The chemcompounds package*

Stephan Schenk
mail (at) schenk-stephan.de

December 1, 2006

Abstract

The `chemcompounds.dtx` package allows for a simple consecutive numbering of chemical compounds. Optionally, it is possible to supply a custom name for each compound. By default the compounds are numbered following the order of their appearance in the text.

Contents

1 Introduction

In chemical publications it is often necessary to consecutively number every compound mentioned in the text. Although this can be simply accomplished by manually inserting the corresponding numbers into the text, it is generally much more tedious work since the numbering scheme tends to change several times during the evolvement of the manuscript. For this reason it would be nice to have an automaticism which will take care of every change.

Being myself a chemist, I've been using the `chemcono` package by Stefan Schulz for this purpose quite successfully over several years. This package creates a library very similar to `thebibliography`. Users can now refer to entries in the library by a command very similar to `\cite`. Thus, once you change the library entry every reference to it will be updated automatically upon running LaTeX on the file. There is only one issue associated with this package: You get a list of all declared compounds inside your document wich at least looks odd. I therefore decided to write a new package `chemcompounds` described in this document to address this problem.

When taking a closer look at the `chemcono` package, I realised that the only thing one has to do is to get rid of everything which produces text. Thus, as a basis I used the mechanism of `\bibitem` and `\cite` in pretty much the same way as `chemcono` does by extracting the corresponding code from `article.cls`

*This file describes version v1.1.3 and has been last revised 2006/12/01.

and `latex.ltx` but deleting any unnecessary commands producing output. I also introduced several lines of code to make the printing of the compound names more customisable.

Currently, the packages knows two different modes of operation. In the default `implicit` mode, a compound name is created automatically as a consecutive number when the compound is referenced for the first time by `\compound`. Thus, all compounds will be numbered consecutively in the order they appear in the text. If the automatically generated name is not appropriate, a custom name can be given to a compound by means of the `\declarecompound` command described below.

In `noimplicit` mode names are not generated fully automatically. Instead, for every compound a `\declarecompound` must be issued. This will again create a subsequent number and a custom name can be given as an optional argument. The main difference to `implicit` mode is that thus compounds will be numbered in the order of the corresponding `\declarecompound` commands rather than in the order of their appearance in the text.

2 The user interface

Because of the way the implementation works, two LaTeX runs are required to get everything right. This should not be a problem since you have to run LaTeX twice anyway to get the table of contents and references right. The package will issue a "labels have changed" warning if you have to rerun LaTeX. For every unknown compound name the package will issue a warning, too.

2.1 Package options

`implicit` This option causes the package to operate in `implicit` mode. This is the default.

`noimplicit` This option is the opposite of `implicit` and causes the package to operate in `noimplicit` mode.

2.2 Assigning and accessing compound names

`\declarecompound` `\declarecompound[`⟨*optional name*⟩`]{`⟨*label*⟩`}`
assigns a name to a compound. If the optional argument is omitted, a consecutive number is automatically taken as compound name. This command can only occur in the preamble. A personal recommendation is to keep all `\declarecompound`s together in a separate file and `\input` this file in the preamble.

In `implicit` mode, if no optional argument is given, the command does nothing since the automatic compound name will be generated by `\compound`.

`\compound` `\compound{`⟨*label1*⟩`,`⟨*label2*⟩`,...}`
prints the name of a compound. If a list of labels is given as argument, a list of names separated by `\compoundseparator` is created. In `implicit` mode this command also creates a new compound name if the label is used for the first time and a custom name has not already been assigned to this compound.

`\compound*` The starred version works in almost exactly the same way as `\compound`. The only difference is that it does not create any output at all. However, it still creates

the label in `implicit` mode. It can thus be used to create "hidden" compounds in `implicit` mode. This, i.e., can be useful if some compounds are depicted in an illustration or scheme which are only later or even never mentioned in the text but the numbering scheme should take care of them.

`\compound+` The version with a '+' really prints only the name of a compound. In `implicit` mode, no label is created. Thus, it is the opposite of the starred command.

2.3 Customization

The commands in this section can be used to fine-tune the appearance of the compound names. In order to change the default behaviour you have to `\renewcommand` the corresponding commands. The defaults for every command are given in parentheses.

`\compoundseparator` `\compoundseparator` (`,\penalty\@m\␣`)
defines the separator in a list of compound names.

`\compoundglobalprefix` `\compoundglobalprefix` ()
defines the prefix for a list of compound names. This will be printed also in case the list has length one.

`\compoundglobalsuffix` `\compoundglobalsuffix` ()
defines the suffix for a list of compound names. This will be printed also in case the list has length one.

`\compoundprefix` `\compoundprefix` ()
defines the prefix for every compound.

`\compoundsuffix` `\compoundsuffix` ()
defines the suffix for every compound.

`\compoundstyle` `\compoundstyle` (`\textbf`)
defines the style of each name.

`\printcompound` `\printcompound` (`{\compoundprefix}{\compoundstyle{#1}}{\compoundsuffix}`)
is used to actually format the name of each compound. If the previous possibilities are not sufficient to meet your formatting demands the thing you should redefine is this one.

2.4 Examples

The following examples using

```
\declarecompound{label1}
\declarecompound{label2}
\declarecompound[5b]{label3}
\compound{label1} and \compound{label1,label2,label3}
```

should clarify the meaning of the above commands. The first two `\declarecommand`s could be omitted in `implicit` mode.

- Using the defaults results in **1** and **1**, **2**, **5b**.

- `\renewcommand{\compoundstyle}{\underbar}`

  gives 1 and 1, 2, 5b.

- `\renewcommand{\compoundseparator}{;}`

  gives **1** and **1**; **2**; **5b**.

- ```
  \renewcommand{\compoundglobalprefix}{(}
  \renewcommand{\compoundglobalsuffix}{)}
  ```

  gives (**1**) and (**1**, **2**, **5b**).

- ```
  \renewcommand{\compoundprefix}{(}
  \renewcommand{\compoundsuffix}{)}
  ```

  gives (**1**) and (**1**), (**2**), (**5b**).

- ```
  \renewcommand{\compoundglobalprefix}{\textbf{[}}
  \renewcommand{\compoundglobalsuffix}{\textbf{]}}
  \renewcommand{\compoundprefix}{\ensuremath{\langle}}
  \renewcommand{\compoundsuffix}{\ensuremath{\rangle}}
  \renewcommand{\compoundstyle}{\emph}
  ```

  gives [⟨*1*⟩] and [⟨*1*⟩, ⟨*2*⟩, ⟨*5b*⟩].

As shown above customization is not limited solely to parentheses etc. but can include formating commands, too.

3 The implementation

3.1 Identification

The package identifies itself at the top using something like

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{chemcompounds}
  [\filedate\space \fileversion\space Dictionary for compound numbering]
```

3.2 Package options

`\ifchemcompounds@implicit` Define a new boolean variable defining whether `implicit` mode is enabled.

```
\newif\ifchemcompounds@implicit
```

`implicit` `noimplicit` The following package options set `ifchemcompounds@implicit` either to true or false. The default is `implicit` mode.

```
\DeclareOption{implicit}{\chemcompounds@implicittrue}
\DeclareOption{noimplicit}{\chemcompounds@implicitfalse}
\ExecuteOptions{implicit}
```

Process options.

```
\ProcessOptions
```
```

3.3 User interface

The work flow for creating and accessing compound names was borrowed from the definition of `\bibitem` and `\cite`.

`\compoundseparator` `\compoundglobalprefix` `\compoundglobalsuffix` `\compoundprefix` `\compoundsuffix` `\compoundstyle` `\printcompound`

The following definitions define the default layout of the names in the text (no surrounding parentheses, multiple compound names separated by comma, names in bold face). `\printcompound` defines the way each name is printed.

```
\def\compoundseparator{,\penalty\@m\ }
\let\compoundglobalprefix\@empty
\let\compoundglobalsuffix\@empty
\let\compoundprefix\@empty
\let\compoundsuffix\@empty
\def\compoundstyle{\textbf}
\def\printcompound#1{{\compoundprefix}{\compoundstyle{#1}}{\compoundsuffix}}
```

`\declarecompound`

This command is used to assign a name to a compound. It just looks ahead whether an optional argument was given and calls the appropriate internal command. To avoid problems with the creation of the labels, this command is only allowed in the preamble.

```
\def\declarecompound{\@ifnextchar[\@ldeclarecompound\@declarecompound}
\@onlypreamble\declarecompound
```

`\ifchemcompounds@print`

Define a new boolean variable indicating whether the starred version of `\compound` was used.

```
\newif\ifchemcompounds@print
```

`\ifchemcompounds@create`

Define a new boolean variable indicating whether a label name shall be created automatically in `implicit` mode.

```
\newif\ifchemcompounds@create
```

`\compound` `\compound*` `\compound+`

This command will finally print the name associated with a compound label. The starred version just creates the label (in `implicit` mode) without printing the value. The command itself just checks whether the starred or plussed version is used, sets the internal flags appropriately and calls the internal command `\@compound`.

```
\DeclareRobustCommand{\compound}{%
  \chemcompounds@createtrue
  \chemcompounds@printtrue
  \@ifnextchar *{\chemcompounds@printfalse\@firstoftwo\@compound}
    {%
      \@ifnextchar +{\chemcompounds@createfalse\@firstoftwo\@compound}
        {\@compound}
    }
}
```

3.4 Internal commands

`\@compound`

This command retrieves the name associated with a compound and prints it in the text using the previously defined format. The code is a modified version of the definition of `\cite` in `latex.ltx`. If this command is invoked by `\compound*`, `\ifchemcompounds@print` will be false and all output will be suppressed.

```
\def\@compound#1{%
```

Print optional prefix to a list of compounds.

```
\ifchemcompounds@print
  \compoundglobalprefix
\fi
```

Now loop over every label in the argument list.

```
\let\@compounda\@empty
\@for\@compoundb:=#1\do{%
  \edef\@compoundb{\expandafter\@firstofone\@compoundb}%
```

Print separator. Note that it is empty for the first entry and `\compoundseparator` otherwise.

```
  \ifchemcompounds@print
    \@compounda
    \def\@compounda{{\compoundseparator}}%
  \fi
```

If compound is undefined, print '**?**' and raise a warning.

```
  \@ifcompoundundefined{\@compoundb}{%
    \ifchemcompounds@print
      \mbox{\reset@font\bfseries ?}%
    \fi
    \G@refundefinedtrue
    \@latex@warning
      {compound `\@compoundb' on page \thepage\space undefined}%
  }{%
```

If compound is known print formatted name.

```
    \ifchemcompounds@print
      \mbox{\printcompound{\@nameuse{comp@\@compoundb}}}%
    \fi
  }%
```

In `implicit` mode `\@createcompoundhook` will generate a new name if this has not been done before. In `noimplicit` mode this does nothing.

```
  \@createcompoundhook{\@compoundb}%
}%
```

Print optional suffix to a list of compounds.

```
\ifchemcompounds@print
  \compoundglobalsuffix
```

Although nothing is printed, under certain conditions an additional space is created. Remove it.

```
\else
  \unskip
\fi
}
```

`\chemcompounds@counter` Define a new counter which will be used for generating the compound names.

```
\newcounter{chemcompounds@counter}
```

`\chemcompounds@label` The next command will be used in the `.aux` file and defines a new label for every compound.

```
\def\chemcompounds@label{\@newl@bel {comp}}
```

\chemcompounds@writelabel Write the label and its value to the aux file.

```
\def\chemcompounds@writelabel#1#2{%
  \if@filesw
    \begingroup
      \def\protect{\noexpand}%
      \immediate\write\@auxout{
        \string\chemcompounds@label{#1}{#2}
      }%
    \endgroup
  \fi
  \ignorespaces
}
```

\@ldeclarecompound The next command gets called if an additional argument was supplied to \declarecompound. It creates the compound with the given name as soon as the aux file is writeable. This command can only be used in the preamble.

```
\def\@ldeclarecompound[#1]#2{%
  \AtBeginDocument{\@createcompound[#1]{#2}}
}
\@onlypreamble\@ldeclarecompound
```

\@declarecompound In implicit mode this command does nothing since default names are created automatically by \compound.

```
\ifchemcompounds@implicit
  \let\@declarecompound\@gobble
```

In noimplicit mode this simply creates the compound as soon as the aux file is writeable.

```
\else
  \def\@declarecompound#1{%
    \AtBeginDocument{\@createcompound{#1}}
  }
\fi
\@onlypreamble\@declarecompound
```

This command can only be used in the preamble.

\@ifcompoundundefined Test whether a compound has already been defined by testing the associated label.

```
\def\@ifcompoundundefined#1{%
  \@ifundefined{comp@#1}
}
```

\@createcompound This command is used to create a new compound name. It just looks ahead whether an optional argument was given and calls the appropriate command.

```
\def\@createcompound{%
  \@ifnextchar[\@@lcreatecompound\@@createcompound
}
```

\@@createcompound If a compound name has not yet been created this command increments chemcompounds@counter and takes the new value as the compound name. The new compound name is written to the aux file and a flag is set to indicate that a name for this compound has already been created.

```
\def\@@createcompound#1{%
```

```
  \@ifnotcompoundcreated{#1}{%
    \stepcounter{chemcompounds@counter}%
    \chemcompounds@writelabel{#1}{\the\value{chemcompounds@counter}}
    \@compoundcreated{#1}%
  }%
}
```

`\@@lcreatecompound` This command creates a compound name from the first parameter and writes it to the `aux` file. A flag is set to indicate that a name for this compound has already been created.

```
\def\@@lcreatecompound[#1]#2{%
  \chemcompounds@writelabel{#2}{#1}
  \@compoundcreated{#2}%
}
```

`\@compoundcreated` Set a flag indicating that the compound name has been created. This is done by defining an appropriate label in `implicit` mode.

```
\ifchemcompounds@implicit
  \def\@compoundcreated#1{%
    \global\@namedef{compc@#1}{}%
  }
```

In `noimplicit` mode this is unnecessary, thus just gobble the argument.

```
\else
  \let\@compoundcreated\@gobble
\fi
```

`\@ifnotcompoundcreated` Check whether a new compound name has already been created. In `implicit` mode existence of the corresponding label (flag) is checked. If it does not exist, the code given as the second argument is executed.

```
\ifchemcompounds@implicit
  \def\@ifnotcompoundcreated#1#2{%
    \@ifundefined{compc@#1}{#2}{}%
  }
```

In `noimplicit` mode the label (flag) is unused and a new name will always be. created. Therefore just execute the code given as second argument.

```
\else
  \let\@ifnotcompoundcreated\@secondoftwo
\fi
```

`\@createcompoundhook` This command gets called everytime a compound name is printed. In `implicit` mode this command creates a new compound name if `\ifchemcompounds@create` is true. The `\compound+` command sets this boolean to false.

```
\ifchemcompounds@implicit
  \def\@createcompoundhook#1{%
    \ifchemcompounds@create
      \@createcompound{#1}%
    \fi
  }
```

In `noimplicit` mode this is unnecessary. Therefore just gobble the argument.

```
\else
  \let\@createcompoundhook\@gobble
\fi
```

125 \endinput

Change History