\fi
\fi
\fi

Find out whether separators have to be inserted.

\ifnprt@numsepfour
\else
\ifnum\csname thenprt@#1@digitsbefore\endcsname<5
\ifnum\csname thenprt@#1@digitsafter\endcsname<5
\np@shortnumbertrue
\fi\fi\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\np@replacenull\@empty
\else
\ifnum\np@numbertoprint=0
\fi
\fi

47
If the number is short (without separators) just print it.

\Ifnprt@shortnumber
\nprt@numbertoprint
\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.
\expandafter\nprt@printthree@after%
\nprt@numbertoprint@empty@empty@empty
\fi
\fi
\endgroup

\nprt@printthree@after The same as \nprt@printthree but with another separator.
\def\nprt@printthree@after#1#2#3#4@empty{%
#1#2#3%
\def\nprt@tmp{#4}%
\ifx\nprt@tmp\empty%
\else
\nprt@separator@after
\nprt@printthree@after#4@empty@empty@empty
\fi
\fi
\}

\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.
\DeclareRobustCommand*\numprint[2][@empty]{%
\Switchofftheerrorflag. This should not be necessary but is done for stability reasons.
\nprt@argumenterrorfalse
\clearthemantissaandtheexponent.
\xdef\nprt@exponent[@empty]%
\xdef\nprt@mantissa[@empty]%
Do everything inside a group to avoid defining too many temporary macros that are not deleted after the macro.
\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.
\begingroup
Don’t expand the unit because that may cause trouble.

Declare some commands to detect empty arguments.

Some debug information.

Test for an empty mandatory argument.

Some debug information.

Test whether only valid characters have been used and divide the argument in the mantissa and the exponent.

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.

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 $\text{numprint}$ If it is not empty, parse it for a sign and a number.

If the mantissa contains only a sign, $\text{nprt@mantissa}$ is set to $\empty$ and thus empty even if it wasn’t 4 lines above. If the mantissa only contains a sign and an exponent is given, everything is fine. If no exponent is given, the input format is invalid.
Round the mantissa if necessary.

Pad the mantissa if necessary.

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

Test whether an exponent character was given but no exponent.

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.

Print the mantissa if present.

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.

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.

Calculate the width of the block for the \displaystyle.

Generate a box with the calculated width and the corresponding math style.
Print the sign if present.
\prt@prints{mantissa}{\prt@mantissa@sign}

Print the integer part into the box.
\prt@printbefore{mantissa}
\{
\prt@mantissa@sign\}

Do the same for \textstyle.
\prt@calcblockwidth{mantissa}{before}{\textstyle}
\makebox[\the\prt@blockwidth][r]{\textstyle
\prt@prints{mantissa}{\prt@mantissa@sign}
\prt@printbefore{mantissa}}
\{
\prt@mantissa@sign\}

Do the same for \scriptstyle.
\prt@calcblockwidth{mantissa}{before}{\scriptstyle}
\makebox[\the\prt@blockwidth][r]{\scriptstyle
\prt@prints{mantissa}{\prt@mantissa@sign}
\prt@printbefore{mantissa}}
\{
\prt@mantissa@sign\}

Do the same for \scriptscriptstyle.
\prt@calcblockwidth{mantissa}{before}{\scriptscriptstyle}
\makebox[\the\prt@blockwidth][r]{\scriptscriptstyle
\prt@prints{mantissa}{\prt@mantissa@sign}
\prt@printbefore{mantissa}}
\{
\prt@mantissa@sign\}

If the number is printed in text mode, the size is preserved inside the box. Thus, no hack as for math mode is necessary.
\prt@calcblockwidth{mantissa}{before}{\@empty}
\makebox[\the\prt@blockwidth][r]{\@empty
\prt@prints{mantissa}{\prt@mantissa@sign}
\prt@printbefore{mantissa}}
\{
\prt@mantissa@sign\}

If the number is printed without fixed width, just print it.
\prt@prints{mantissa}{\prt@mantissa@sign}
\prt@printbefore{mantissa}
\{
\prt@mantissa@sign\}

Print the after-decimal-sign digits. This works exactly as the integer part.
\ifs@mantissa@fixeddigits
\ifs@mode
\mathchoice{%\iftex@mode
\prt@calcblockwidth{mantissa}{after}{\displaystyle}
\makebox[\the\prt@blockwidth][l]{\displaystyle
\ifs@mantissa@decimalfound
\prt@decimal
\fi
\prt@printafter{mantissa}}\{
\prt@mantissa@sign\}
\fi
\else
\prt@calcblockwidth{mantissa}{after}{\textstyle}
\makebox[\the\prt@blockwidth][l]{\textstyle
\ifs@mantissa@decimalfound
\prt@decimal
\fi
\prt@printafter{mantissa}}\{
\prt@mantissa@sign\}
\fi
\fi
\else
\prt@calcblockwidth{mantissa}{after}{\@empty}
\makebox[\the\prt@blockwidth][l]{\@empty
\prt@printafter{mantissa}}
\{
\prt@mantissa@sign\}
\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}%
\fi
\else
\fi
\else
\ifnprt@mantissa@decimalfound
\nprt@decimal
\fi
\nprt@printafter{mantissa}%
\fi
\else
\fi
\else
\fi
\fi
\else
\fi
\fi
\else
\nprt@error{Invalid number format in exponent. Printing garbage}{Something is wrong in the format of the exponent}%
\fi
\else
\fi
\ifnprt@exponent@fixeddigits
\def\nprt@printexp##1{\hphantom{##1}}%
\else
\fi
\fi
\fi
\else
\fi
\else
\fi
\fi
\else
\fi
\ifnprt@exponent@empty
\nprt@error{Invalid number format in exponent. Printing garbage}{Something is wrong in the format of the exponent}%
\fi
\else
\fi
\ifnprt@exponent@fixeddigits
\def\nprt@printexp##1{\hphantom{##1}}%
\else
\fi
\fi
\fi
\ifnprt@mantissa@decimalfound
\nprt@decimal
\fi
\nprt@printafter{mantissa}%
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 \textcolor{red}{} command is redefined to do nothing here. Do it in a group to restrain this to the exponent itself.
\begingroup
\def\color##1{}%
\begingroup
\def\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.
\fi
\begingroup
\def\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.
\endgroup
\ifnprt@exponent@fixeddigits
\else
\mathchoice{%
\makebox[\the\nprt@blockwidth][r]{$\displaystyle$
\nprt@printsign{exponent}{\nprt@exponent@sign} \nprt@printbefore{exponent}$}%
\makebox[\the\nprt@blockwidth][r]{$\textstyle$
\nprt@printsign{exponent}{\nprt@exponent@sign} \nprt@printbefore{exponent}$}%
\makebox[\the\nprt@blockwidth][r]{$\scriptstyle$
\nprt@printsign{exponent}{\nprt@exponent@sign} \nprt@printbefore{exponent}$}%
\makebox[\the\nprt@blockwidth][r]{$\scriptscriptstyle$
\nprt@printsign{exponent}{\nprt@exponent@sign} \nprt@printbefore{exponent}$}%
\endgroup
\else
\makebox[\the\nprt@blockwidth][r]{\@empty}%
\endgroup
Produce a warning since this is uncommon in exponents.
\ifnumprt@exponent@decimalfound
PackageWarning{numprint}{Non-integer exponent}%
\fi
Print the part after the decimal sign.
\ifnumprt@exponent@fixeddigits
\ifmmode
\makebox[\the\np@blockwidth][r]{%}
\np@printsign{exponent}{\np@exponent@sign}%
\np@printbefore{exponent}%
\fi
\else
\np@printsign{exponent}{\np@exponent@sign}%
\np@printbefore{exponent}%
\fi
\fi
\else
\makebox[\the\np@blockwidth][r]{%}
\np@printsign{exponent}{\np@exponent@sign}%
\np@printbefore{exponent}%
\fi
\fi
If the unit is not empty, print it, too.

\fi
\else
\if\prt@exponent@decimalfound
\prt@decimal
\fi
\if\prt@printafter{exponent}\
\fi
}% 10^{-}
\fi

\endgroup
\fi

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

\ifx\prt@oarg\prt@@empty
\else
All units expect the degree symbol are separated from the number. Detect whether the degree symbol is used.

The \textdegree command is not useable in math mode. Redeclate 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\MessageBreak 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

\PackageWarning{numprint}{The unit is typeset in mathmode. Use \string\tcdegree\space of\MessageBreak the mathcomp package or \string\degree\space of the\MessageBreak gensymb\MessageBreak package}{If you load the mathcomp package \string\textcelsius\space is substituted by \string\tcdegree. If you load the gensymb package \string\textcelsius\space is substituted by \string\celsius.}%

\endgroup
\def\textcelsius{%
\begingroup
\@ifundefined{tccelsius}{\@ifundefined{celsius}{%
\def\prt@PackageError####1####2{%\PackageError{numprint}{####1}{####2}}%}{\celsius}}{	ccelsius}%
\prt@PackageError{The unit is typeset in mathmode. Use \string\tcelsius\space of\MessageBreak the mathcomp package or \string\celsius\space of the\MessageBreak gensymb\MessageBreak package}{If you load the mathcomp package \string\textcelsius\space is substituted by \string\tcelsius. If you load the gensymb package \string\textcelsius\space is substituted by \string\celsius.}%
\endgroup
\begingroup
\@ifundefined{tcohm}{\@ifundefined{ohm}{%
\def\prt@PackageError####1####2{%\PackageError{numprint}{####1}{####2}}%}{\ohm}}{	cohm}%
\prt@PackageError{The unit is typeset in mathmode. Use \string\tcelsius\space of\MessageBreak the mathcomp package or \string\celsius\space of the\MessageBreak gensymb\MessageBreak package}{If you load the mathcomp package \string\textcelsius\space is substituted by \string\tcelsius. If you load the gensymb package \string\textcelsius\space is substituted by \string\celsius.}%
\endgroup
\def\textohm{%
\begingroup
\@ifundefined{tcohm}{\@ifundefined{ohm}{%
\def\prt@PackageError####1####2{%\PackageError{numprint}{####1}{####2}}%}{\ohm}}{	cohm}%
\prt@PackageError{The unit is typeset in mathmode. Use \string\tcelsius\space of\MessageBreak the mathcomp package or \string\celsius\space of the\MessageBreak gensymb\MessageBreak package}{If you load the mathcomp package \string\textcelsius\space is substituted by \string\tcelsius. If you load the gensymb package \string\textcelsius\space is substituted by \string\celsius.}%
\endgroup
\def\textmu{\begingroup\@ifundefined{tcmu}{\@ifundefined{mu}{%
\def\prt@PackageError####1####2{%\PackageError{numprint}{####1}{####2}}%}{\mu}}{	cmu}%
\prt@PackageError{The unit is typeset in mathmode. Use \string\tcelsius\space of\MessageBreak the mathcomp package or \string\celsius\space of the\MessageBreak gensymb\MessageBreak package}{If you load the mathcomp package \string\textcelsius\space is substituted by \string\tcelsius. If you load the gensymb package \string\textcelsius\space is substituted by \string\celsius.}%
\endgroup
\def\textperthousand{\begingroup\@ifundefined{tcperthousand}{\@ifundefined{perthousand}{%
\def\prt@PackageError####1####2{%\PackageError{numprint}{####1}{####2}}%}{\perthousand}}{	tperthousand}%
\prt@PackageError{The unit is typeset in mathmode. Use \string\tcelsius\space of\MessageBreak the mathcomp package or \string\celsius\space of the\MessageBreak gensymb\MessageBreak package}{If you load the mathcomp package \string\textcelsius\space is substituted by \string\tcelsius. If you load the gensymb package \string\textcelsius\space is substituted by \string\celsius.}%
\endgroup

55
\def\nprt\PackageError####1####2{%
  \PackageError{numprint}{####1}{####2}%
  }\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 package}{If you load the mathcomp package \string\textohm\space is substituted by\MessageBreak \string\tcohm. If you load the gensymb package \string\textohm\space is substituted by \string\ohm.}%\nendgroup}%\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 the mathcomp package}{If you load the mathcomp package \string\textmu\space is substituted by \string\tcmu.}%\nendgroup}%\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\space of \MessageBreak the mathcomp package or \string\perthousand\space of \MessageBreak the 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.}%\nendgroup}%
\def\nprt@tmpunit{\tcdegree}\
\ifx\nprt@oarg\nprt@tmpunit\ensuremath{\nprt@degreesep}\else
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{\degree}\
\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.
If the unit is \textdegree{} from the textcomp package, the unit is a degree sign. Then print the separator \npert@degreesep instead of \npert@unitsep.

\begin{verbatim}
\def\npert@tmpunit{\textdegree}\
\if\npert@oarg\npert@tmpunit
\ensuremath{\npert@degreesep}\
\else
If the unit is \%, the unit is a percent sign. Then print the separator \npert@percentsep instead of \npert@unitsep.
\def\npert@tmpunit{\%}\
\if\npert@oarg\npert@tmpunit
\ensuremath{\npert@percentsep}\
\else
Else, print \npert@unitsep.
\fi
\fi
\fi
\fi
\fi
\fi
Finally, print the unit.
\npunitcommand{\npert@oarg}\
\fi
\endgroup
\end{verbatim}

F.10 \textbf{Print lengths and counters}

\cntprint The macro for printing counters. The mandatory argument takes a \LaTeXX{} counter. The optional argument may contain a unit which is printed, too.
\begin{verbatim}
\newcounter{npert@cntprint}\
\DeclareRobustCommand*{\cntprint}[2]{\@empty}{% \\
\@ifundefined{c@#2}{\@nocounterr{#2}}{% \\
\numprint[#1]{\arabic{#2}}% \\
\fi\fi\fi
\endgroup
\end{verbatim}

\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.
\begin{verbatim}
\DeclareRobustCommand*{\lenprint}[2]{\@empty}{% \\
Do everything in a group to avoid macros to resist outside. \\
\begingroup
\Save the optional argument in \npert@oarg and expand it.
\edef\npert@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\npert@tmp{\@empty}%
\end{verbatim}
If the chosen unit is not defined produce an error message and fall back to pt.
\ifx\nprt@oarg\nprt@tmp\def\nprt@oarg{pt}\fi
\PackageError{numprint}{Invalid unit ‘#1’}{The requested unit is not defined using \string\npdefunit.}^^J
Using ‘pt’ instead.}
\def\nprt@oarg{pt}
\@ifundefined{nprt@factor@\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.}}%
\edef\nprt@oarg{\csname nprt@unit@\nprt@oarg\endcsname}
\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#3*
\else
\expandafter\def\csname nprt@factor@#1\endcsname{#3}\%
\fi
\expandafter\def\csname nprt@unit@#1\endcsname{#2}\%
}
\npdefunit{pt}{pt}{1.00000000000}
\npdefunit{bp}{bp}{0.99626400996}
58
F.11 Internationalization

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

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

\newcommand*{\npstyledefault}{\npthousandsep{,}\npproductsign{\cdot}\npunitseparator{,}\nppercentseparator{\prt@unitsep}}

\newcommand*{\npdefunit}{in}{in}{0.01383700013}
\newcommand*{\npdefunit}{ft}{ft}{0.00115308334}
\newcommand*{\npdefunit}{mm}{mm}{0.35145980351}
\newcommand*{\npdefunit}{cm}{cm}{0.03514598035}
\newcommand*{\npdefunit}{m}{m}{0.0003514598035}
\newcommand*{\npdefunit}{km}{km}{0.0000003514598035}
Sets 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{\pret@unitsep}%
}\n
\npstyleenglish Sets the parameters to English habit.
\newcommand*{\npstyleenglish}{%
\npthousandsep{,}%
\npdecimalsign{.}%
\npproductsign{\times}%
\npunitseparator{,}%
\npdegreeseparator{}%
\nppercentseparator{\pret@unitsep}%
}\n
\npstyleportuguese Sets the parameters to German habit.
\newcommand*{\npstyleportuguese}{%
\npthousandsep{,}%
\npdecimalsign{,}%
\npproductsign{\cdot}%
\npunitseparator{,}%
\npdegreeseparator{}%
\nppercentseparator{\pret@unitsep}%
}\n
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 compatibility reasons. Proceed only if it is switched on.
\ifnprt@autolanguage
\n Automatic language support only works with babel.
\ifpackageloaded(babel)%
\n Adds the language settings to the known languages if they are provided by babel.
The current version knows all German, English, and Portuguese dialects.
\n\npaddtolanguage{UKenglish}{english}%
\npaddtolanguage{USenglish}{english}%
\npaddtolanguage{american}{english}%
\npaddtolanguage{austrian}{german}%
\npaddtolanguage{british}{english}%
\npaddtolanguage{canadian}{english}%
\npaddtolanguage{english}{english}%
\npaddtolanguage{german}{german}%
\quad \text{Set the active language again to activate the extras section.}
\expandafter\selectlanguage\expandafter{\languagename}\%

\IfBabel\isnotloaded\but\automatichandles\languages
\If\selectlanguage{english}\english

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

\nprt@renameerror A command for producing an error message for redefined macros.
\expandafter\def\csname #1\endcsname{\PackageError{numprint}{This command has been renamed\MessageBreak to MessageBreak\string np #1}{In order to avoid problems with other\MessageBreak packages and for consistency, this\MessageBreak command has been renamed in this version.}\MessageBreak}\% 
\fourdigitsep \Define replacements for the old commands that produce error messages.
\fourdigitnosep \nprt@renameerror{fourdigitsep}
\addmissingzero \nprt@renameerror{addmissingzero}
\noaddmissingzero \nprt@renameerror{addmissingzero}
\nodigits \nprt@renameerror{nodigits}
\exponentdigits \nprt@renameerror{exponentdigits}
\noexponentdigits \nprt@renameerror{noexponentdigits}

\section*{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 \texttt{numprint032.dtx} or \texttt{README} to get it.

\subsection*{1.00}

General: Automatic support for different number formats in different...
ent languages .................. 8
Automatically don't separate degree sign from number .......... 4
Support for adding a plus to a number ......................... 6
Total new implementation .................. 1
\addmissingzero: Renamed to \npaddmissingzero ........... 61
\digits: Renamed to \npdigits .............. 61
\exponentdigits: Renamed to \npexponentdigits .......... 61
\fourdigitnosep: Renamed to \npfourdigitnosep .......... 61
\fourdigitsep: Renamed to \npfourdigitsep ............ 61
\naddmissingzero: Renamed to \npnaddmissingzero ........ 61
\ndigits: Renamed to \npdigits .................. 61
\noexponentdigits: Renamed to \npnoexponentdigits .... 61
\np: Add shortcut for \numprint 23

1.10 General: Avoid use of subst package .................. 21, 26
Define math version npbold 11, 34
New tabular alignment mechanism ......................... 8, 38
\npmakebox: Declare \npmakebox command ................... 34

1.11 General: Avoid use of \fileversion etc. .................. 1

1.12 General: Adapt Makefile to \TeXLive ...................... 1

1.13 General: Small fix in documentation ...................... 1

1.20 General: Add padding a number on the left side .......... 1
Allow \numprint{-e5} in addition to \numprint{e5} ........ 1
Remove the reduction in font sizes in documentation .......... 1

1.21 General: Add support for Portuguese ..................... 1

1.22 General: Use \numunitcommand to typeset units .......... 1
\nprt@printsing: Print sign in a group to avoid additional space for an operator (for nbaseprt) 44
\numprint: Enable to use \textcelsius, \textohm, \textmu, and \textperthousand in the unit .................. 55
Produce a warning rather than an error when using \textdegree as unit ........ 55
Restrain empty \color command to exponent .................. 53

1.30 General: Add the commands \lenprint and \cntprint that print lengths and counters ...... 1

1.31 \nprt@getnexttok: Bugfix: The numprint column types did not like empty cells in the last column of a tabular .................. 35
\numprint: Bugfix: \numprint{e123} reprinted the last used sign .. 49

1.32 General: Improve documentation about textcomp symbols ........ 4
\numprint: Produce an error rather than a warning when using textcomp symbols in the unit if mathcomp or gensymb is not loaded .................. 55

1.33 General: Avoid to use the calc package since it causes problems with many other packages ... 21

1.34 \nprt@calcblockwidth: Fix a bug invoked in version 1.33. Found by Stefan Salewski ........ 33

1.35 General: Support not to separate percent sign from number .......... 1
Index

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

Symbols
\% .......................... 1099
@firstoftwo  . 119, 1161
@ifnextchar  ............. 337, 461, 471, 478
@ifpackageloaded  1210
@ifundefined  . 618, 1028, 1044, 1060, 1072, 1114, 1125, 1133
@nloccounterr  .......... 1114
@nomath  ................. 352
@secondfoto  121, 1163
@temptoena  485, 489
^  .......................... 766

A
\active  .......................... 766
\addmissingzero  1245
\addto  .......................... 1168
\addtocounter  ............... 345, 653, 659
\addtolength  325, 328
\arabic  .......................... 1115
\AtBeginDocument  1208

C\c@nprt@blockcnt  .......................... 316, 654
\c@nprt@c@ntprint  658
\catcode  766
\c@dot  1178, 1187, 1203
\celsius  1031, 1034, 1039
\c@ntprint  5, 1112
\color  920
\columns@type\n  9, 13
\columns@type\n  9, 13

D\DeclareMathVersion  346
\DeclareOption  21–36, 38
\DeclareRobustCommand 336, 759, 1113, 1118
\degree  1020, 1091
\digits  1245
\displaystyle  827, 828, 861, 862, 930, 931, 967, 968
\divide  316, 654
\ifnprt@mantissa@decimalfound  . 149, 863, 870, 877, 884, 892, 899
\ifnprt@mantissa@fixeddigits  ....... 83, 84, 858
\ifnprt@mathb@ular  ............... 412, 442
\ifnprt@newcolumntype  ............ 11, 456
\ifnprt@newbold  12, 345
\ifnprt@numfound  ............... 358, 371, 439
\ifnprt@numsepfour  ............. 5, 317, 639, 726
\ifnprt@roundup  497, 514, 539, 572, 576
\ifnprt@shortnumber  ............ 629, 647, 741
\ignorespaces  486
\InputIfFileExists  1231
\language  1225
\lenprint  5, 1118
\loop  .......................... 53

M\makebox  338, 343, 828, 833, 838, 843, 849, 862, 869, 876, 883, 891, 931, 936, 941, 946, 952, 968, 975, 982, 989, 997
\mathchoice  826, 866, 929, 966
\mathversion  11, 353
\multicolumn  11
\multiply  658

N\NC@find  490
\NC@rewrite\n  458
\NC@rewrite\n  468
\NeedsTeXFormat  1
\newcolumntype  457, 467, 493, 494
\newtoks  355–357

63