CNTFORMATS

v0.5 2012/04/23

part of the ExSheets bundle

a different way to read counters

Clemens Niederberger

https://bitbucket.org/cgnieder/exsheets/
contact@mychemistry.eu

English documentation

Contents

1 Motivation

CNTFORMATS provides a way to format counters with what I will call patterns. This does not in any way effect the usual LaTeX 2_ε way of treating counters and does not use `\the<counter>` nor is it affected by the redefinition of them.

This package is aimed at package or class authors and probably not very useful for document authors.

When I first had the idea for this package the idea of what it does already existed as part of the ExSheets package. I can't recall why I came up with the idea in the first place or why I originally wanted a new syntax for formatting the `question` counter. Anyway, here we are.

2 License and Requirements

CNTFORMATS is placed under the terms of the LaTeX Project Public License, version 1.3 or later (`http://www.latex-project.org/lppl.txt`). It has the status "maintained."

CNTFORMATS requires the etoolbox[1] package.

3 Example

A use case typically looks as follows:

```
\ReadCounterPattern{se.sse}
```

3.0

where the key `se` stands for the current value of the `section` counter and `sse` for `subsection`, respectively. `se.sse` is an example for what will be called *pattern*. The keys for the counters can have optional arguments that specify the format:

```
\stepcounter{subsection}
\ReadCounterPattern{se[A](sse[r])}
```

C(i)

`A` stands for `\Alph` and `r` for `\roman`. A complete list can be found in table 2 on page 4. As you can see you can insert arbitrary other tokens in a pattern that won't be changed. It is important to notice, though, that the patterns are only replaced if they're *not* placed in a braced group!

```
\ReadCounterPattern{{se[A](sse[r])}}
```

se[A](sse[r])

I would imagine that the argument to `\ReadCounterPattern` is usually supplied by a user setting an option …

```
\somesetupcommand{
  counter-format = se[A](sse[r])
}
```

… and then internally used by the corresponding package or class.

4 Usage

In the following description of the available commands the symbol ▷ means that the command is expandable, ► means that it isn't.

In order to make counters known to CNTFORMATS the following commands are used:

► `\AddCounterPattern*[<module>]{<counter>}{<pattern>}`

This command will make the (existing) counter `<counter>` known to CNTFORMATS and assign the pattern `<pattern>` to it.

[1] CTAN: etoolbox

▶ `\NewCounterPattern*[<module>]{<counter>}{<pattern>}`
This command will create a new counter `<counter>`, make it known to CNTFORMATS and assign the pattern `<pattern>` to it.

▶ `\ReadCounterFrom[<module>]{<counter>}{<internal cmd>}`
If you use one of the commands above with the starred version the number for the pattern is not automatically fetched from the internal `\c@<counter>`. This can now be assigned with `\ReadCounterFrom` where `<internal cmd>` is the macro that holds the number.

The commands above can only be used in the document preamble.

After the creation of these pattern markers one wants to be able to use them. There are a number of macros that allow different aspects of usage.

▶ `\ReadCounterPattern[<module>]{<pattern>}`
Reads, interprets and prints a pattern.

▷ `\@cntfmts@parsed@pattern`
After `\ReadCounterPattern` has been used the current pattern interpretation is stored in this macro. The *interpretation* is *not* what is printed. See the examples below for details.

▶ `\ReadCounterPatternFrom[<module>]{<macro that holds pattern>}`
Reads, interprets and prints a pattern that's stored in a macro.
Otherwise the same as `\ReadCounterPattern`.

▶ `\SaveCounterPattern{<cmd a>}{<cmd b>}{<pattern>}`
Saves the `<pattern>` in `<cmd a>` and the interpreted pattern in `<cmd b>`.

▶ `\eSaveCounterPattern[<module>]{<cmd a>}{<cmd b>}{<pattern>}`
Saves the `<pattern>` in `<cmd a>` and the expanded pattern in `<cmd b>`.

▶ `\SaveCounterPatternFrom[<module>]{<cmd a>}{<cmd b>}{<macro that holds pattern>}`
Like `\SaveCounterPattern` but reads the pattern from a macro.

▶ `\eSaveCounterPatternFrom[<module>]{<cmd a>}{<cmd b>}{<macro that holds pattern>}`
Like `\eSaveCounterPattern` but reads the pattern from a macro.

The optional argument `<module>` should be specific for a package, say, so that different patterns for the `section` for example don't interfer with each other. If you leave the argument the default module `cntfmts` is used.

The ExSheets packages uses the commands with the module `exsheets`. You can find the following lines in ExSheets' code:

```
\AddCounterPattern*[exsheets]{section}{se}
\ReadCounterFrom[exsheets]{section} \l__exsheets_counter_sec_int
\NewCounterPattern*[exsheets]{question}{qu}
\ReadCounterFrom[exsheets]{question} \l__exsheets_counter_qu_int
```

Now let’s see a short example that hopefully explains what the macros do:

```
% preamble
% \NewCounterPattern{testa}{ta}
\setcounter{testa}{11}
\ReadCounterPattern{ta}
\ReadCounterPattern{ta[a]} \\
\ttfamily\makeatletter
\meaning\@cntfmts@parsed@pattern

\bigskip
\SaveCounterPattern\tmpa\tmpb{ta[a]}
\meaning\tmpa \\
\meaning\tmpb

\bigskip
\eSaveCounterPattern\tmpa\tmpb{ta[a]}
\meaning\tmpa \\
\meaning\tmpb
```

```
11 k
macro:->{}\csuse
{@cntfmts@read@ta@counter}[a]\relax

macro:->ta[a]
macro:->{}\csuse
{@cntfmts@read@ta@counter}[a]\relax

macro:->ta[a]
macro:->{}k\relax
```

You can see that somehow additional (empty) groups and a `\relax` found their way into the interpreted and thus the expanded pattern. This is due to the fact that reading optional arguments expandably isn’t easy and must have some safety net.

5 Predefined and New Patterns and Format Keys

5.1 Predefined Patterns and Format Keys

CNTFORMATS predefines a number of pattern keys. These are listed in table 1.

Table 1: Predefined Patterns for the module `cntfmts`.

counter	pattern
`chapter`	`ch`
`section`	`se`
`subsection`	`sse`
`subsubsection`	`ssse`
`paragraph`	`pg`

Table 2: Predefined Format Keys

key	format
`1`	`\arabic`
`a`	`\alph`
`A`	`\Alph`
`r`	`\roman`
`R`	`\Roman`

5.2 New Patterns and Format Keys

Table 2 lists the predefined formats. If you want you can add own formats.

► `\NewPatternFormat{<pattern>}{<format>}`

`<format>` is a number presentation command like `\@alph`, *i.e.*, it needs a mandatory argument that takes a number. It is used in `<format>` *without* its argument. This command can only be used in the preamble.

Here are now a few examples of possible new patterns. Suppose the following code in the preamble:

```
\usepackage{alphalph,fmtcount}
\newcommand*\myoddnumber[1]{\the\numexpr2*(#1)-1\relax}

\NewPatternFormat{aa}{\alphalph}
\NewPatternFormat{o}{\ordinalnum}
\NewPatternFormat{x}{\myoddnumber}

\newcounter{testa}
\NewCounterPattern{testa}{ta}
\setcounter{testa}{4}
```

Then we can use the new pattern and the new formats as follows:

```
\ReadCounterPattern{ta[aa]}
\ReadCounterPattern{ta[o]}
\ReadCounterPattern{ta[x]}
```

d 4th 7

6 Implementation

In the following code the lines 1–30 have been omitted. They only repeat the license statement which has already been mentioned in section 2.

```
31 \def\@cntfmts@date{2012/04/23}
32 \def\@cntfmts@version{v0.5}
33
34 \ProvidesPackage{cntformats}[\@cntfmts@date\space \@cntfmts@version\space A
     different way to read counters.]
35 \RequirePackage{etoolbox}
36 % -------------------------------------------------------------------------
37 % message handling
38 \def\@cntfmts@error@message{%
39   For details have a look at the ‘exsheets’ manual.}
40
41 \def\@cntfmts@create@message#1{%
42   \ifstrequal{#1}{Error}
43     {%
44       \lowercase{\csdef{@cntfmts@#1}}##1{%
45         \csuse{Package#1}{cntformats}{##1}{\@cntfmts@error@message}}%
46     }{%
47       \lowercase{\csdef{@cntfmts@#1}}##1{%
48         \csuse{Package#1}{cntformats}{##1}}%
49     }}
50 \@cntfmts@create@message{Error}
```

```
\@cntfmts@create@message{Warning}
\@cntfmts@create@message{WarningNoLine}
\@cntfmts@create@message{Info}

\def\@cntfmts@err@already@defined#1{%
  \@cntfmts@error{The counter ‘#1’ already exists!}}
\def\@cntfmts@err@unknown@counter#1{%
  \@cntfmts@error{The counter ‘#1’ is undefined!}}
\def\@cntfmts@err@pattern@defined#1#2{%
  \@cntfmts@error{The pattern ‘#2’ for module ‘#1’ is already defined!}}
\def\@cntfmts@err@pattern@undefined#1#2{%
  \@cntfmts@error{The pattern ‘#2’ for module ‘#1’ is undefined!}}

% ---------------------------------------------------------------------------
% tokenlist analyzing and manipulating
%   if you happen to read this code and notice some obvious reasons why these
%   macros shoudn’t be used like this, please send me your improved version
%   (seriously! But please explain why yours is better)
\long\def\@cntfmts@ifintl#1#2{%
  \def\@cntfmts@intl@tmpa##1#2{}%
  \expandafter\if\expandafter\relax\expandafter
  \detokenize\expandafter{\@cntfmts@intl@tmpa#1{}{}#2}\relax%
    \expandafter\@secondoftwo
  \else
    \expandafter\@firstoftwo
  \fi
}
\long\def\@cntfmts@ifintlcs#1{%
  \expandafter\@cntfmts@ifintl\expandafter{#1}}

\long\def\@cntfmts@replaceonceincs#1#2#3{%
  \@cntfmts@ifintlcs#1{#2}%
    {\@cntfmts@replace@once@incs#1{#2}{#3}}{}%
}
\long\def\@cntfmts@replace@once@incs#1#2#3{%
  \def\@cntfmts@replo@tmpa##1#2##2\q@stop{%
    \def#1{##1#3##2}}%
  \expandafter\@cntfmts@replo@tmpa#1\q@stop
}

\long\def\@cntfmts@replaceallin#1#2#3{%
  \def\@cntfmts@repla@tmpa{#1}%
  \@cntfmts@replaceonceincs\@cntfmts@repla@tmpa{#2}{#3}%
  \@cntfmts@ifintlcs\@cntfmts@repla@tmpa{#2}%
    {\expandafter\@cntfmts@replaceallin\expandafter{\@cntfmts@repla@tmpa
  }{#2}{#3}}%
    {\expandonce\@cntfmts@repla@tmpa}%
}
\long\def\@cntfmts@replaceallincs#1#2#3{%
```

```
  \@cntfmts@replaceonceincs#1{#2}{#3}%
  \@cntfmts@ifintlcs#1{#2}{\@cntfmts@replaceallincs#1{#2}{#3}}{}%
}

% ------------------------------------------------------------------------------
% expansion tools
%   heavily inspired by expl3’s \exp_args:N<spec> -- one might say: copied
\long\def\@cntfmts@getnextbraced#1#2#3{#2\@cntfmts@firstofone{#3{#1}}}
\long\def\@cntfmts@firstofone#1{#1}
\long\def\@cntfmts@braced@unexpanded#1\@cntfmts@firstofone#2#3{%
  #1\@cntfmts@firstofone{#2{#3}}}
\long\def\@cntfmts@braced@expandedonce#1\@cntfmts@firstofone#2#3{%
  \expandafter\@cntfmts@getnextbraced\expandafter{#3}{#1}{#2}}

\long\def\@cntfmts@expand@first{%
  \@cntfmts@braced@expandedonce
  \@cntfmts@firstofone}

\long\def\@cntfmts@expand@second{%
  \@cntfmts@braced@unexpanded
  \@cntfmts@braced@expandedonce
  \@cntfmts@firstofone}

\long\def\@cntfmts@expand@third{%
  \@cntfmts@braced@unexpanded
  \@cntfmts@braced@unexpanded
  \@cntfmts@braced@expandedonce
  \@cntfmts@firstofone}

% ------------------------------------------------------------------------------
% expandable commands with optional argument (and no mandatory)
%   this is probably not very robust. If you know what you’re doing you’re
%   probably able to define a better version yourself. If not, don’t use it
\long\def\@cntfmts@isopt@#1[#2\q@marker#3#{%
  \if\relax\detokenize{#1#2}\relax
    \expandafter\@firstoftwo
  \else
    \expandafter\@secondoftwo
  \fi}
\long\def\@cntfmts@isopt#1#2#3{%
  \@cntfmts@isopt@#1\q@marker[\q@marker{#2#1}{#2[{#3}]{#1}}}

\long\def\@cntfmts@defwopt#1[#2]#3{%
  \long\def#1##1{\@cntfmts@isopt{##1}{\csuse{\@cntfmts@strip@bs#1@opt}}{#2}}%
  \long\csdef{\@cntfmts@strip@bs#1@opt}[##1]{#3}}

\def\@cntfmts@strip@bs{\expandafter\@gobble\string}

% ------------------------------------------------------------------------------
% #1: module
% #2: counter
\def\@cntfmts@def@counter@read#1#2#3{%
  \@cntfmts@expand@second\csdef
    {@#1@read@#3@counter}%
    {\csname @#1@read@#2@counter\endcsname}}

% create pattern for an existing counter:
% #1: module
% #2: counter
% #3: id
\newcommand*\@cntfmts@add@counter@pattern[3][cntfmts]{%
  \ifcsdef{c@#2}
    {}{\@cntfmts@err@unknown@counter{#2}}%
  \ifcsdef{@#1@#2@counter}
    {\@cntfmts@err@pattern@defined{#1}{#2}}
    {\csdef{@#1@#2@counter}{}}%
  \listcsadd{@#1@counter@ids}{#3}%
  \@cntfmts@def@counter@read{#1}{#2}{#3}%
}

\newcommand*\@cntfmts@add@counter@pattern@nostar[3][cntfmts]{%
  \@cntfmts@add@counter@pattern[#1]{#2}{#3}%
  \@cntfmts@expand@third
    \@cntfmts@create@read@counter{#1}{#2}{\csname c@#2\endcsname}}

\newrobustcmd*\AddCounterPattern{%
  \@ifstar
    {\@cntfmts@add@counter@pattern}
    {\@cntfmts@add@counter@pattern@nostar}}
\@onlypreamble\AddCounterPattern

% create pattern for a new counter
% #1: counter
% #2: id
\newcommand*\@cntfmts@new@counter@pattern[3][cntfmts]{%
  \ifcsdef{the#2}
    {\@cntfmts@err@counter@defined{#2}}{}%
  \newcounter{#2}%
  \listcsadd{@#1@counter@ids}{#3}%
  \@cntfmts@def@counter@read{#1}{#2}{#3}%
  \csdef{@#1@#2@counter}{}
}

\newcommand*\@cntfmts@new@counter@pattern@nostar[3][cntfmts]{%
  \@cntfmts@new@counter@pattern[#1]{#2}{#3}%
  \@cntfmts@expand@third
    \@cntfmts@create@read@counter{#1}{#2}{\csname c@#2\endcsname}}

```

```
\newrobustcmd*\NewCounterPattern{%
  \@ifstar
    {\@cntfmts@new@counter@pattern}
    {\@cntfmts@new@counter@pattern@nostar}}
\@onlypreamble\NewCounterPattern

% renew pattern:
\newcommand*\@cntfmts@renew@counter@pattern[3][cntfmts]{%
  \ifcsdef{the#2}
    {}{\@cntfmts@err@unknown@counter{#2}}%
  \ifcsdef{@#1@#2@counter}{}
    {\@cntfmts@err@pattern@undefined{#1}{#2}}%
  \ifinlistcs{@#1@counter@ids}{#3}
    {}{\listcsadd{@#1@counter@ids}{#3}}%
  \@cntfmts@def@counter@read{#1}{#2}{#3}%
}

\newcommand*\@cntfmts@renew@counter@pattern@nostar[3][cntfmts]{%
  \@cntfmts@renew@counter@pattern[#1]{#2}{#3}%
  \@cntfmts@expand@third
    \@cntfmts@create@read@counter{#1}{#2}{\csname c@#2\endcsname}}

\newrobustcmd\RenewCounterPattern{%
  \@ifstar
    {\@cntfmts@renew@counter@pattern}
    {\@cntfmts@renew@counter@pattern@nostar}}
\@onlypreamble\RenewCounterPattern

% ----------------------------------------------------------------------------
% assign a different internal command to <counter> than \c@<counter>
% can/must be used after a call from \AddCounterPattern* or
% \NewCounterPattern*
% #1: module
% #2: counter
% #3: internal command
\def\@cntfmts@create@read@counter#1#2#3{%
  \expandafter\@cntfmts@defwopt\csname @#1@read@#2@counter\endcsname[]{%
    \ifblank{##1}
      {\@arabic{#3}}
      {\csuse{@cntfmts@counter@type@##1}{#3}}%
  }%
}

\newrobustcmd*\ReadCounterFrom[3][cntfmts]{%
  \@cntfmts@create@read@counter{#1}{#2}{#3}}

% ----------------------------------------------------------------------------
% pattern formats
% #1: key
% #2: number presentation command like \@alph
\def\@cntfmts@new@pattern@format#1#2{%
  \csdef{@cntfmts@counter@type@#1}{#2}}

\newrobustcmd*\NewPatternFormat[2]{%
  \@cntfmts@new@pattern@format{#1}{#2}}
\@onlypreamble\NewPatternFormat

% ---------------------------------------------------------------------------
% interpret and print pattern:
% #1: module
% #2: list
% #3: pattern
\def\@cntfmts@read@counter@pattern#1#2#3{%
  \def\@cntfmts@parsed@pattern{#3\relax}%
  \forlistcsloop{\@cntfmts@replace@pattern{#1}}{#2}%
}
% #1: module
% #2: pattern-key
\def\@cntfmts@replace@pattern#1#2{%
  \@cntfmts@replaceallincs\@cntfmts@parsed@pattern
    {#2}{{}\csuse{@#1@read@#2@counter}}}

\newrobustcmd*\ReadCounterPattern[2][cntfmts]{%
  \@cntfmts@read@counter@pattern{#1}{@#1@counter@ids}{#2}\
 @cntfmts@parsed@pattern}

\newrobustcmd*\ReadCounterPatternFrom[2][cntfmts]{%
  \@cntfmts@expand@third\@cntfmts@read@counter@pattern{#1}{@#1@counter@ids}{#2}
 %
  \@cntfmts@parsed@pattern}

% #1: save pattern in
% #2: save interpretation in
% #3: pattern
% #4: module
\newcommand*\@cntfmts@save@counter@pattern[4]{%
  \@cntfmts@read@counter@pattern{#4}{@#4@counter@ids}{#3}%
  \let#2\@cntfmts@parsed@pattern
  \def#1{#3}%
}

\newcommand*\@cntfmts@esave@counter@pattern[4]{%
  \@cntfmts@read@counter@pattern{#4}{@#4@counter@ids}{#3}%
  \edef#2{\@cntfmts@parsed@pattern}%
  \def#1{#3}%
}

\newrobustcmd*\SaveCounterPattern[4][cntfmts]{%
```

```
  \@cntfmts@save@counter@pattern{#2}{#3}{#4}{#1}}

\newrobustcmd*\SaveCounterPatternFrom[4][cntfmts]{%
  \@cntfmts@expand@third\@cntfmts@save@counter@pattern{#2}{#3}{#4}{#1}}

\newrobustcmd*\eSaveCounterPattern[4][cntfmts]{%
  \@cntfmts@esave@counter@pattern{#2}{#3}{#4}{#1}}

\newrobustcmd*\eSaveCounterPatternFrom[4][cntfmts]{%
  \@cntfmts@expand@third\@cntfmts@esave@counter@pattern{#2}{#3}{#4}{#1}}

% ------------------------------------------------------------------------------
% predefined formats and pattern
\NewPatternFormat{a}{\@alph}
\NewPatternFormat{A}{\@Alph}
\NewPatternFormat{1}{\@arabic}
\NewPatternFormat{r}{\@roman}
\NewPatternFormat{R}{\@Roman}

\ifdef\c@paragraph
  {\AddCounterPattern{paragraph}{pg}}{}
\ifdef\c@subsubsection
  {\AddCounterPattern{subsubsection}{ssse}}{}
\ifdef\c@subsection
  {\AddCounterPattern{subsection}{sse}}{}
\ifdef\c@section
  {\AddCounterPattern{section}{se}}{}
\ifdef\c@chapter
  {\AddCounterPattern{chapter}{ch}}{}

\endinput

% HISTORY:
2012/09/30 v0.2beta - first version (as part of the 'exsheets' bundle)
2012/11/08 v0.4     - stepped number with 'exsheets' until now; next stepping
                      won't synchronize but will step to whatever deems
                      appropriate
2012/04/23 v0.5     - changed tests for heading commands to test explicitly for
                      the associated counters
                    - change test for counter in \@cntfmts@add@counter@pattern
                      from \the<ounter> to \c@<ounter>
                    - rename tokenlist test macros to use cntformats'
                      namespace
```

Index

Section titles are indicated **bold**, packages sans serif, commands \brown and options yellow.