CHEMMACROS

v5.3 2016/01/13

comprehensive support for typesetting chemistry documents

Clemens NIEDERBERGER

http://www.mychemistry.eu/forums/forum/chemmacros/

contact@mychemistry.eu

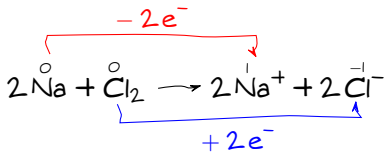


Table of Contents

Part I.
Preliminaries

1. Licence, Requirements and README

Permission is granted to copy, distribute and/or modify this software under the terms of the LaTeX Project Public License (LPPL), version 1.3 or later (http://www.latex-project.org/lppl.txt). The software has the status "maintained."

CHEMMACROS loads the packages expl3 [L3Pa] and xparse [L3Pb]. Depending on your usage other packages will be loaded. They are mentioned when the corresponding module using the package is described.

2. Motivation and Background

This package grew from a small collection of personal helper macros back in 2010 into a rather big package supporting various different chemical typesetting tasks. I hope I have achieved the

following points with this package:

- Intuitive usage as far as the syntax of the commands is concerned.
- A comprehensive set of macros! If there are any needs you might have with respect to typesetting of chemistry which is not supported by this package[1] then let me know so CHEMMACROS can be extended.
- The commands shall not only make typesetting easier and faster but also the document source more readable with respect to semantics (`\ortho-dichlorobenzene` is easier to read and understand than `\textit{o}-dichlorobenzene`); the first variant in my opinion also is more in the spirit of LaTeX 2_ε.
- As much customizability as I could think of so every user can adapt the commands to his or her own wishes. Every now and then users have wishes which can't be solved with the available options. Almost always I'll add options then. If you find something please contact me, see section B starting on page 62.
- Default settings that are compliant with the recommendations of the International Union of Pure and Applied Chemistry (IUPAC).

Especially the last point in the past needed some pushing from users to get things right in many places. If you find anything not compliant with IUPAC recommendations please contact me, see section B starting on page 62. Don't forget to add references for the corresponding IUPAC recommendation.

3. The Structure of CHEMMACROS

3.1. General Structure

Introduced in version 5.0

Since version 5.0 the CHEMMACROS package has a strictly modular structure. On the one hand this eases maintenance but it will also allow for easy and quick extension in the future. In a way it is a logical consequence from CHEMMACROS' history: since version 2.0, *i. e.*, since the fall of 2011 CHEMMACROS already had modular options.

The different modules of CHEMMACROS are divided into two groups:

1. Internal modules which provide underlying functionality or basic functionality which is not of direct interest from a user perspective but might be if you plan to write a module yourself (see section A for details).
2. User modules which provide all the stuff for typesetting.

Both groups each are subdivided into two groups: preloaded modules and modules which have to be loaded by the programmer (internal modules) or by the document author (user modules). Those modules are described in parts II (preloaded modules) and III (additional modules) of this manual.

The above means that not all functionality is available per default. If you want to load *all* modules no matter what then you can say:

1. Not including needs already solved by other packages such as chemnum or chemfig.

```
\usechemmodule{all}
```

or

```
\chemsetup{modules=all}
```

which will load all modules which are part of CHEMMACROS (also see section 7.1 starting on page 30). Own modules (see section A starting on page 61) are *not* loaded through this, though, and still have to be loaded additionally.

In part II starting on page 8 the preloaded modules are described, first the user modules then the internal ones, in part III starting on page 30 the other available modules are described, again the user modules first. In each section the modules are described in an alphabetical order.

3.2. Using CHEMMACROS' Options

Prior to v5.0 CHEMMACROS had quite a number of package options. CHEMMACROS v5.0 or higher has none! All of CHEMMACROS' options are set using the command

`\chemsetup[`⟨*module*⟩`]{`⟨*option list*⟩`}`
CHEMMACROS' setup command.

When an option is described then in the left margin the module the option belongs to is denoted. This looks something like this:

`module` » `option` = {⟨*value*⟩} (initially empty)
Description of `option`. The module is printed in the left margin. The default value to the right is the setting the option has when CHEMMACROS is loaded. This can be an explicit setting but the option can also be empty.

`module` » `choice-option` = `list|of|choices` Default: `list`
Description of `choice-option`. A choice option can only be used with a predefined list of values. If one of the values is underlined it means that the option can be used without value in which case the underlined value is chosen. If no value is underlined then a value *has* to be given by the user.

`module` » `boolean-option` = `true|false` Default: `true`
Description of `boolean-option`. A boolean option is a choice option with exactly the two values `true` and `false`. If the option is called without value then the underlined value is chosen (which is always `true` for a boolean option).

An option or list of options belonging to a module `module` can be set in two ways:

```
% first possibility:
\chemsetup[module]{
  option1 = value ,
  option2 = value
}
```

```
% second possibility:
\chemsetup{
  module/option1 = value ,
  module/option2 = value
}
```

The second way allows to set options belonging to different modules with one call of \chemsetup.

3.3. Support Package CHEMFORMULA

CHEMFORMULA provides means of typesetting chemical formulas and reactions. You will see its macros \ch and \chcpd every now and then in this manual. When using CHEMMACROS you can consider the CHEMFORMULA package [Nie15a] to be loaded as CHEMMACROS makes use of it in various places. CHEMMACROS and CHEMFORMULA are tightly intertwined. Nevertheless you should be able to use the mhchem [Hen15] package with CHEMMACROS. Please see section 6.2.1 starting on page 27 for details and *caveats*. *The recommendation is to use CHEMFORMULA.*

A historical note: CHEMFORMULA started as a part of CHEMMACROS in January 2012. Since July 2013 it is a completely independent package – from CHEMFORMULA's point of view. It is maintained independently and has a manual of its own.

3.4. Upgrading from version < 5.0

People upgrading from versions < 5.0 will find that almost everything they know from earlier versions is the same in versions ≥ 5.0. But there are important and *breaking* differences:

- CHEMMACROS has no package options any more, all options are set via \chemsetup, also see section 3.2 on the previous page.
- Not all of the features are available per default any more, for some the corresponding module has to be loaded explicitly, see section 4. If suddenly some commands or environments seem to be undefined this is the most likely reason.
- Some option modules have been renamed (*e. g.*, iupac is now nomenclature). If you experience strange errors when you upgrade your document this is the most likely source.
- The command family \NewChem..., \RenewChem... and \DeclareChem... has a new member \ProvideChem....
- In \iupac the macro \- no longer gives a dash with breaking point. Instead - can be used directly.[2]
- The macro \ox has another default behaviour (pos = {super}) and the starred version has another effect (pos = {top}) than the same macro in earlier versions. Now the default behaviour follows IUPAC recommendations. A second change is that the atom is now treated as a CHEMFORMULA formula.
- The syntax of \NewChemReaction and friends is now different from what it used to be. If you have defined your own reaction environments you need to adapt!

2. \- will be provided a bit longer for easing the upgrade step but will be dropped eventually.

- CHEMMACROS offers a macro \state which is similar to but different from the earlier macro \State. \State is deprecated. There are also differences in the syntax of \enthalpy *vs.* the earlier \Enthalpy, \entropy *vs.* \Entropy and \gibbs *vs.* \Gibbs. The uppercase versions are deprecated. The macro \NewChemState also has a different syntax now.
- At various places in the code improvements and fixes have been made, too many to list them here. You should keep an open eye and first of all read the manual closely.

3.5. Compatibility Mode

3.5.1. For Users

It is actually not true that CHEMMACROS' has no package options any more. It has one very important package option:

compatibility = ⟨*num*⟩|newest|latest Default: 5.3

Let's you specify the version number of CHEMMACROS you want to use. Any version earlier than 5.0 will load v4.7. *i. e.*, the last version before CHEMMACROS got its modular structure.[3] Not using the option will always load the newest version. Please note that you only can specify the *number* of the version. For a version "5.2c" you can only set compatibility mode "5.2" but not specify the subrelease.

Both values newest and latest will choose the latest version available.

In your document you can check for the compatibility mode. For the following functions it is important to understand the rules: *greater* means *newer*. The version number is *not* a usual decimal number! The syntax for ⟨*num*⟩ is ⟨*major*⟩.⟨*minor*⟩. This means that a version 5.11 is *newer* than a version 5.7! In the same way *less* means *older*. As last example: 5.10 is *newer* (greater) than 5.1.

\IfChemCompatibilityTF{⟨*comp*⟩}{⟨*num*⟩}{⟨*true*⟩}{⟨*false*⟩}
Checks the value given through the option compatibility against ⟨*num*⟩ using ⟨*comp*⟩ and either leaves ⟨*true*⟩ or ⟨*false*⟩ in the input stream. ⟨*comp*⟩ can be one of <, <=, =, >= or >.

\IfChemCompatibilityT{⟨*comp*⟩}{⟨*num*⟩}{⟨*true*⟩}
Checks the value given through the option compatibility against ⟨*num*⟩ using ⟨*comp*⟩ and leaves ⟨*true*⟩ in the input stream if the check is logically true. ⟨*comp*⟩ can be one of <, <=, =, >= or >.

\IfChemCompatibilityF{⟨*comp*⟩}{⟨*num*⟩}{⟨*false*⟩}
Checks the value given through the option compatibility against ⟨*num*⟩ using ⟨*comp*⟩ and leaves ⟨*false*⟩ in the input stream if the check is logically false. ⟨*comp*⟩ can be one of <, <=, =, >= or >.

A possible usage:

```
1 \IfChemCompatibilityT{>=}{5.0}{\usechemmodule{all}}
```

Loading CHEMMACROS with compatibility = {4.7} also allows to use the package options from that version:

3. Mostly: the loaded v4.7 has got a few fixes

```
\usepackage[compatibility=4.7,language=german]{chemmacros}
```

3.5.2. For Module Writers

For future versions the aim is not to make such breaking changes again. While we never know what the future actually will bring CHEMMACROS now has the tools for tying code to a version number:

* `\chemmacros_if_compatiblity:nnTF` {⟨*comp*⟩} {⟨*num*⟩} {⟨*true*⟩} {⟨*false*⟩}
expl3 version of `\IfChemCompatibilityTF`.

In modules one can try adding code for a certain version or range of versions:

`\ChemCompatibility`{⟨*num*⟩} ⟨*code*⟩ `\EndChemCompatibility`
Leaves ⟨*code*⟩ in the input stream if the compatibility version x given by `compatibility` matches ⟨*num*⟩ ($x = \langle num \rangle$).

`\ChemCompatibilityFrom`{⟨*num*⟩} ⟨*code*⟩ `\EndChemCompatibility`
Leaves ⟨*code*⟩ in the input stream if the compatibility version x given by `compatibility` matches ⟨*num*⟩ or newer. This means ⟨*num*⟩ is the *oldest* version allowed ($x \geq \langle num \rangle$).

`\ChemCompatibilityTo`{⟨*num*⟩} ⟨*code*⟩ `\EndChemCompatibility`
Leaves ⟨*code*⟩ in the input stream if the compatibility version x given by `compatibility` matches ⟨*num*⟩ or older. This means ⟨*num*⟩ is the *newest* version allowed ($x \leq \langle num \rangle$).

`\ChemCompatibilityBetween`{⟨*num1*⟩}{⟨*num2*⟩} ⟨*code*⟩ `\EndChemCompatibility`
Leaves ⟨*code*⟩ in the input stream if the compatibility version x given by `compatibility` lies between ⟨*num2*⟩ and ⟨*num2*⟩ ($\langle num1 \rangle \leq x \leq \langle num2 \rangle$).

`\EndChemCompatibility`
This macro *must* end each of the `\ChemCompatibility...` macros.

You may refer to the current version of CHEMMACROS with the following tokenlists:

`\c_chemmacros_date_tl`
The current release date: “2016/01/13”.

`\c_chemmacros_version_major_number_tl`
The current major version: “5”.

`\c_chemmacros_version_minor_number_tl`
The current minor version: “3”.

`\c_chemmacros_version_number_tl`
The current version number: “5.3”.

`\c_chemmacros_version_subrelease_tl`
The current sub-release: “”.

`\c_chemmacros_version_tl`
The current version: “5.3”.

`\c_chemmacros_info_tl`
The package information: “comprehensive support for typesetting chemistry documents”.

4. General Options

CHEMMACROS has some core options which don't belong to any of the modules described in parts II and III. Those options have no module denoted in the left margin next to their descriptions and are also set without specifying a module:

```
\chemsetup{
  option1 = value ,
  option2 = value
}
```

Two of those options are explained now:

`modules` = {⟨*comma separated list of module names*⟩} (initially empty)
With this option you can specify which modules you want to load. Alternatively you can use `\usechemmodule`{⟨*comma separated list of module names*⟩}.

`greek` = {⟨*mapping*⟩} (initially empty)
Explicitly specify which mapping should be used by the CHEMGREEK package [Nie15b]. For details about what this means please refer to section 6.4 starting on page 28.

Some internal modules may also define core options, *e. g.*, the `lang` module, see section 6.5 starting on page 28.

Part II.
The Preloaded Modules

5. User Modules

5.1. The `acid-base` Module

Easy representation of pH, pK_a…

`\pH`
pH

`\pOH`
pOH

`\Ka`
K_a, depends on language settings, see section 6.5 starting on page 28. The translations can be adapted.

`\Kb`
K_b

`\Kw`
K_w

`\pKa[`⟨*num*⟩`]`
`\pKa`: pK_a, `\pKa[1]`: pK_{a1}, depends on language settings, see section 6.5 starting on page 28. The translations can be adapted.

`\pKb[`⟨*num*⟩`]`
`\pKb`: pK_b, `\pKb[1]`: pK_{b1}

`\p{`⟨*anything*⟩`}`
e. g. `\p{\Kw}` pK_w

```
\Ka\ \Kb\ \pKa\ \pKa[1] \pKb\ \pKb[1]
```

K_a K_b pK_a pK_{a1} pK_b pK_{b1}

> The operator p […] shall be printed in Roman type.
>
> The IUPAC Green Book *[Coh+08, p. 103]*

There is one option which changes the style the p is typeset, other options allow to change the subscript of the constants:

`acid-base » p-style = italics|slanted|upright` Default: `upright`
Set the style of the p operator.

`acid-base » K-acid = {`⟨*text*⟩`}` Default: `\ChemTranslate{K-acid}`
The subscript to `\Ka` and `\pKa`.

`acid-base » K-base = {`⟨*text*⟩`}` Default: `\ChemTranslate{K-base}`
The subscript to `\Kb` and `\pKb`.

`acid-base » K-water = {`⟨*text*⟩`}` Default: `\ChemTranslate{K-water}`
The subscript to `\Kw`.

```
\pH, \pKa \par
\chemsetup[acid-base]{p-style=slanted} \pH, \pKa \par
\chemsetup[acid-base]{p-style=italics} \pH, \pKa
```

---

pH, pK_a
*p*H, *p*K_a
*p*H, *p*K_a

As you can see the default subscripts of `\Kw`, `\Ka` and `\Kb` are lowercase letters. The literature is inconclusive about if this is the right way or if uppercase letters should be preferred. In textbooks the uppercase variant usually seems to be used while journals seem to prefer the lowercase variant. CHEMMACROS' default follows the usage in *The IUPAC Green Book* [Coh+08]. If you want to change this you have two possibilities:

```
% this works only in the preamble:
% \DeclareTranslation{English}{K-acid}{\mathrm{A}}% use your language here
% alternative:
\chemsetup{acid-base/K-acid=\mathrm{A}}% overwrites language dependent settings
\pKa
```

---

pK_A

5.2. The charges Module

The `charges` module loads the module `chemformula`.

5.2.1. Charge Symbols

`\fplus`
⊕ formal positive charge

`\fminus`
⊖ formal negative charge

`\scrp`
+ scriptstyle positive charge (*e. g.*, for usage in chemfig's [Tel15] formulas).

`\scrm`
- scriptstyle negative charge (*e. g.*, for usage in chemfig's formulas).

`\fscrp`
⊕ scriptstyle formal positive charge (*e. g.*, for usage in chemfig's formulas).

`\fscrm`
⊖ scriptstyle formal negative charge (*e. g.*, for usage in chemfig's formulas).

`\fsscrp`
⊕ scriptscriptstyle formal positive charge (*e. g.*, for usage in chemfig's formulas).

`\fsscrm`
⊖ scriptscriptstyle formal negative charge (*e. g.*, for usage in chemfig's formulas).

5.2.2. Ion Charges

Simple displaying of (real) charges. It is worth noting that these commands really are relicts from a time when CHEMMACROS tried hard to be compliant with mhchem and CHEMFORMULA didn't exist, yet. They are still provided for backwards compatibility but *my recommendation is to use* `\ch` (see the documentation of the CHEMFORMULA package [Nie15a]) *and forget about these commands:*

`\pch[`⟨*number*⟩`]`
positive charge

`\mch[`⟨*number*⟩`]`
negative charge

`\fpch[`⟨*number*⟩`]`
formal positive charge

`\fmch[`⟨*number*⟩`]`
formal negative charge

```
A\pch\ B\mch[3] C\fpch[2] D\fmch
```

$A^{+}\ B^{3-}\ C^{2\oplus}\ D^{\ominus}$

5.2.3. Partial Charges and Similar Stuff

The next ones probably are seldomly needed but nevertheless useful:

`\delp`
$\delta+$ partial positive charge

`\delm`
$\delta-$ partial negative charge

`\fdelp`
$\delta\oplus$ partial formal positive charge

`\fdelm`
$\delta\ominus$ partial formal negative charge

These macros for example can be used with the `\ox` command (see section 7.7 starting on page 41) or with the chemfig package:

```
\chemsetup{
  charges/circled = all,
  redox/parse     = false,
  redox/pos       = top
}
\ch{"\ox{\delp,H}" -{} "\ox{\delm,Cl}"} \hspace*{1cm}
\chemfig{\chemabove[3pt]{\lewis{246,Br}}{\delm}-\chemabove[3pt]{H}{\delp}}
```

5.2.4. Charge Options

`charges » circled = formal|all|none` Default: `formal`
CHEMMACROS uses two different kinds of charges which indicate the usage of real (+/−) and formal (⊕/⊖) charges. The option `formal` distinguishes between them, option `none` displays them all without circle, option `all` circles all.

`charges » circletype = chem|math` Default: `chem`
This option switches between two kinds of circled charge symbols: `\fplus` ⊕/`\fminus` ⊖ (`chem`) and `$\oplus$` ⊕/`$\ominus$` ⊖ (`math`).

`charges » partial-format = {`⟨*LaTeX code*⟩`}` Default: `\tiny`
Code which formats the macros defined with `\NewChemPartialCharge` (see section 5.2.5).

5.2.5. Own Charge Macros

Just in case the existing macros don't fit you needs there are commands for defining new ones or modifying the existing ones. These commands define macros like those described in section 5.2.2 starting on page 10.

`\NewChemCharge{⟨cs⟩}{⟨charge symbol⟩}`
Defines a new macro ⟨cs⟩. Raises an error if ⟨cs⟩ already exists.

`\RenewChemCharge{⟨cs⟩}{⟨charge symbol⟩}`
Redefines a new macro ⟨cs⟩. Raises an error if ⟨cs⟩ doesn't exist.

`\DeclareChemCharge{⟨cs⟩}{⟨charge symbol⟩}`
Defines a macro ⟨cs⟩. Silently overwrites ⟨cs⟩ if it exists.

`\ProvideChemCharge{⟨cs⟩}{⟨charge symbol⟩}`
Defines a new macro ⟨cs⟩. Does nothing if ⟨cs⟩ already exists.

An example of usage is the definition of the existing ion charge macros:

```
1 \NewChemCharge\fpch{\fplus}
2 \NewChemCharge\fmch{\fminus}
```

These commands define macros like those described in section 5.2.3 on the preceding page.

`\NewChemPartialCharge{⟨cs⟩}{⟨charge symbol⟩}`
Defines a new macro ⟨cs⟩. Raises an error if ⟨cs⟩ already exists.

`\RenewChemPartialCharge{⟨cs⟩}{⟨charge symbol⟩}`
Redefines a new macro ⟨cs⟩. Raises an error if ⟨cs⟩ doesn't exist.

`\DeclareChemPartialCharge{⟨cs⟩}{⟨charge symbol⟩}`
Defines a macro ⟨cs⟩. Silently overwrites ⟨cs⟩ if it exists.

`\ProvideChemPartialCharge{⟨cs⟩}{⟨charge symbol⟩}`
Defines a new macro ⟨cs⟩. Does nothing if ⟨cs⟩ already exists.

An example of usage is the definition of the existing partial charge macros:

```
1 \NewChemPartialCharge\fdelp{\fplus}
2 \NewChemPartialCharge\fdelm{\fminus}
```

5.3. The `nomenclature` Module

The `nomenclature` module loads the `tikz` module. It also loads the package scrlfile which is part of the KOMA-Script bundle [Koh15].

5.3.1. The \iupac Command

Similar to the bpchem package [Ped04] CHEMMACROS provides a command[4] for typesetting IUPAC names. Why is that useful? IUPAC names can get very long. So long indeed that they span over more than two lines, especially in two-column documents. This means they must be allowed to be broken more than one time. This is what the following command does.

`\iupac{`⟨*IUPAC name*⟩`}`

Inside this command use `|` indicate a breaking point `^` as a shortcut for `\textsuperscript`. `-`, `(` and `)` allow words to be broken while still allow the rest of word to be hyphenated, likewise `[` and `]`.

```
\begin{minipage}{.4\linewidth}
  \iupac{%
    Tetra|cyclo[2.2.2.1^{1,4}]-un|decane-2-dodecyl-%
    5-(hepta|decyl|iso|dodecyl|thio|ester)%
  }
\end{minipage}
```

Tetracyclo[2.2.2.1^{1,4}]-undecane-2-dodecyl-5-(heptadecylisododecyl-thioester)

The `\iupac` command is more of a semantic command. In many cases you can achieve (nearly) the same thing by using `\-` instead of `|`, and `\textsuperscript` instead of `^` without `\iupac`. There are some important differences, though:

- The character `-` inserts a small space before the hyphen and removes a small space after it. Also usually words with hyphens are only allowed to break at the hyphen. Inside `\iupac` the hyphen will not prevent further hyphenation. The amount of inserted space can be customized.
- The character `|` not only prevents ligatures but also inserts a small space. The amount of inserted space can be customized.
- The characters `(` and `)` allow the word to be hyphenated and don't prevent further hyphenation, likewise `[` and `]`.

Introduced in version 5.3

- The character `'` is printed as `\chemprime`.

```
\huge\iupac{2,4-Di|chlor|pentan} \par
2,4-Dichlorpentan
```

2,4-Dichlorpentan
2,4-Dichlorpentan

4. The idea and initial implementation is shamelessly borrowed from bpchem by Bjørn PEDERSEN.

Table 1: Demonstration of iupac's modes.

	auto	restricted	strict
\L	Ł	Ł	L
\iupac{\L}	L	L	L
\D	D	—	D
\iupac{\D}	D	D	D

\chemprime

Introduced in version 5.3

Prints a prime character in superscript position. It is defined as \ensuremath{{}^{\prime}}.

The spaces inserted by - and | can be customized.

nomenclature » hyphen-pre-space = {⟨*dim*⟩} Default: .01em

Set the space that is inserted before the hyphen set with -.

nomenclature » hyphen-post-space = {⟨*dim*⟩} Default: -.03em

Set the space that is inserted after the hyphen set with -.

nomenclature » break-space = {⟨*dim*⟩} Default: .01em

Set the space inserted by |.

The command \iupac serves another purpose, too, however. Regardless of the setting of the iupac option (see below) all the commands presented in this section are always defined *inside* \iupac. Quite a number of the naming commands have very general names: \meta, \D, \E, \L, \R, \S, \trans and so forth.[5] This means they either are predefined already (\L Ł) or are easily defined by another package or class (the cool package defines both \D and \E, for example). In order to give you control which commands are defined in which way, there is the option iupac:

nomenclature » iupac = auto|restricted|strict Default: auto

Take care of how IUPAC naming commands are defined.

It has three modes:

- iupac = {auto}: if the commands are *not* defined by any package or class you're using they are available generally, otherwise only *inside* \iupac.
- iupac = {restricted}: all naming commands are *only* defined inside \iupac. If the commands are defined by another package they of course have that meaning outside. They're not defined outside otherwise.
- iupac = {strict}: CHEMMACROS overwrites any other definition and makes the commands available throughout the document. Of course the commands can be redefined (but only in the document body). They will still be available inside \iupac then.

Table 1 demonstrates the different modes.

5. Please read section 5.3.2 on the next page before you consider using the one-letter commands

Table 2: IUPAC shortcuts for Greek letters.

macro	\a	\b	\g	\d	\k	\m	\n	\w
letter	α	β	γ	δ	κ	μ	η	ω

5.3.2. Macros Defined (Not Only) For Usage in \iupac

One-letter Macros For some of the macros explained in this section one-letter commands are defined – with a *caveat* in mind, though: they are not actively recommended. One-letter commands seldomly have meaningful names and often they've also been defined by other packages. This means they make collaboration more difficult than it needs to be and are a source for package conflicts. CHEMMACROS solves the latter problem by only providing them inside the argument of \iupac. The one exception CHEMMACROS makes is the command \p (for things like pH) which is and will remain an official command (see section 5.1 starting on page 8). For all other one-letter macros alternatives with more meaningful names exist.

Greek Letters Greek letters in compound names are typeset upright. Here are a few examples for the existing macros:

\chemalpha α
Upright lowercase alpha

\chembeta β
Upright lowercase alpha

\chemgamma γ
Upright lowercase alpha

\chemdelta δ
Upright lowercase alpha

There exist two commands for each of the twenty-four Greek letters: a lowercase and an uppercase version (\chemalpha and \chemAlpha). Those commands are actually provided by the CHEMGREEK package. For more details read section 6.4 starting on page 28 and also refer to CHEMGREEK's documentation.

There are a number of one-letter commands that some people may find convenient to use which use above mentioned commands to print Greek letters inside \iupac. They're listed in table 2.

```
1 \iupac{5\chemalpha-androstan-3\chembeta-ol} \par
2 \iupac{\chemalpha-(tri|chloro|methyl)-\chemomega
3   -chloro|poly(1,4-phenylene|methylene)}
```

5α-androstan-3β-ol
α-(trichloromethyl)-ω-chloropoly(1,4-phenylenemethylene)

Hetero Atoms and added Hydrogen Attachments to hetero atoms and added hydrogen atoms are indicated by italic letters [Coh+08]. CHEMMACROS defines a few macros for the most common ones.

`\hydrogen` *H*
The italic H for hydrogen. (An alias for this command is `\H`.)

`\oxygen` *O*
The italic O for oxygen. (An alias for this command is `\O`.)

`\nitrogen` *N*
The italic N for nitrogen. (An alias for this command is `\N`.)

`\sulfur` *S*
The italic S for sulfur. (An alias for this command is `\Sf`.)

`\phosphorus` *P*
The italic P for phosphorus. (An alias for this command is `\P`.)

```
\iupac{\nitrogen-methyl|benz|amide}

\iupac{3\hydrogen-pyrrole}

\iupac{\oxygen-ethyl hexanethioate}
```

N-methylbenzamide
3*H*-pyrrole
O-ethyl hexanethioate

Cahn-Ingold-Prelog

`\cip{`⟨*conf*⟩`}`
Typeset Cahn-Ingol-Prelog descriptors, *e. g.*: `\cip{R,S}` (*R*,*S*). ⟨*conf*⟩ may be a csv list of entries.

`\rectus` (*R*)
Typeset rectus descriptor. (An alias for this command is `\R`.)

`\sinister` (*S*)
Typeset sinister descriptor. (An alias for this command is `\S`.)

Both these commands and the entgegen/zusammen descriptors get a small additional amount of kerning after the closing parenthesis. This amount can be changed through the following option:

`nomenclature » cip-kern = {`⟨*dim*⟩`}` Default: `.075em`
Set the amount of kerning after the closing parenthesis.

Fischer

`\dexter` D
Typeset dexter descriptor. (An alias for this command is `\D`.)

`\laevus` L
Typeset laevus descriptor. (An alias for this command is `\L`.)

cis/trans, zusammen/entgegen, syn/anti & tert

- `\cis` *cis* `\trans` *trans*
- `\fac` *fac* `\mer` *mer*
- `\sin` *sin* `\ter` *ter*
- `\zusammen` (*Z*) `\entgegen` (*E*)
- `\syn` *syn* `\anti` *anti*
- `\tert` *tert*

An alias for `\entgegen` is `\E` and an alias for `\zusammen` is `\Z`.

ortho/meta/para

`\ortho` *o* `\meta` *m* `\para` *p*

Although these commands are provided I like to cite *The IUPAC Blue Book* [PPR04]:

> The letters *o*, *m*, and *p* have been used in place of *ortho*, *meta*, and *para*, respectively, to designate the 1,2-, 1,3-, and 1,4- isomers of disubstituted benzene. This usage is strongly discouraged and is not used in preferred IUPAC names. *[PPR04, p. 90]*

Absolute Configuration

`\Rconf[`⟨*letter*⟩`]`

`\Rconf`: (R) `\Rconf[]`: 

`\Sconf[`⟨*letter*⟩`]`

`\Sconf`: (S) `\Sconf[]`: 

Coordination Chemistry CHEMMACROS provides a few commands useful in coordination chemistry:

`\bridge{`⟨*num*⟩`}` μ_3-
Denote bridging ligand connection.

`\hapto{`⟨*num*⟩`}` η^5-
Denote hapticity.

`\dento{`⟨*num*⟩`}` κ^2-
Denote denticity.

```
Ferrocene = \iupac{bis(\hapto{5}cyclo|penta|dienyl)iron} \par
\iupac{tetra-\bridge{3}iodido-tetrakis[tri|methyl|platinum(IV)]}
```

Ferrocene = bis(η^5-cyclopentadienyl)iron
tetra-μ_3-iodido-tetrakis[trimethylplatinum(IV)]

Two options allow customization:

`nomenclature » bridge-number = sub|super` Default: `sub`

Appends the number as a subscript or superscript, depending on the choice. The IUPAC recommendation is the subscript [Con+05].

`nomenclature » coord-use-hyphen = true|false` Default: `true`

Append a hyphen to `\hapto`, `\dent` and `\bridge` or don't.

Examples

```
\iupac{\dexter-Wein|s\"aure} =
\iupac{\cip{2S,3S}-Wein|s\"aure} \par
\iupac{\dexter-($-$)-Threose} =
\iupac{\cip{2S,3R}-($-$)-2,3,4-Tri|hydroxy|butanal} \par
\iupac{\cis-2-Butene} =
\iupac{\zusammen-2-Butene}, \par
\iupac{\cip{2E,4Z}-Hexa|diene} \par
\iupac{\meta-Xylol} =
\iupac{1,3-Di|methyl|benzene}
```

D-Weinsäure = (2*S*,3*S*)-Weinsäure
D-(−)-Threose = (2*S*,3*R*)-(−)-2,3,4-Trihydroxybutanal
cis-2-Butene = (*Z*)-2-Butene,
(2*E*,4*Z*)-Hexadiene
m-Xylol = 1,3-Dimethylbenzene

5.3.3. Own `\iupac` Macros And Shorthands

If you find any commands missing you can define them using

`\NewChemIUPAC{⟨cs⟩}{⟨declaration⟩}`
Define a new IUPAC command that is in any case defined inside of `\iupac` regardless if ⟨*cs*⟩ is defined elsewhere already.

`\ProvideChemIUPAC{⟨cs⟩}{⟨declaration⟩}`
Define a new IUPAC command that is in any case defined inside of `\iupac` regardless if ⟨*cs*⟩ is defined elsewhere already only if the corresponding IUPAC macro is not defined, yet.

`\RenewChemIUPAC{⟨cs⟩}{⟨declaration⟩}`
Redefine an existing IUPAC command that is in any case defined inside of `\iupac` regardless if ⟨*cs*⟩ is defined elsewhere already.

`\DeclareChemIUPAC{⟨cs⟩}{⟨declaration⟩}`
Define a new IUPAC command that is in any case defined inside of `\iupac` regardless if ⟨*cs*⟩ is defined elsewhere already. This silently overwrites an existing IUPAC macro definition.

`\LetChemIUPAC{⟨cs1⟩}{⟨cs2⟩}`
Defines ⟨*cs1*⟩ to be an alias of ⟨*cs2*⟩.

A command defined in this way will obey the setting of the option iupac. This means any existing command is only overwritten with iupac = {strict}. However, \NewChemIUPAC will *not* change the definition of an existing IUPAC naming command but issue an error if the IUPAC naming command already exists. \DeclareChemIUPAC *will* overwrite an existing IUPAC command.

```
\NewChemIUPAC\endo{\textsc{endo}}
\RenewChemIUPAC\anti{\textsc{anti}}
\iupac{(2-\endo,7-\anti)-2-bromo-7-fluoro|bicyclo[2.2.1]heptane}
```

(2-ENDO,7-ANTI)-2-bromo-7-fluorobicyclo[2.2.1]heptane

\RenewChemIUPAC allows you to redefine the existing IUPAC naming commands.

```
\iupac{\meta-Xylol} \par
\RenewChemIUPAC\meta{\textup{m}}
\iupac{\meta-Xylol}
```

m-Xylol
m-Xylol

There's also a way for defining new IUPAC shorthands or changing the existing ones:

\NewChemIUPACShorthand⟨*shorthand token*⟩⟨*control sequence*⟩
Defines a new IUPAC shorthand. Inside \iupac it will be equal to using ⟨*control sequence*⟩. This throws an error if ⟨*shorthand token*⟩ is already defined.

\RenewChemIUPACShorthand⟨*shorthand token*⟩⟨*control sequence*⟩
Redefines an existing IUPAC shorthand. This throws an error if ⟨*shorthand token*⟩ is not defined, yet.

\DeclareChemIUPACShorthand⟨*shorthand token*⟩⟨*control sequence*⟩
Defines a new IUPAC shorthand or redefines an existing one.

\ProvideChemIUPACShorthand⟨*shorthand token*⟩⟨*control sequence*⟩
Provides a new IUPAC shorthand. Does nothing if ⟨*shorthand token*⟩ is already defined.

\RemoveChemIUPACShorthand⟨*shorthand token*⟩
Deletes an existing IUPAC shorthand.

5.3.4. Latin Phrases

The package chemstyle [Wri13] provides the command \latin to typeset common latin phrases in a consistent way. CHEMMACROS defines a similar \latin only if chemstyle has *not* been loaded and additionally provides these commands:

\insitu *in situ* \abinitio *ab initio* \invacuo *in vacuo*

If the package chemstyle has been loaded they are defined using chemstyle's \latin *command. This means that then the appearance depends on chemstyle's option* abbremph.
The commands are defined through

`\NewChemLatin{⟨cs⟩}{⟨phrase⟩}`
Define a new latin phrase. Gives an error if ⟨*cs*⟩ already exists.

`\DeclareChemLatin{⟨cs⟩}{⟨phrase⟩}`
Define a new latin phrase. Silently redefined existing macros.

`\RenewChemLatin{⟨cs⟩}{⟨phrase⟩}`
Redefine an existing latin phrase. Gives an error if ⟨*cs*⟩ doesn't exist.

`\ProvideChemLatin{⟨cs⟩}{⟨phrase⟩}`
Define a new latin phrase only if ⟨*cs*⟩ doesn't exist.

```
\NewChemLatin\ltn{latin text}\ltn          latin text
```

If you have *not* loaded chemstyle you can change the appearance with this option:

`nomenclature » format = {⟨definition⟩}` Default: `\itshape`
Set the format of the latin phrases.

5.4. The `particles` Module

The `particles` module loads the modules `charges` and `chemformula`.

5.4.1. Provided Particle Macros

The `particles` defines a number of macros which can be used for typesetting common particles in the running text. Most of them don't make much sense in chemformula [Nie15a]'s `\ch`, though, which doesn't mean that they can't be used there, of course:

`\el` e^- `\prt` p^+ `\ntr` n^0 `\Hyd` OH^- `\Oxo` H_3O^+ `\water` H_2O `\El` E^+ `\Nuc` Nu^- `\ba` ba^-

All of these macros are defined using chemformula's `\chcpd`. The details are explained in section 5.4.2 on the following page.

The macros `\Nuc` and `\ba` are special: they have an optional argument for the following options:

`particles » elpair = dots|dash|false` Default: `false`
Determine how the electron pair of the nucleophiles is displayed. The electron pair is drawn using CHEMFORMULA's `\chlewis` macro.

`particles » space = {⟨dim⟩}` Default: `.1em`
Introduced in version 5.3
Sets the space that is inserted between the electron pair and the negative charge sign.

Both options can of course also be set with `\chemsetup`.

```
\ba[elpair=dots] \Nuc[elpair=dash]
                                           ba:⁻ Nu|⁻
\chemsetup[particles]{elpair=false}        ba⁻ Nu⁻
\ba\ \Nuc
```

5.4.2. Defining Own Particle Macros

There are two sets of macros, one for defining particles and one for defining nucleophiles.

`\NewChemParticle{⟨cs⟩}{⟨formula⟩}`
Defines a new macro ⟨*cs*⟩. ⟨*formula*⟩ is any valid CHEMFORMULA compound. Raises an error if ⟨*cs*⟩ already exists.

`\RenewChemParticle{⟨cs⟩}{⟨formula⟩}`
Redefines a new macro ⟨*cs*⟩. ⟨*formula*⟩ is any valid CHEMFORMULA compound. Raises an error if ⟨*cs*⟩ doesn't exist.

`\DeclareChemParticle{⟨cs⟩}{⟨formula⟩}`
Defines a macro ⟨*cs*⟩. ⟨*formula*⟩ is any valid CHEMFORMULA compound. Silently overwrites ⟨*cs*⟩ if it exists.

`\ProvideChemParticle{⟨cs⟩}{⟨formula⟩}`
Defines a new macro ⟨*cs*⟩. ⟨*formula*⟩ is any valid CHEMFORMULA compound. Does nothing if ⟨*cs*⟩ already exists.

An example of usage is the definition of the existing particle macros:

```
\NewChemParticle\el {e-}
\NewChemParticle\prt{p+}
\NewChemParticle\ntr{n^0}
```

The following set defines macros like `\Nuc`

`\NewChemNucleophile{⟨cs⟩}{⟨formula⟩}`
Defines a new macro ⟨*cs*⟩. ⟨*formula*⟩ is any valid CHEMFORMULA compound. Note that ⟨*formula*⟩ will get a trailing negative charge! Raises an error if ⟨*cs*⟩ already exists.

`\RenewChemNucleophile{⟨cs⟩}{⟨formula⟩}`
Redefines a new macro ⟨*cs*⟩. ⟨*formula*⟩ is any valid CHEMFORMULA compound. Note that ⟨*formula*⟩ will get a trailing negative charge! Raises an error if ⟨*cs*⟩ doesn't exist.

`\DeclareChemNucleophile{⟨cs⟩}{⟨formula⟩}`
Defines a macro ⟨*cs*⟩. ⟨*formula*⟩ is any valid CHEMFORMULA compound. Note that ⟨*formula*⟩ will get a trailing negative charge! Silently overwrites ⟨*cs*⟩ if it exists.

`\ProvideChemNucleophile{⟨cs⟩}{⟨formula⟩}`
Defines a new macro ⟨*cs*⟩. ⟨*formula*⟩ is any valid CHEMFORMULA compound. Note that ⟨*formula*⟩ will get a trailing negative charge! Does nothing if ⟨*cs*⟩ already exists.

An example of usage is the definition of the existing nucleophile macros:

```
\NewChemNucleophile\Nuc{Nu}
\NewChemNucleophile\ba {ba}
```

A macro defined this way will have an optional argument for the `elpair` option.

5.5. The phases Module

The phases module loads the chemformula modul.

5.5.1. Basics

These commands are intended to indicate the phase of a compound.

\sld (s) \lqd (l) \gas (g) \aq (aq)

```
\ch{C\sld{} + 2 H2O\lqd{} -> CO2\gas{} + 2 H2\gas}\par
To make it complete: NaCl\aq.
```

$\mathrm{C(s) + 2\,H_2O(l) \longrightarrow CO_2(g) + 2\,H_2(g)}$
To make it complete: $\mathrm{NaCl(aq)}$.

The IUPAC recommendation to indicate the state of aggregation is to put it in parentheses after the compound [Coh+08]. However, you might want to put it as a subscript which is also very common.

> The [...] symbols are used to represent the states of aggregation of chemical species. The letters are appended to the formula in parentheses and should be printed in Roman (upright) type without a full stop (period). The IUPAC Green Book *[Coh+08, p. 54]*

There are two options to customize the output:

phases » pos = side|sub Default: side
Switch the position of the phase indicator.

phases » space = {⟨*dim*⟩} Default: .1333em
Change the default spacing between compound a phase indicator if pos = {side}. A TEX dimension.

```
\chemsetup[phases]{pos=sub}
\ch{C\sld{} + 2 H2O\lqd{} -> CO2\gas{} + 2 H2\gas}\par
To make it complete: NaCl\aq.
```

$\mathrm{C_{(s)} + 2\,H_2O_{(l)} \longrightarrow CO_{2(g)} + 2\,H_{2(g)}}$
To make it complete: $\mathrm{NaCl_{(aq)}}$.

All those phase commands have an optional argument:

```
\ch{H2O "\lqd[\SI{5}{\celsius}]"}
```
$H_2O\,(\mathrm{l},\,5\,^{\circ}C)$

There is also a generic phase command:

`\phase{`⟨*phase*⟩`}`
If you need a phase indicator just once or twice. You can use it to denote a phase for which there is no phase command, yet.

5.5.2. Define Own Phases

Depending on the subject of your document you might need to indicate other states of aggregation. You can easily define them.

`\NewChemPhase{`⟨*cs*⟩`}{`⟨*symbol*⟩`}`
Define a new phase command. See section 5.5.3 for a way to define language dependent settings. Gives an error if ⟨*cs*⟩ already exists.

`\DeclareChemPhase{`⟨*cs*⟩`}{`⟨*symbol*⟩`}`
Define a new phase command. See section 5.5.3 for a way to define language dependent settings. Overwrites previous definitions of ⟨*cs*⟩.

`\RenewChemPhase{`⟨*cs*⟩`}{`⟨*symbol*⟩`}`
Redefine an existing phase command. See section 5.5.3 for a way to define language dependent settings. Gives an error if ⟨*cs*⟩ is not defined.

`\ProvideChemPhase{`⟨*cs*⟩`}{`⟨*symbol*⟩`}`
Define a new phase command. See section 5.5.3 for a way to define language dependent settings. Does nothing if ⟨*cs*⟩ is already defined.

```
% preamble:
\NewChemPhase\aqi{aq,$\infty$} % aqueous solution at infinite dilution
\NewChemPhase\cd {cd} % condensed phase
\NewChemPhase\lc {lc} % liquid crystal
\ch{NaOH\aqi} \ch{H2O\cd} \ch{U\phase{cr}} \ch{A\lc}\par
\chemsetup[phases]{pos=sub}
\ch{NaOH\aqi} \ch{H2O\cd} \ch{U\phase{cr}} \ch{A\lc}
```

$\mathrm{NaOH\,(aq,\infty)\ H_2O\,(cd)\ U\,(cr)\ A\,(lc)}$
$\mathrm{NaOH_{(aq,\infty)}\ H_2O_{(cd)}\ U_{(cr)}\ A_{(lc)}}$

5.5.3. Language Dependencies

For each phase command a translation into the custom language can be defined. If a phase is declared with `\NewChemPhase` no translation exists and for every babel language the literal string is used that was provided as a definition. Let's say you define the phase

```
\NewChemPhase\liquid{l}
```

and want to add the German translation “fl”. Then you could do

```
\DeclareTranslation{German}{phase-liquid}{f\/l}
```

This way, when you use it in a German document using the appropriate babel option using `\liquid` would correctly translate. For this the package translations [Nie13] is used. The ID always is `phase-`⟨*csname*⟩ where ⟨*csname*⟩ is the name of the phase command you defined without leading backslash.

See section 6.5 starting on page 28 for predefined translations and general language options of CHEMMACROS.

5.6. The `symbols` Module

The `symbols` module defines a few symbols chemists need now and then. It loads the package amstext [MS00].

`\transitionstatesymbol`
This is self-explaining: ‡

`\standardstate`
Again self-explaining: ⦵

`\changestate`
The uppercase delta used in ΔH for example.

6. Internal Modules

6.1. The `base` Module

The `base` module is the core module of CHEMMACROS. It defines some tools which can (and should) be used in other modules. This means this section is only interesting for you if you plan to write a module yourself (see section A starting on page 61 for details).

This module requires the packages bm [CM04], amstext [MS00], and etoolbox [Leh15].

This module also provides `\chemsetup` and the option `modules`.

It also provides a number of (expl3) macros which may be used in other modules. In the macro descriptions below *TF* denotes that a `T`, an `F` and a `TF` variant exist. In case of an expandable conditional (*) also the predicate variant is available.

* `\chemmacros_is_int:n`*TF* {⟨*number*⟩} {⟨*true*⟩} {⟨*false*⟩}
Checks if ⟨*number*⟩ is an integer or not.

* `\chemmacros_if_loaded:nn`*TF* {⟨*package*⟩|⟨*class*⟩} {⟨*name*⟩} {⟨*true*⟩} {⟨*false*⟩}
Checks if package (or class) ⟨*name*⟩ has been loaded. Also works after begin document.

* \chemmacros_if_package_loaded:nTF {⟨name⟩} {⟨true⟩} {⟨false⟩}
  Checks if package ⟨name⟩ has been loaded. Also works after begin document.

* \chemmacros_if_class_loaded:nTF {⟨name⟩} {⟨true⟩} {⟨false⟩}
  Checks if class ⟨name⟩ has been loaded. Also works after begin document.

\chemmacros_leave_vmode:
  Equivalent of \leavevmode.

\chemmacros_nobreak:
  Inserts a penalty of 10 000.

\chemmacros_allow_break:
  Inserts a penalty of 0.

\chemmacros_skip_nobreak:N ⟨skip/length variable⟩
  Insert a horizontal skip while linebreak is disallowed.

\chemmacros_if_is_int:nTF {⟨input⟩} {⟨true⟩} {⟨false⟩}
  Checks if ⟨input⟩ is an integer or something else.

\chemmacros_if_bold:TF {⟨true⟩} {⟨false⟩}
  Checks if the current font weight is one of b, bc, bm, bx, bux, eb, ebc, ebx, mb, sb, sbc, sbx, ub, ubc or ubx.

\chemmacros_bold:n {⟨text⟩}
  Checks if the current font weight is bold and if yes places ⟨text⟩ in \textbf if in text mode or in \bm if in math mode. If false ⟨text⟩ simply is placed in the input stream as is.

\chemmacros_text:n {⟨text⟩}
  Ensures that ⟨text⟩ is placed in text mode.

\chemmacros_math:n {⟨text⟩}
  Ensures that ⟨text⟩ is placed in math mode.

\chemmacros_new_macroset:nnn {⟨type⟩} {⟨arg spec⟩} {⟨internal command call⟩}
  A command to define a set of macros \NewChem⟨type⟩, \RenewChem⟨type⟩, \DeclareChem⟨type⟩ and \ProvideChem⟨type⟩. ⟨arg spec⟩ is any valid argument specification for xparse's \DeclareDocumentCommand [L3Pb]. ⟨internal command call⟩ should be a macro which makes definitions *without* error checks, *i. e.*, define new macros or redefine existing ones like \def does. This macro just should get the arguments passed on to. Have a look at the example below.

  This is how the macros \NewChemParticle, \RenewChemParticle, \DeclareChemParticle and \ProvideChemParticle were defined:

```
1 \chemmacros_new_macroset:nnn {Particle} {mm}
2   { \chemmacros_define_particle:Nn #1 {#2} }
```

The following macros strictly speaking are not provided by the base module but this place fits best for their description.

* `\chemmacros_if_module_exist:nTF` {⟨*module*⟩} {⟨*true*⟩} {⟨*false*⟩}
Checks if a file with the correct name for a module ⟨*module*⟩ can be found.

* `\chemmacros_if_module_loaded:nTF` {⟨*module*⟩} {⟨*true*⟩} {⟨*false*⟩}
Checks if the module ⟨*module*⟩ has already been loaded or not.

`\chemmacros_load_module:n` {⟨*module*⟩}
Loads module ⟨*module*⟩ if it hasn't been loaded, yet.

`\chemmacros_load_modules:n` {⟨*csv list of modules*⟩}
Loads every module in ⟨*csv list of modules*⟩ if they haven't been loaded, yet. This is the code level variant of `\usechemmodule`.

`\chemmacros_before_module:nn` {⟨*module*⟩} {⟨*code*⟩}
Introduced in version 5.1
Saves ⟨*code*⟩ and inserts it right before ⟨*module*⟩ is loaded. If ⟨*module*⟩ is never loaded then ⟨*code*⟩ is never inserted. If ⟨*module*⟩ already is loaded when the command is used then ⟨*code*⟩ also is never inserted.

`\chemmacros_after_module:nn` {⟨*module*⟩} {⟨*code*⟩}
Introduced in version 5.1
Saves ⟨*code*⟩ and inserts it right after ⟨*module*⟩ is loaded. If ⟨*module*⟩ is never loaded then ⟨*code*⟩ is never inserted. If ⟨*module*⟩ already is loaded when the command is used then ⟨*code*⟩ is inserted immediately.

6.2. The `chemformula` Module

The `chemformula` module loads the chemformula package [Nie15a] or the mhchem package [Hen15] and the amstext package [MSoo]. It also loads the `charges` module.

6.2.1. For Users

This module provides a general option:

`formula` = `chemformula|mhchem` Default: `chemformula`
Introduced in version 5.1
This option let's you choose if chemical formula's are typeset using chemformula's `\ch` and `\chcpd` or mhchem's `\ce`. The corresponding package is loaded.

If you explicitly set this option the corresponding package is loaded immediately. Otherwise the option is set at the end of the preamble.

Introduced in version 5.2
If you load either mhchem or chemformula and haven't set `formula` CHEMMACROS will choose the corresponding package. If you have loaded both mhchem and chemformula CHEMMACROS will raise a warning and choose chemformula. *All automatic choices only happen at the end of the preamble.*

Using the chemformula Package If you set `formula = {chemformula}` the `chemformula` module makes it possible that you can set all CHEMFORMULA options via the `\chemsetup` command using the module `chemformula`, for example:

```
1 \chemsetup[chemformula]{format=\sffamily}
```

Everywhere where CHEMMACROS typesets chemical formulas CHEMFORMULA's macros `\chcpd` or `\ch` are used, for example in the reaction environments provided by the `reactions` module.

Using the mhchem Package If you set `formula = {mhchem}` the `chemformula` module makes it possible that you can set all of mhchem's options via the `\chemsetup` command using the module `mhchem`, for example:

```
\chemsetup[mhchem]{format=\sffamily}
```

Everywhere where CHEMMACROS typesets chemical formulas mhchem's macro `\ce` is used, for example in the reaction environments provided by the `reactions` module.

There are a few *caveat*s if you use this method:

- This method has not been extensively tested, yet. There may be errors and wrong output at unexpected places.
- Using this method effectively disables the different values of the `particles` option `elpair` (see section 5.4).
- The different kinds of formal charges provided by the `charges` module (see section 5.2.2) are disabled. Formal charges always use the math method now.
- There may also be other incompatibilities (*e. g.*, mhchem has it's own method of setting upright Greek letters so it may or may not disable CHEMMACROS' mechanism).

6.2.2. For Module Writers

There's are two macros for module writers:

`\chemmacros_chemformula:n` {⟨*formula*⟩}
This is only a wrapper for `\chcpd` or `\ce`. It is recommended that module writers use this macro (or a variant thereof) inside of CHEMMACROS' macros whenever they want to display a chemical formula. Writers who prefer traditional LaTeX 2ε programming over expl3 should use `\chemmacros@formula`.

`\chemmacros_reaction:n` {⟨*reaction*⟩}
This is only a wrapper for `\ch` or `\ce`. It is recommended that module writers use this macro (or a variant thereof) inside of CHEMMACROS' macros whenever they want to display a chemical reaction. Writers who prefer traditional LaTeX 2ε programming over expl3 should use `\chemmacros@reaction`.

6.3. The `errorcheck` Module

Introduced in version 5.2

The `errorcheck` module provides some rudimentary support for giving users more meaningful messages when they use a command or environment provided by a module that they haven't loaded.

6.4. The greek Module

The greek module loads the chemgreek package [Nie15b].

This module provides one option:

greek = {⟨*mapping*⟩}

A valid value is any valid CHEMGREEK ⟨*mapping*⟩. CHEMMACROS will warn you if no mapping has been chosen or if you are using the default or the var-default mapping because this means that no upright Greek letters are available.

If you load a CHEMGREEK support package which allows an unambiguous choice of a mapping CHEMGREEK will make this choice automatically. This means if you say

```
\usepackage{upgreek}
\usepackage{chemmacros}
```

then CHEMMACROS will use upgreek's upright Greek letters. If you have

```
\usepackage{upgreek}
\usepackage{chemmacros}
\usepackage{textgreek}
```

then no unambiguous choice is possible and you should choose a mapping yourself, for example:

```
\usepackage{upgreek}
\usepackage{chemmacros}
\usepackage{textgreek}
\chemsetup{greek=textgreek}
```

For further details on mappings please refer to CHEMGREEK's manual.

6.5. The lang Module

The lang module provides language support for CHEMMACROS. It loads the package translations [Nie13].

6.5.1. Information For Users

This module defines the following option:

language = auto|⟨*language*⟩ Default: auto

If set to auto CHEMMACROS will detect the language used by babel [Bra13] or ployglossia [Cha13] automatically, fallback is English. Any language known to the translations package is a valid value for ⟨*language*⟩.

Table 3: Translation keys predefined by chemmacros.

key	fallback translation	German
scheme-name	Scheme	Schema
scheme-list	List of Schemes	Verzeichnis der Schemata
K-acid	\mathrm {a}	\mathrm {s}
K-base	\mathrm {b}	
K-water	\mathrm {w}	
phase-sld	s	f
phase-lqd	l	f\/l
phase-gas	g	g
phase-aq	aq	aq
list-of-reactions	List of Reactions	Reaktionsverzeichnis
reaction	Reaction	Reaktion

The language chosen via `language` is used for translation of certain strings in different places all over chemmacros. They are mentioned in the places when the corresponding function of chemmacros is explained.

Translation is done with the help of the translations package, available translation keys are listed in section 6.5.2.

6.5.2. Available Translation Keys

Table 3 lists (almost) all keys which are predefined in chemmacros. A translation key is a key which is understood by the translations package and its commands like `\GetTranslation`. For each key at least the English fallback translation is provided, for most also the German translation is provided. For a few keys also other translations are provided. If you find that your translation is missing you can provide it in the preamble:

`\DeclareTranslation`{⟨*language*⟩}{⟨*key*⟩}{⟨*translation*⟩}
Defines a translation of key ⟨*key*⟩ for the language ⟨*language*⟩. No error will be raised if a translation of ⟨*key*⟩ already exists. This command can only be used in the preamble and is defined by the translations package.

If you send me an email (see section B starting on page 62) with the translations for your language I'll gladly add them to the next release of chemmacros!

6.5.3. Information For Module Writers

* `\chemmacros_translate:n` {⟨*translation key*⟩}
Translates the given key to the language which is detected automatically or given by the user. Should be used in chemmacros' macros instead of translations' `\GetTranslation`.

`\l_chemmacros_language_tl`
A token list variable that holds the language which is used by `\chemmacros_translate:n` for translation, *after begin document*.

`\ChemTranslate`{⟨*translation key*⟩}
A version of `\chemmacros_translate:n` for those who prefer traditional LaTeX 2ε programming over expl3.

Part III.
Additional Modules

7. User Modules

7.1. The `all` *pseudo*-module

The `all` module is a pseudo module: it doesn't define any functionality at all. It does however load all other modules. So you can say

```
\chemsetup{ modules = all }
```

to ensure that every module is available. This *will not* load personal modules!

7.2. The `isotopes` Module

The `isotope` module loads the elements package [Nie15d]. This module defines one user command:

`\isotope*{`⟨*input*⟩`}`

⟨*input*⟩ can either be the *symbol* of an element or the *name* of an element. Be aware that *the name is language dependent*, refer to the manual of the elements package for details. To be on the safe side use the element symbol.

⟨*input*⟩ can also be comma separated list: `\isotope{`⟨*nuc*⟩`,`⟨*symbol*⟩`}`. If you leave ⟨*nuc*⟩ out then `\isotope` will display the most common isotope. Otherwise ⟨*nuc*⟩ will be used. If ⟨*nuc*⟩ is an isotope unknown to the elements package `\isotope` will write a warning to the log file.

The starred variant omits the element number.

```
\isotope{C}
\isotope*{C}
\isotope{14,C}
\isotope*{14,C}
```

$^{12}_{6}\mathrm{C}$ $^{12}\mathrm{C}$ $^{14}_{6}\mathrm{C}$ $^{14}\mathrm{C}$

As input for the element symbol you can choose any of the elements known to the elements package.

There are options which allow you to determine how the isotope is printed:

`isotopes » format = super|side` Default: `super`

Either print the isotope number as superscript or to the right of the element symbol.

`isotopes » side-connect = {`⟨*input*⟩`}` Default: -

Determine what is printed between the element symbol and the isotope number if `format = {side}`.

```
\isotope{C}
\chemsetup[isotopes]{format=side}
\isotope{C}
\chemsetup[isotopes]{side-connect=}
\isotope{C}
```

$^{12}_{6}\mathrm{C}$ C-12 C12

7.3. The mechanisms Module

The module mechanisms loads the package amstext [MSoo]. It provides one macro:

\mech[⟨*type*⟩]
Allows to specify the most common reaction mechanisms.

⟨*type*⟩ can have one of the following values:

\mech
(empty, no opt. argument) nucleophilic substitution S_N

\mech[1]
unimolecular nucleophilic substitution S_N1

\mech[2]
bimolecular nucleophilic substitution S_N2

\mech[se]
electrophilic substitution S_E

\mech[1e]
unimolecular electrophilic substitution S_E1

\mech[2e]
bimolecular electrophilic substitution S_E2

\mech[ar]
electrophilic aromatic substitution Ar-S_E

\mech[e]
elimination E

\mech[e1]
unimolecular elimination E1

\mech[e2]
bimolecular elimination E2

\mech[cb]
unimolecular elimination “conjugated base”, *i. e.*, via carbanion $E1_{cb}$

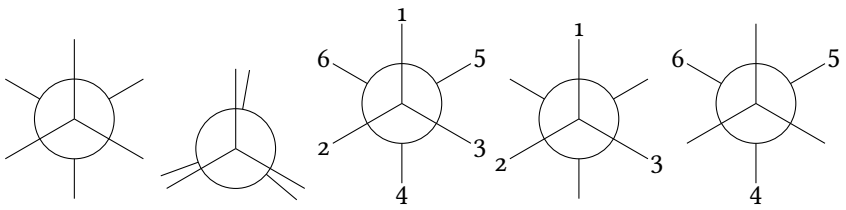
7.4. The newman Module

The newman module provides a command for drawing Newman projections. It loads the tikz module.

`\newman[`⟨*options*⟩`](`⟨*angle*⟩`){`⟨*1*⟩`,`⟨*2*⟩`,`⟨*3*⟩`,`⟨*4*⟩`,`⟨*5*⟩`,`⟨*6*⟩`}`

Create Newman projections. This command uses Ti*k*Z internally. ⟨*angle*⟩ rotates the back atoms counter clockwise with respect to the front atoms and is an optional argument. ⟨*1*⟩ to ⟨*6*⟩ are the positions, the first three are the front atoms, the last three the back atoms.

```
\newman{} \newman(170){}
\newman{1,2,3,4,5,6} \newman{1,2,3} \newman{,,,4,5,6}
```



Several options allow customization:

`newman » angle = {`⟨*angle*⟩`}` Default: `0`

Default angle.

`newman » scale = {`⟨*factor*⟩`}` Default: `1`

Scale the whole projection by factor ⟨*factor*⟩.

`newman » ring = {`⟨*tikz*⟩`}` (initially empty)

Customize the ring with Ti*k*Z keys.

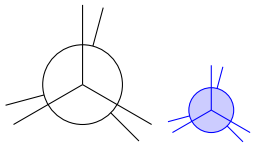
`newman » atoms = {`⟨*tikz*⟩`}` (initially empty)

Customize the nodes within which the atoms are set with Ti*k*Z keys.

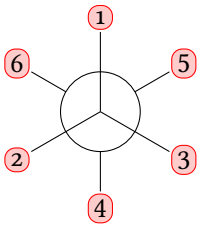
`newman » back-atoms = {`⟨*tikz*⟩`}` (initially empty)

Explicitly customize the nodes of the back atoms with Ti*k*Z keys.

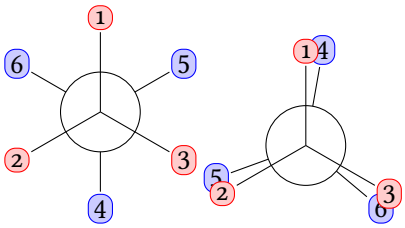
```
\chemsetup[newman]{angle=45} \newman{}
\newman[scale=.75,ring={draw=blue,fill=blue!20}]{}
```



```
\chemsetup[newman]{atoms={draw=red,fill=red!20,inner sep=2pt,rounded corners}}
\newman{1,2,3,4,5,6}
```




```
\chemsetup[newman]{
  atoms = {draw=red,fill=red!20,inner sep=2pt,rounded corners},
  back-atoms = {draw=blue,fill=blue!20,inner sep=2pt,rounded corners}
}
\newman{1,2,3,4,5,6} \newman(170){1,2,3,4,5,6}
```



7.5. The `orbital` Module

The `orbital` module loads the `tikz` module. It provides the following command to create orbitals:

`\orbital[`⟨*options*⟩`]{`⟨*type*⟩`}`
Draw an orbital shape of type ⟨*type*⟩. This command uses Ti*k*Z internally.

There are the following types available for ⟨*type*⟩:

`s` `p` `sp` `sp2` `sp3`

```
\orbital{s} \orbital{p} \orbital{sp} \orbital{sp2} \orbital{sp3}
```

Depending on the type you have different options to modify the orbitals:

`orbital` » `phase` = `+|-` Default: `+`
changes the phase of the orbital (all types)

`orbital` » `scale` = {⟨*factor*⟩} Default: `1`
changes the size of the orbital (all types)

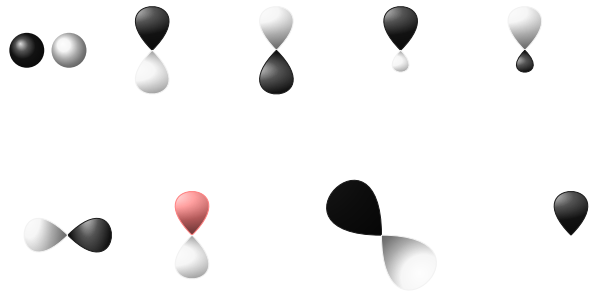
`orbital` » `color` = {⟨*color*⟩} Default: `black`
changes the color of the orbital (all types)

`orbital` » `angle` = {⟨*angle*⟩} Default: `0`
rotates the orbitals with a p contribution counter clockwise (all types except `s`)

`orbital` » `half` = `true|false` Default: `false`
displays only half an orbital (only `p`)

```
\orbital{s} \orbital[phase=-]{s}
\orbital{p} \orbital[phase=-]{p}
\orbital{sp3} \orbital[phase=-]{sp3}

\orbital[angle=0]{p} \orbital[color=red!50]{p}
\orbital[angle=135,scale=1.5]{p} \orbital[half]{p}
```



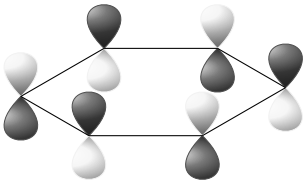
Additionally there are two options, with which the Ti*k*Z behaviour can be changed.

`orbital` » `overlay` = `true|false`
The orbital "doesn't need space"; it is displayed with the Ti*k*Z option `overlay`.

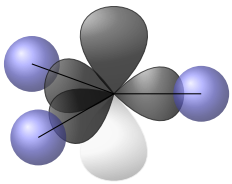
`orbital` » `opacity` = {⟨*num*⟩}
The orbital becomes transparent; ⟨*value*⟩ can have values between `1` (fully opaque) to `0` (invisible).

```
\vspace{7mm}
\chemsetup[orbital]{
  overlay,
  p/color = black!70
}
```

```
\setbondoffset{0pt}
\chemfig{
  ?\orbital{p}
  -[,1.3]{\orbital[phase=-]{p}}
  -[:30,1.1]\orbital{p}
  -[:150,.9]{\orbital[phase=-]{p}}
  -[4,1.3]\orbital{p}
  -[:-150,1.1]{\orbital[phase=-]{p}}?
}
\vspace{7mm}
```



```
\vspace{7mm}
\setbondoffset{0pt}
\chemsetup[orbital]{
  overlay ,
  opacity = .75 ,
  p/scale = 1.6 ,
  s/color = blue!50 ,
  s/scale = 1.6
}
\chemfig{
  \orbital{s}
  -[:-20]{\orbital[scale=2]{p}}
         {\orbital[half,angle=0]{p}}
         {\orbital[angle=170,half]{p}}
         {\orbital[angle=-150,half]{p}}
  (-[:-150]\orbital{s})-\orbital{s}
}
\vspace{1cm}
```



7.6. The reactions Module

The reactions module loads the chemformula module and the mathtools package [MRW13].

7.6.1. Predefined Environments

You can use these environments for numbered…

`\begin{reaction}`
A single reaction where CHEMFORMULA code is placed directly in the environment body. A wrapper around the `equation` environment. The environment body is parsed with `\ch` or `\ce` depending on the value of the `formula` option, see section 6.2 starting on page 26.

`\begin{reactions}`
Several aligned reactions. A wrapper around amsmath's `align` environment. The environment body is parsed with `\ch` or `\ce` depending on the value of the `formula` option, see section 6.2 starting on page 26.

…and their starred versions for unnumbered reactions.

`\begin{reaction*}`
A wrapper around the `equation*` environment. The environment body is parsed with `\ch` or `\ce` depending on the value of the `formula` option, see section 6.2 starting on page 26.

`\begin{reactions*}`
A wrapper around amsmath's `align*` environment. The environment body is parsed with `\ch` or `\ce` depending on the value of the `formula` option, see section 6.2 starting on page 26.

With those environments you can create (un)numbered reaction equations similar to mathematical equations.

Theses environments use the `equation`/`equation*` environments or the `align`/`align*` environments, respectively, to display the reactions.

```
1 Reaction with counter:
2 \begin{reaction}
3   A -> B
4 \end{reaction}
```

Reaction with counter:

$$\mathrm{A} \longrightarrow \mathrm{B} \qquad \{1\}$$

```
1 Reaction without counter:
2 \begin{reaction*}
3   C -> D
4 \end{reaction*}
```

Reaction without counter:

$$\mathrm{C} \longrightarrow \mathrm{D}$$

```
Several aligned reactions with counter:
\begin{reactions}
  A     &-> B + C \\
  D + E &-> F
\end{reactions}
```

Several aligned reactions with counter:

$$\mathrm{A} \longrightarrow \mathrm{B} + \mathrm{C} \qquad \{2\}$$
$$\mathrm{D} + \mathrm{E} \longrightarrow \mathrm{F} \qquad \{3\}$$

```
Several aligned reactions without counter:
\begin{reactions*}
  G     &-> H + I \\
  J + K &-> L
\end{reactions*}
```

Several aligned reactions without counter:

$$\mathrm{G} \longrightarrow \mathrm{H} + \mathrm{I}$$
$$\mathrm{J} + \mathrm{K} \longrightarrow \mathrm{L}$$

If you want to change the layout of the counter tags, you can use

`\renewtagform{`⟨*tagname*⟩`}[`⟨*format*⟩`]{`⟨*right delimiter*⟩`}{`⟨*left delimiter*⟩`}`
Provided by the mathtools package.

```
\renewtagform{reaction}[R \textbf]{[}{]}
\begin{reaction}
  H2O + CO2 <=> H2CO3
\end{reaction}
```

$$\mathrm{H_2O} + \mathrm{CO_2} \rightleftharpoons \mathrm{H_2CO_3} \qquad [\mathrm{R}\ \mathbf{4}]$$

The use of $\mathcal{AMS}$math's `\intertext` is possible:

```
\begin{reactions}
  A + 2 B &-> 3 C + D "\label{rxn:test}"
  \intertext{Some text in between aligned reactions}
  3 E + F &<=> G + 1/2 H
\end{reactions}
See reaction~\ref{rxn:test}.
```

$$\mathrm{A} + 2\,\mathrm{B} \longrightarrow 3\,\mathrm{C} + \mathrm{D} \tag{5}$$

Some text in between aligned reactions

$$3\,\mathrm{E} + \mathrm{F} \rightleftharpoons \mathrm{G} + \frac{1}{2}\,\mathrm{H} \tag{6}$$

See reaction 5.

7.6.2. Own Reactions

You can create new types of reactions with the command:

`\NewChemReaction{`⟨*name*⟩`}[`⟨*number of arguments*⟩`]{`⟨*math name*⟩`}`
⟨*name*⟩ will be the name of the new chem environment. ⟨*math name*⟩ is the underlying math environment. Gives an error if ⟨*name*⟩ already exists.

`\RenewChemReaction{`⟨*name*⟩`}[`⟨*number of arguments*⟩`]{`⟨*math name*⟩`}`
⟨*name*⟩ is the name of the renewed chem environment. ⟨*math name*⟩ is the underlying math environment. Gives an error if ⟨*name*⟩ does not exist.

`\DeclareChemReaction{`⟨*name*⟩`}[`⟨*number of arguments*⟩`]{`⟨*math name*⟩`}`
⟨*name*⟩ will be the name of the chem environment. ⟨*math name*⟩ is the underlying math environment.

`\ProvideChemReaction{`⟨*name*⟩`}[`⟨*number of arguments*⟩`]{`⟨*math name*⟩`}`
⟨*name*⟩ will be the name of the new chem environment. ⟨*math name*⟩ is the underlying math environment. The new environment is only defined if it doesn't exist, yet.

```
\NewChemReaction{reaction}  {equation}
\NewChemReaction{reaction*} {equation*}
\NewChemReaction{reactions} {align}
\NewChemReaction{reactions*}{align*}
```

Let's suppose, you'd like to have the alignment behaviour of the `alignat` environment for CHEMFORMULA reactions. You could do the following:

```
\NewChemReaction{reactionsat}[1]{alignat}
```

With this the `reactionsat` environment is defined.

```
\NewChemReaction{reactionsat}[1]{alignat}
\NewChemReaction{reactionsat*}[1]{alignat*}
\begin{reactionsat}{3}
  A     &-> B     &&-> C     &&-> D \\
  aaaaa &-> bbbbb &&-> ccccc &&-> ddddd
\end{reactionsat}
\begin{reactionsat*}{2}
  A     &-> B     & C            &-> D  \\
  aaaaa &-> bbbbb &\quad{} ccccc &-> ddddd
\end{reactionsat*}
```

$$\begin{aligned} A &\longrightarrow B &&\longrightarrow C &&\longrightarrow D && \{7\}\\ aaaaa &\longrightarrow bbbbb &&\longrightarrow ccccc &&\longrightarrow ddddd && \{8\} \end{aligned}$$

$$\begin{aligned} A &\longrightarrow B & C &\longrightarrow D\\ aaaaa &\longrightarrow bbbbb & ccccc &\longrightarrow ddddd \end{aligned}$$

7.6.3. List of Reactions

The `reactions` module also provides a command to display a list of the reactions created with the `reaction` environment.

`\listofreactions`
Print a list of reactions.

```
\listofreactions
```

List of Reactions

The output of this list can be modified by two options:

`reactions` » `list-name` = {⟨*name of the list*⟩} Default: `\ChemTranslate{list-of-reactions}`
Let's you set the name of the list manually. The default name is language dependent, see section 6.5 starting on page 28.

`reactions` » `list-entry` = {⟨*prefix to each entry*⟩} Default: `\ChemTranslate{reaction}`
Let's you set a prefix to each list entry. The default name is language dependent, see section 6.5 starting on page 28.

`reactions` » `list-heading-cmd` = {⟨*code*⟩} Default: `\section*{#1}`
Introduced in version 5.2
The macro that is called at the beginning of the list. Inside of ⟨*code*⟩ `#1` refers to the actual heading of the list. The default setting is not entirely true: if a macro `\chapter` is defined `\chapter*{#1}` is used.

Instead of using the option `list-name` you also could redefine `\reactionlistname`.

The list lists all reactions with a number and disregards reactions without number. All reaction environments without star have an optional argument which let's you add a description (or caption) for the entry in the list.

```
\begin{reaction}[Autoprotolyse]
  2 H2O <<=> H3O+ + OH-
\end{reaction}
```

$$2\,H_2O \rightleftharpoons H_3O^+ + OH^- \qquad \{9\}$$

If you use the `reactions` environment this will not work, though. In this case you can use

`\AddRxnDesc{`⟨*description*⟩`}`
Add a description to a reaction.

```
\begin{reactions}
  "\chlewis{0.}{Cl}" + CH4  &
    -> HCl + "\chlewis{180.}{C}" H3 \AddRxnDesc{first~step~of~chain} \\
  "\chlewis{180.}{C}" H3 + Cl2 &
    -> CH3Cl + "\chlewis{0.}{Cl}" \AddRxnDesc{second~step~of~chain}
\end{reactions}
```

$$\mathrm{Cl^\bullet + CH_4 \longrightarrow HCl + {^\bullet CH_3}} \quad \{10\}$$
$$\mathrm{{^\bullet CH_3} + Cl_2 \longrightarrow CH_3Cl + Cl^\bullet} \quad \{11\}$$

7.7. The redox Module

The redox module loads the modules tikz and xfrac. It also loads the packages mathtools [MRW13] and relsize [Ars13].

7.7.1. Oxidation Numbers

Regarding the typesetting of oxidation numbers *The IUPAC Green Book* [Coh+08] says the following:

> Oxidation numbers are denoted by positive or negative Roman numerals or by zero [...]
>
> *Examples* $\mathrm{Mn^{VII}}$, manganese (VII), $\mathrm{O^{-II}}$, $\mathrm{Ni^0}$ *[Coh+08, p. 50]*

The following command is provided to set oxidation numbers:

`\ox*[`⟨*options*⟩`]{`⟨*number*⟩`,`⟨*atom*⟩`}`
Places ⟨*number*⟩ as right superscript to ⟨*atom*⟩; ⟨*number*⟩ has to be a (rational) number! ⟨*atom*⟩ is treated as a CHEMFORMULA formula, like it would be in `\chcpd`.

```
1 \ox{+1,Na}, \ox{2,Ca}, \ox{-2,S}, \ox{-1,F}
```

$\mathrm{Na^{I}}$, $\mathrm{Ca^{II}}$, $\mathrm{S^{-II}}$, $\mathrm{F^{-I}}$

There are a number of options that can be used to modify the typeset result:

`redox » parse = true|false` Default: `true`
When `false` an arbitrary entry can be used for `<number>`.

`redox » roman = true|false` Default: `false`
Switches from roman to arabic numbers.

`redox » pos = top|super|side` Default: `super`
`top` places ⟨*number*⟩ above ⟨*atom*⟩, `super` to the upper right as superscript and `side` to the right and inside brackets. Both `super` and `side` follow IUPAC recommendation, `top` does not!

`redox » explicit-sign = true|false` Default: `false`
Shows the + for positiv numbers and the ± for 0.

`redox » decimal-marker = comma|point` Default: `point`
Choice for the decimal marker for formal oxidation numbers like $\mathrm{X^{1.2}}$.

`redox » align = center|right` Default: `center`
Center the oxidation number relative to the atom or right-align it.

redox » side-connect = {⟨code⟩} Default: \,
Code that is inserted between atom and oxidation number if pos = {side} is used.

redox » text-frac = {⟨cs⟩} Default: \chemfrac[text]{#1}{#2}
The fraction macro that is used for fractions if pos = {side} is used. ⟨cs⟩ must be a macro that takes two mandatory arguments, the first for the numerator and the second for the denominator.

redox » super-frac = {⟨cs⟩} Default: \chemfrac[superscript]{#1}{#2}
The fraction macro that is used for fractions if pos = {top} or pos = {super} is used. ⟨cs⟩ must be a macro that takes two mandatory arguments, the first for the numerator and the second for the denominator.

```
\ox[roman=false]{2,Ca} \ox{2,Ca} \\
\ox[pos=top]{3,Fe}-Oxide \\
\ox[pos=side]{3,Fe}-Oxide \\
\ox[parse=false]{?,Mn} \\
\ox[pos=top,align=right]{2,Ca}
```

$\mathrm{Ca}^{2}\ \mathrm{Ca}^{\mathrm{II}}$
$\overset{\mathrm{III}}{\mathrm{Fe}}$-Oxide
Fe (III)-Oxide
$\mathrm{Mn}^{?}$
$\overset{\mathrm{II}}{\mathrm{Ca}}$

The pos = {top} variant also can be set with the shortcut \ox*:

```
\ox{3,Fe} \ox*{3,Fe}
```

$\mathrm{Fe}^{\mathrm{III}}\ \overset{\mathrm{III}}{\mathrm{Fe}}$

Using the explicit-sign option will always show the sign of the oxidation number:

```
\chemsetup[redox]{explicit-sign = true}
\ox{+1,Na}, \ox{2,Ca}, \ox{-2,S}, \ch{"\ox{0,F}" {}2}
```

$\mathrm{Na}^{+\mathrm{I}}$, $\mathrm{Ca}^{+\mathrm{II}}$, $\mathrm{S}^{-\mathrm{II}}$, $\mathrm{F}^{0}{}_{2}$

```
\chemsetup[redox]{pos=top}
Compare \ox{-1,O2^2-} to \ch{"\ox{-1,O}" {}2^2-}
```

Compare $\overset{-\mathrm{I}}{\mathrm{O}}_{2}^{2-}$ to $\overset{-\mathrm{I}}{\mathrm{O}}_{2}^{2-}$

Sometimes one might want to use formal oxidation numbers like 0.5 or ⅓:

```
\chemsetup[redox]{pos=top}
\ox{.5,Br2}
\ch{"\ox{1/3,I}" {}3+}

\chemsetup[redox]{pos=side}
\ox{1/3,I3+}
```

$\overset{0.5}{\mathrm{Br_2}}\ \overset{⅓}{\mathrm{I_3^+}}$

$\mathrm{I_3^+}\ (⅓)$

The fraction is displayed with the help of the xfrac package [L3Pb]. For more details on how CHEMMACROS uses it read section 8.2 starting on page 60.

7.7.2. Redox Reactions

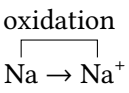
CHEMMACROS provides two commands to visualize the transfer of electrons in redox reactions. Both commands are using Ti*k*Z.

`\OX{`⟨*name*⟩`,`⟨*atom*⟩`}`
Label ⟨*atom*⟩ with the label ⟨*name*⟩.

`\redox(`⟨*name1*⟩`,`⟨*name2*⟩`)[`⟨*tikz*⟩`][`⟨*num*⟩`]{`⟨*text*⟩`}`
Connect two ⟨*atom*⟩s previously labelled with `\OX`. Only the first argument `(`⟨*name1*⟩`,`⟨*name2*⟩`)` is required, the others are all optional.

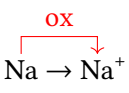
`\OX` places ⟨*atom*⟩ into a node, which is named with ⟨*name*⟩. If you have set two `\OX`, they can be connected with a line using `\redox`. To do so the names of the two nodes that are to be connected are written in the round braces. Since `\redox` draws a `tikzpicture` with options `remember picture,overlay`, the document needs to be *compiled at least two times*.

```
\vspace{7mm}
\OX{a,Na} $\rightarrow$ \OX{b,Na}\pch\redox(a,b){oxidation}
```



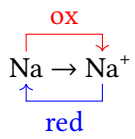
This line can be customized using Ti*k*Z keys in `[`⟨*tikz*⟩`]`:

```
\vspace{7mm}
\OX{a,Na} $\rightarrow$ \OX{b,Na}\pch\redox(a,b)[->,red]{ox}
```



With the argument `[`⟨*num*⟩`]` the length of the vertical parts of the line can be adjusted. The default length is `.6em`. This length is multiplied with ⟨*num*⟩. If you use a negative value the line is placed *below* the text.

```
\vspace{7mm}
\OX{a,Na} $\rightarrow$ \OX{b,Na}\pch
\redox(a,b)[->,red]{ox}
\redox(a,b)[<-,blue][-1]{red}
\vspace{7mm}
```

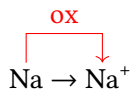


The default length of the vertical lines can be customized with the option

redox » dist = {⟨*dim*⟩} Default: .6em

A TEX dimension.

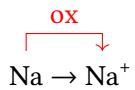
```
\vspace{7mm}
\chemsetup{redox/dist=1em}
\OX{a,Na} $\rightarrow$ \OX{b,Na}\pch\redox(a,b)[->,red]{ox}
```



redox » sep = {⟨*dim*⟩} Default: .2em

The option can be used to change the distance between the atom and the beginning of the line.

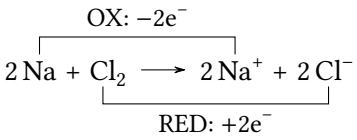
```
\vspace{7mm}
\chemsetup{redox/sep=.5em}
\OX{a,Na} $\rightarrow$ \OX{b,Na}\pch\redox(a,b)[->,red]{ox}
```



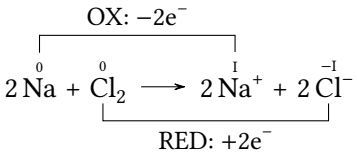
Examples

```
\vspace{7mm}
\ch{
  2 "\OX{o1,Na}" + "\OX{r1,Cl}" {}2
  ->
```

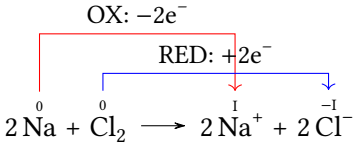
```
  2 "\OX{o2,Na}" {}+ + 2 "\OX{r2,Cl}" {}-
}
\redox(o1,o2){\small OX: $- 2\el$}
\redox(r1,r2)[][-1]{\small RED: $+ 2\el$}
\vspace{7mm}
```



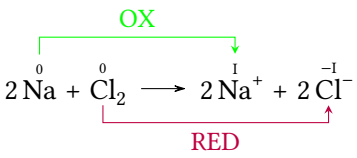

```
\vspace{7mm}
\ch{
  2 "\OX{o1,\ox*{0,Na}}" + "\OX{r1,\ox*{0,Cl}}" {}2
  ->
  2 "\OX{o2,\ox*{+1,Na}}" {}+ + 2 "\OX{r2,\ox*{-1,Cl}}" {}-
}
\redox(o1,o2){\small OX: $- 2\el$}
\redox(r1,r2)[][-1]{\small RED: $+ 2\el$}
\vspace{7mm}
```




```
\vspace{14mm}
\ch{
  2 "\OX{o1,\ox*{0,Na}}" + "\OX{r1,\ox*{0,Cl}}" {}2
  ->
  2 "\OX{o2,\ox*{+1,Na}}" {}+ + 2 "\OX{r2,\ox*{-1,Cl}}" {}-
}
\redox(o1,o2)[draw=red,->][3.33]{\small OX: $- 2\el$}
\redox(r1,r2)[draw=blue,->]{\small RED: $+ 2\el$}
```



```
\vspace{7mm}
\ch{
  2 "\OX{o1,\ox*{0,Na}}" + "\OX{r1,\ox*{0,Cl}}" {}2
  -> 2 "\OX{o2,\ox*{+1,Na}}" {}+ + 2 "\OX{r2,\ox*{-1,Cl}}" {}-
}
\redox(o1,o2)[green,-stealth]{\small OX}
\redox(r1,r2)[purple,-stealth][-1]{\small RED}
\vspace{7mm}
```



7.8. The scheme Module

The scheme module loads the chemnum package [Nie15c] and defines a floating environment \begin{scheme}. That is, it *only* defines this float if no environment scheme exists at the end of the preamble. The module checks for different available float defining methods, in *this* order:

- If the current class is a KOMA-Script class \DeclareNewTOC will be used.
- If the current class is memoir, memoir's methods are used.
- If the package tocbasic has been loaded \DeclareNewTOC will be used.
- If the package newfloat has been loaded \DeclareFloatingEnvironment will be used.

Introduced in version 5.1

- If the package floatrow has been loaded its method will be used.
- If the package float has been loaded its method will be used.
- If neither of the above the "manual" method is used. This means the environment is defined the same way like figure is defined in the article class or the book class, depending if \chapter is defined or not.

The list name and the caption name both are translated to the language specified according to the lang option and the provided translations, see section 6.5 starting on page 28 for details. If you want to manually change them then redefine these macros after begin document:

\listschemename
The name of the list of schemes.

\schemename
The name used in captions.

The list of schemes is printed as expected with

\listofschemes

7.9. The spectroscopy Module

The spectroscopy module loads the chemformula module and the siunitx package [Wri15].

7.9.1. The \NMR Command

When you're trying to find out if a compound is the one you think it is often NMR spectroscopy is used. The experimental data are typeset similar to this:

$$^{1}\text{H-NMR (400 MHz, } CDCl_3\text{): } \delta = 1.59$$

The spectroscopy module provides a command which simplifies the input.

\NMR*{⟨num⟩,⟨element⟩}(⟨num⟩,⟨unit⟩)[⟨solvent⟩]
Typeset nuclear magnetic resonance data. ⟨num⟩ is a valid siunitx number input, ⟨unit⟩ is a valid siunitx unit input. ⟨solvent⟩ is any valid CHEMFORMULA input as in \chcpd.

All Argument are optional! Without arguments we get:

```
\NMR \par
\NMR*
```

^{1}H-NMR: δ
^{1}H-NMR

The first argument specifies the kind of NMR:

```
\NMR{13,C}
```

^{13}C-NMR: δ

The second argument sets the frequency (in MHz):

```
\NMR(400)
```

^{1}H-NMR (400 MHz): δ

You can choose another unit:

```
\NMR(4e8,\hertz)
```

^{1}H-NMR (4×10^{8} Hz): δ

Please note that the setup of siunitx also affects this command:

```
\sisetup{exponent-product=\cdot}
\NMR(4e8,\hertz)
```

^{1}H-NMR ($4 \cdot 10^{8}$ Hz): δ

The third argument specifies the solvent:

```
\NMR[CDCl3]
```
^{1}H-NMR ($CDCl_3$): δ

7.9.2. Short Cuts

It is possible to define short cut commands for specific nuclei.

`\NewChemNMR{`⟨*cs*⟩`}{`⟨*num*⟩`,`⟨*atom*⟩`}`
Define a new shortcut macro for typesetting a certain type of magnetic resonence data. Gives an error if ⟨*cs*⟩ already exists.

`\DeclareChemNMR{`⟨*cs*⟩`}{`⟨*num*⟩`,`⟨*atom*⟩`}`
Define a new shortcut macro for typesetting a certain type of magnetic resonence data. Overwrites an existing macro.

`\RenewChemNMR{`⟨*cs*⟩`}{`⟨*num*⟩`,`⟨*atom*⟩`}`
Redefine an existing shortcut macro for typesetting a certain type of magnetic resonence data. Gives an error if ⟨*cs*⟩ doesn't exist.

`\ProvideChemNMR{`⟨*cs*⟩`}{`⟨*num*⟩`,`⟨*atom*⟩`}`
Define a new shortcut macro for typesetting a certain type of magnetic resonence data. ⟨*cs*⟩ is only defined if it doesn't exist, yet.

This defines a command with the same arguments as `\NMR` *except* for `{`⟨*num*⟩`,`⟨*atom*⟩`}`:

```
\NewChemNMR\HNMR{1,H}%
\NewChemNMR\CNMR{13,C}%
\CNMR*(100) \par
\HNMR*(400)
```
^{13}C-NMR (100 MHz)
^{1}H-NMR (400 MHz)

7.9.3. An Environment to Typeset Experimental Data

The `spectroscopy` module provides an environment to ease the input of experimental data.

`\begin{experimental}`
Environment for the output of experimental data. Inside the environment the following commands are defined.

`\data{`⟨*type*⟩`}[`⟨*specification*⟩`]`
Type of data, *e. g.* IR, MS… The optional argument takes further specifications which are output in parentheses.

`\data*{`⟨*type*⟩`}[`⟨*specification*⟩`]`
Like `\data` but changes the `=` into a `:`, given that `use-equal = {true}` is used.

`\NMR{`⟨*num*⟩`,`⟨*elem*⟩`[`⟨*coupling core*⟩`]}(`⟨*num*⟩`,`⟨*unit*⟩`)[`⟨*solvent*⟩`]`
This command gets an additional argument: `\NMR{13,C[^1H]}` ^{13}C{^{1}H}-NMR: δ

`\J(`⟨*bonds*⟩`;`⟨*nuclei*⟩`)[`⟨*unit*⟩`]{`⟨*list of nums*⟩`}`
Coupling constant, values are input separated by `;` (NMR). The arguments `(`⟨*bonds*⟩`;`⟨*nuclei*⟩`)` and `[`⟨*unit*⟩`]` are optional and enable further specifications of the coupling.

`\#{`⟨*num*⟩`}`
Number of nuclei (NMR).

`\pos{`⟨*num*⟩`}`
Position of nuclues (NMR).

`\val{`⟨*num*⟩`}`
A number, an alias of siunitx' `\num{`⟨*num*⟩`}`.

`\val{`⟨*num1*⟩`--`⟨*num2*⟩`}`
An alias of siunitx' `\numrange{`⟨*num1*⟩`}{`⟨*num2*⟩`}`.

```
\begin{experimental}
  \data{type1} Data.
  \data{type2}[specifications] More data.
  \data*{type3} Even more data.
\end{experimental}
```

---

type1 Data. type2 (specifications) More data. type3 Even more data.

Customization The output of the environment and of the NMR commands can be customized be a number of options. For historical reasons they all belong to the module `nmr`.

`spectroscopy » unit = {`⟨*unit*⟩`}` Default: `\mega\hertz`
The used default unit.

`spectroscopy » nucleus = {`⟨*num*⟩`,`⟨*atom*⟩`}` Default: `{1,H}`
The used default nucleus.

`spectroscopy » connector = {`⟨*code*⟩`}` Default: `-`
Places ⟨*code*⟩ between the nucleus and the method.

`spectroscopy » method = {`⟨*code*⟩`}` Default: `NMR`
The measuring method.

`spectroscopy » format = {`⟨*commands*⟩`}` (initially empty)
For example `\bfseries`.

`spectroscopy » pos-number = side|sub|super` Default: `side`
Position of the number next to the atom.

`spectroscopy » coupling-symbol = {`⟨*code*⟩`}` Default: `J`
The symbol used for the coupling constant.

`spectroscopy » coupling-unit = {`⟨*unit*⟩`}` Default: `\hertz`
A siunitx unit.

`spectroscopy » coupling-pos = side|sub` Default: `side`
Placement of the coupling nuclei next to the symbol J (or rather the symbol specified with option `coupling-symbol`).

spectroscopy » coupling-nuclei-pre = {⟨code⟩} Default: (
Code inserted before the coupling nuclei when coupling-pos = {side}.

spectroscopy » coupling-nuclei-post = {⟨code⟩} Default:)
Code inserted after the coupling nuclei when coupling-pos = {side}.

spectroscopy » coupling-bonds-pre = {⟨code⟩} (initially empty)
Code inserted before the coupling bonds.

spectroscopy » coupling-bonds-post = {⟨code⟩} Default: \!
Code inserted after the coupling bonds.

spectroscopy » coupling-pos-cs = {⟨cs⟩} Default: \@firstofone
Set the macro that prints the number set with the \pos macro. This needs to be a command with one mandatory argument.

spectroscopy » atom-number-cs = {⟨cs⟩} Default: \@firstofone
Set the macro that prints the number set with the \# macro. This needs to be a command with one mandatory argument.

spectroscopy » atom-number-space = {⟨dim⟩} Default: .16667em
Introduced in version 5.3
Horizontal space inserted between number and atom (printed by \#).

spectroscopy » parse = true|false Default: true
Treat the solvent as CHEMFORMULA formula or not.

spectroscopy » delta = {⟨tokens⟩} (initially empty)
The ⟨tokens⟩ are added after δ.

spectroscopy » list = true|false Default: false
The environment nmr is formatted as a list

spectroscopy » list-setup = {⟨setup⟩}
Setup of the list. See below for the default settings.

spectroscopy » use-equal = true|false Default: false
Add egual sign after \NMR and \data.

The default setup of the list:

```
\topsep\z@skip \partopsep\z@skip
\itemsep\z@ \parsep\z@ \itemindent\z@
\leftmargin\z@
```

```
\begin{experimental}[format=\bfseries]
  \data{type1} Data.
  \data{type2}[specifications] More data.
  \data*{type3} Even more data.
\end{experimental}
```

type1 Data. **type2 (specifications)** More data. **type3** Even more data.

The command \NMR and all commands defined through \NewChemNMR can be used like \data for the NMR data.

```
\begin{experimental}[format=\bfseries,use-equal]
  \data{type1} Data.
  \data{type2}[specifications] More data.
  \NMR Even more data.
\end{experimental}
```

---

type1 = Data. **type2 (specifications)** = More data. **^{1}H-NMR:** δ =Even more data.

An Example The code below is shown with different specifications for ⟨*options*⟩. Of course options can also be chosen with \chemsetup.

```
\sisetup{separate-uncertainty,per-mode=symbol,detect-all,range-phrase=--}
\begin{experimental}[<optionen>]
  \data*{yield} \SI{17}{\milli\gram} yellow needles (\SI{0.04}{\milli\mole},
    \SI{13}{\percent}).
  %
  \data{mp.} \SI{277}{\celsius} (DSC).
  %
  \NMR(600)[CDCl3] \val{2.01} (s, \#{24}, \pos{5}), \val{2.31} (s, \#{12},
  \pos{1}), \val{6.72--6.74} (m, \#{2}, \pos{11}), \val{6.82} (s, \#{8},
  \pos{3}), \val{7.05--7.07} (m, \#{2}, \pos{12}), \val{7.39--7.41} (m, \#{4},
  \pos{9}), \val{7.48--7.49} (m, \#{4}, \pos{8}).
  %
  \NMR{13,C}(150)[CDCl3] \val{21.2} ($+$, \#{4}, \pos{1}), \val{23.4} ($+$,
  \#{8}, \pos{5}), \val{126.0} ($+$, \#{4}, \pos{9}), \val{128.2} ($+$, \#{8},
  \pos{3}), \val{130.8} ($+$, \#{2}, \pos{12}), \val{133.6} ($+$, \#{2},
  \pos{11}), \val{137.0} ($+$, \#{4}, \pos{8}), \val{138.6} (q, \#{4},
  \pos{2}), \val{140.6} (q, \#{2}, \pos{10}), \val{140.8} (q, \#{8}, \pos{4}),
  \val{141.8} (q, \#{4}, \pos{6}), \val{145.6} (q, \#{2}, \pos{7}).
  %
  \data{MS}[DCP, EI, \SI{60}{\electronvolt}] \val{703} (2, \ch{M+}), \val{582}
  (1), \val{462} (1), \val{249} (13), \val{120} (41), \val{105} (100).
  %
  \data{MS}[\ch{MeOH + H2O + KI}, ESI, \SI{10}{\electronvolt}] \val{720} (100,
  \ch{M+ + OH-}), \val{368} (\ch{M+ + 2 OH-}).
  %
  \data{IR}[KBr] \val{3443} (w), \val{3061} (w), \val{2957} (m), \val{2918}
  (m), \val{2856} (w), \val{2729} (w), \val{1725} (w), \val{1606} (s),
  \val{1592} (s), \val{1545} (w), \val{1446} (m), \val{1421} (m), \val{1402}
  (m), \val{1357} (w), \val{1278} (w), \val{1238} (s), \val{1214} (s),
  \val{1172} (s), \val{1154} (m), \val{1101} (w), \val{1030} (w), \val{979}
  (m), \val{874} (m), \val{846} (s), \val{818} (w), \val{798} (m), \val{744}
  (w), \val{724} (m), \val{663} (w), \val{586} (w), \val{562} (w), \val{515}
```

```
  (w).
  %
  \data*{UV-Vis} \SI{386}{\nano\metre} ($\varepsilon = \val{65984}$),
  \SI{406}{\nano\metre} ($\varepsilon = \val{65378}$).
  %
  \data*{quantum yield} $\Phi = \val{0.74+-0.1}$\,.
\end{experimental}
```

Nearly Standard Output with these options:

```
delta=(ppm),pos-number=sub,use-equal
```

yield: 17 mg yellow needles (0.04 mmol, 13 %). mp. = 277 °*C* (DSC). ^{1}H-NMR (600 MHz, $CDCl_3$): δ (ppm) = 2.01 (s, 24 H, H_5), 2.31 (s, 12 H, H_1), 6.72–6.74 (m, 2 H, H_{11}), 6.82 (s, 8 H, H_3), 7.05–7.07 (m, 2 H, H_{12}), 7.39–7.41 (m, 4 H, H_9), 7.48–7.49 (m, 4 H, H_8). ^{13}C-NMR (150 MHz, $CDCl_3$): δ (ppm) = 21.2 (+, 4 C, C_1), 23.4 (+, 8 C, C_5), 126.0 (+, 4 C, C_9), 128.2 (+, 8 C, C_3), 130.8 (+, 2 C, C_{12}), 133.6 (+, 2 C, C_{11}), 137.0 (+, 4 C, C_8), 138.6 (q, 4 C, C_2), 140.6 (q, 2 C, C_{10}), 140.8 (q, 8 C, C_4), 141.8 (q, 4 C, C_6), 145.6 (q, 2 C, C_7). MS (DCP, EI, 60 eV) = 703 (2, M^+), 582 (1), 462 (1), 249 (13), 120 (41), 105 (100). MS (MeOH + H_2O + KI, ESI, 10 eV) = 720 (100, $M^+ + OH^-$), 368 ($M^+ + 2\,OH^-$). IR (KBr) = 3443 (w), 3061 (w), 2957 (m), 2918 (m), 2856 (w), 2729 (w), 1725 (w), 1606 (s), 1592 (s), 1545 (w), 1446 (m), 1421 (m), 1402 (m), 1357 (w), 1278 (w), 1238 (s), 1214 (s), 1172 (s), 1154 (m), 1101 (w), 1030 (w), 979 (m), 874 (m), 846 (s), 818 (w), 798 (m), 744 (w), 724 (m), 663 (w), 586 (w), 562 (w), 515 (w). UV-Vis: 386 nm (ε = 65 984), 406 nm (ε = 65 378). quantum yield: $\Phi = 0.74 \pm 0.10$.

Formatted List Output with these options:

```
format=\bfseries,delta=(ppm),list=true,use-equal
```

yield: 17 mg yellow needles (0.04 mmol, 13 %).
mp. = 277 °*C* (DSC).
^{1}H-NMR (600 MHz, $CDCl_3$): δ (ppm) = 2.01 (s, 24 H, H_5), 2.31 (s, 12 H, H_1), 6.72–6.74 (m, 2 H, H_{11}), 6.82 (s, 8 H, H_3), 7.05–7.07 (m, 2 H, H_{12}), 7.39–7.41 (m, 4 H, H_9), 7.48–7.49 (m, 4 H, H_8).
^{13}C-NMR (150 MHz, $CDCl_3$): δ (ppm) = 21.2 (+, 4 C, C_1), 23.4 (+, 8 C, C_5), 126.0 (+, 4 C, C_9), 128.2 (+, 8 C, C_3), 130.8 (+, 2 C, C_{12}), 133.6 (+, 2 C, C_{11}), 137.0 (+, 4 C, C_8), 138.6 (q, 4 C, C_2), 140.6 (q, 2 C, C_{10}), 140.8 (q, 8 C, C_4), 141.8 (q, 4 C, C_6), 145.6 (q, 2 C, C_7).
MS (DCP, EI, 60 eV) = 703 (2, M^+), 582 (1), 462 (1), 249 (13), 120 (41), 105 (100).
MS (MeOH + H_2O + KI, ESI, 10 eV) = 720 (100, $M^+ + OH^-$), 368 ($M^+ + 2\,OH^-$).
IR (KBr) = 3443 (w), 3061 (w), 2957 (m), 2918 (m), 2856 (w), 2729 (w), 1725 (w), 1606 (s), 1592 (s), 1545 (w), 1446 (m), 1421 (m), 1402 (m), 1357 (w), 1278 (w), 1238 (s), 1214 (s), 1172 (s), 1154 (m), 1101 (w), 1030 (w), 979 (m), 874 (m), 846 (s), 818 (w), 798 (m), 744 (w), 724 (m), 663 (w), 586 (w), 562 (w), 515 (w).
UV-Vis: 386 nm (ε = 65 984), 406 nm (ε = 65 378).
quantum yield: $\Phi = 0.74 \pm 0.10$.

Crazy Output for these options:

```
format=\color{red}\itshape,
list=true,
delta=\textcolor{green}{\ch{M+ + H2O}},
pos-number=side,
coupling-unit=\mega\gram\per\square\second,
list-setup=,
use-equal
```

yield: 17 mg yellow needles (0.04 mmol, 13 %).

mp. = 277 °*C* (DSC).

^{1}*H-NMR (600 MHz, CDCl$_3$):* δ $M^+ + H_2O$ = 2.01 (s, 24 H, H-5), 2.31 (s, 12 H, H-1), 6.72–6.74 (m, 2 H, H-11), 6.82 (s, 8 H, H-3), 7.05–7.07 (m, 2 H, H-12), 7.39–7.41 (m, 4 H, H-9), 7.48–7.49 (m, 4 H, H-8).

^{13}*C-NMR (150 MHz, CDCl$_3$):* δ $M^+ + H_2O$ = 21.2 (+, 4 C, C-1), 23.4 (+, 8 C, C-5), 126.0 (+, 4 C, C-9), 128.2 (+, 8 C, C-3), 130.8 (+, 2 C, C-12), 133.6 (+, 2 C, C-11), 137.0 (+, 4 C, C-8), 138.6 (q, 4 C, C-2), 140.6 (q, 2 C, C-10), 140.8 (q, 8 C, C-4), 141.8 (q, 4 C, C-6), 145.6 (q, 2 C, C-7).

MS (DCP, EI, 60 eV) = 703 (2, M^+), 582 (1), 462 (1), 249 (13), 120 (41), 105 (100).

MS (MeOH + H_2O + KI, ESI, 10 eV) = 720 (100, $M^+ + OH^-$), 368 ($M^+ + 2\,OH^-$).

IR (KBr) = 3443 (w), 3061 (w), 2957 (m), 2918 (m), 2856 (w), 2729 (w), 1725 (w), 1606 (s), 1592 (s), 1545 (w), 1446 (m), 1421 (m), 1402 (m), 1357 (w), 1278 (w), 1238 (s), 1214 (s), 1172 (s), 1154 (m), 1101 (w), 1030 (w), 979 (m), 874 (m), 846 (s), 818 (w), 798 (m), 744 (w), 724 (m), 663 (w), 586 (w), 562 (w), 515 (w).

UV-Vis: 386 nm (ε = 65 984), 406 nm (ε = 65 378).

quantum yield: $\Phi = 0.74 \pm 0.10$.

7.10. The `thermodynamics` Module

The `thermodynamics` module loads the siunitx package [Wri15].

7.10.1. The `\state` Macro

`\state[`⟨*options*⟩`]{`⟨*symbol*⟩`}`

Typeset a state variable.

This macro can be used to write the thermodynamic state variables.

```
\state{A}, \state[subscript-left=f]{G} ,
\state[subscript-right=\ch{Na}]{E},
\state[superscript-right=\SI{1000}{\celsius}]{H}
```

$\Delta A^{\ominus}$, $\Delta_{\mathrm{f}} G^{\ominus}$, $\Delta E^{\ominus}_{\mathrm{Na}}$, $\Delta H^{1000\,^{\circ}C}$

These options are available:

`thermodynamics » pre` = {⟨*text*⟩} Default: `\changestate`
Code inserted before the variable. Inserted in text mode.

`thermodynamics » post` = {⟨*text*⟩} (initially empty)
Code inserted after the variable. Inserted in text mode.

`thermodynamics » superscript-left` = {⟨*text*⟩} (initially empty)
The left superscript. Inserted in text mode.

`thermodynamics » superscript-right` = {⟨*text*⟩} Default: `\standardstate`
The right superscript. Inserted in text mode.

`thermodynamics » superscript` = {⟨*text*⟩}
An alias of `superscript-right`.

`thermodynamics » subscript-left` = {⟨*text*⟩} (initially empty)
The left subscript. Inserted in text mode.

`thermodynamics » subscript-right` = {⟨*text*⟩} (initially empty)
The right subscript. Inserted in text mode.

`thermodynamics » subscript` = {⟨*text*⟩}
An alias of `subscript-left`.

7.10.2. Thermodynamic Variables

The `thermodynamics` module provides a few commands for specific thermodynamic variables:

`\enthalpy*[`⟨*options*⟩`](`⟨*subscript*⟩`){`⟨*value*⟩`}`
Typeset the amount of enthalpy.

`\entropy*[`⟨*options*⟩`](`⟨*subscript*⟩`){`⟨*value*⟩`}`
Typeset the amount of entropy.

`\gibbs*[`⟨*options*⟩`](`⟨*subscript*⟩`){`⟨*value*⟩`}`
Typeset the amount of Gibbs enthalpy.

Their usage is pretty much self-explaining:

```
\enthalpy{123} \par
\entropy{123} \par
\gibbs{123}
```

$\Delta H^{\ominus} = 123\,\mathrm{kJ\,mol^{-1}}$
$S^{\ominus} = 123\,\mathrm{J\,K^{-1}\,mol^{-1}}$
$\Delta G^{\ominus} = 123\,\mathrm{kJ\,mol^{-1}}$

The argument (⟨*subscript*⟩) adds a subscript for specification, * hides number and unit:

```
\enthalpy(r){123} \par
\enthalpy*{123} \par
```

$\Delta_r H^\ominus = 123\,\mathrm{kJ\,mol^{-1}}$
$\Delta H^\ominus$

thermodynamics » pre = {⟨*text*⟩} Default: \changestate
Code inserted before the variable. Inserted in text mode.

thermodynamics » post = {⟨*text*⟩} (initially empty)
Code inserted after the variable. Inserted in text mode.

thermodynamics » superscript-left = {⟨*text*⟩} (initially empty)
The left superscript. Inserted in text mode.

thermodynamics » superscript-right = {⟨*text*⟩} Default: \standardstate
The right superscript. Inserted in text mode.

thermodynamics » superscript = {⟨*text*⟩}
An alias of superscript-right.

thermodynamics » subscript-left = {⟨*text*⟩} (initially empty)
The left subscript. Inserted in text mode.

thermodynamics » subscript-right = {⟨*text*⟩} (initially empty)
The right subscript. Inserted in text mode.

thermodynamics » subscript = {⟨*text*⟩}
An alias of subscript-left.

thermodynamics » subscript-pos = left|right Default: left
Determines wether the subscript given in ({⟨*subscript*⟩}) is placed to the left or the right of the variable.

thermodynamics » symbol = {⟨*symbol*⟩} (initially empty)
The symbol of the variable. Inserted in math mode.

thermodynamics » unit = {⟨*unit*⟩} (initially empty)
A valid siunitx unit.

The default values depend on the command.

```
\enthalpy[unit=\kilo\joule]{-285} \par
\gibbs[pre=]{0} \par
\entropy[pre=$\Delta$,superscript
  =]{56.7}
```

$\Delta H^\ominus = -285\,\mathrm{kJ}$
$G^\ominus = 0\,\mathrm{kJ\,mol^{-1}}$
$\Delta S = 56.7\,\mathrm{J\,K^{-1}\,mol^{-1}}$

The unit is set corresponding to the rules of siunitx and depends on its settings:

```
\enthalpy{-1234.56e3} \par
\sisetup{
  per-mode=symbol,
  exponent-product=\cdot,
  output-decimal-marker={,},
  group-four-digits=true
}
\enthalpy{-1234.56e3}
```

$\Delta H^\ominus = -1234.56 \times 10^3\,\mathrm{kJ\,mol^{-1}}$
$\Delta H^\ominus = -1\,234{,}56 \cdot 10^3\,\mathrm{kJ/mol}$

7.10.3. Create New Variables or Redefine Existing Ones

`\NewChemState{`⟨*cs*⟩`}{`⟨*options*⟩`}`
Define new state commands like `\enthalpy`. Gives an error is ⟨*cs*⟩ already exists.

`\RenewChemState{`⟨*cs*⟩`}{`⟨*options*⟩`}`
Redefine existing state commands.

`\DeclareChemState{`⟨*cs*⟩`}{`⟨*options*⟩`}`
Like `\NewChemState` but gives now error if ⟨*cs*⟩ already exists.

`\ProvideChemState{`⟨*cs*⟩`}{`⟨*options*⟩`}`
Define new state commands like `\enthalpy`. Defines ⟨*cs*⟩ only if it is not defined, yet.

The argument ⟨*options*⟩ is a comma separated list of key/value options:

`thermodynamics » pre = {`⟨*text*⟩`}` Default: `\changestate`
Code inserted before the variable. Inserted in text mode.

`thermodynamics » post = {`⟨*text*⟩`}` (initially empty)
Code inserted after the variable. Inserted in text mode.

`thermodynamics » superscript-left = {`⟨*text*⟩`}` (initially empty)
The left superscript. Inserted in text mode.

`thermodynamics » superscript-right = {`⟨*text*⟩`}` Default: `\standardstate`
The right superscript.

`thermodynamics » superscript = {`⟨*text*⟩`}`
An alias of `superscript-right`.

`thermodynamics » subscript-left = {`⟨*text*⟩`}` (initially empty)
The left subscript. Inserted in text mode.

`thermodynamics » subscript-right = {`⟨*text*⟩`}` (initially empty)
The right subscript. Inserted in text mode.

`thermodynamics » subscript = {`⟨*text*⟩`}`
An alias of `subscript-left`.

`thermodynamics » subscript-pos = left|right` Default: `left`
Determines wether the subscript given in `(`⟨*subscript*⟩`)` is placed to the left or the right of the variable.

thermodynamics » symbol = {⟨*symbol*⟩} (initially empty)
The symbol of the variable.

thermodynamics » unit = {⟨*unit*⟩} (initially empty)
A valid siunitx unit.

```
\NewChemState\Helmholtz{ symbol=A , unit=\kilo\joule\per\mole }
\NewChemState\ElPot{ symbol=E , subscript-pos=right , superscript= , unit=\volt
  }
\Helmholtz{123.4} \par
\ElPot{-1.1} \par
\ElPot[superscript=0]($\ch{Sn}|\ch{Sn^2+}||\ch{Pb^2+}|\ch{Pb}$){0.01} \par
\RenewChemState\enthalpy{ symbol=h , unit=\joule} \par
\enthalpy(f){12.5}
```

$\Delta A^{\ominus} = 123.4\,\mathrm{kJ\,mol^{-1}}$
$\Delta E = -1.1\,\mathrm{V}$
$\Delta E^{o}_{\mathrm{Sn|Sn^{2+}||Pb^{2+}|Pb}} = 0.01\,\mathrm{V}$
$\Delta_f h^{\ominus} = 12.5\,\mathrm{J}$

The existing commands have been defined like this:

```
\NewChemState \enthalpy{ symbol = H, unit = \kilo\joule\per\mole }
\NewChemState \entropy { symbol = S, unit = \joule\per\kelvin\per\mole, pre = }
\NewChemState \gibbs   { symbol = G, unit = \kilo\joule\per\mole }
```

So – for following thermodynamic conventions – one could define a molar and an absolute variable:

```
\RenewChemState\enthalpy{symbol=h,superscript=,unit=\kilo\joule\per\mole}%
  molar
\NewChemState\Enthalpy{symbol=H,superscript=,unit=\kilo\joule}% absolute
\enthalpy{-12.3} \Enthalpy{-12.3}
```

$\Delta h = -12.3\,\mathrm{kJ\,mol^{-1}}$ $\Delta H = -12.3\,\mathrm{kJ}$

7.11. The units Module

The units module loads the siunitx package [Wri15].

In chemistry some non-SI units are very common. siunitx provides the command

`\DeclareSIUnit{⟨cs⟩}{⟨unit⟩}`
Define ⟨*cs*⟩ to be a valid unit command inside siunitx' macros `\SI` and `\si` which represents ⟨*unit*⟩.

to add arbitrary units. CHEMMACROS uses that command to provide some units. Like all siunitx units they're only valid inside `\SI{⟨num⟩}{⟨unit⟩}` and `\si{⟨unit⟩}`.

`\atmosphere`
atm

`\atm`
atm

`\calory`
cal

`\cal`
cal

`\cmc`
cm^3

The units `\cmc`, `\molar`, and `\Molar` are defined by the package chemstyle as well. CHEMMACROS only defines them, if chemstyle is not loaded.

`\molar`
$\mathrm{mol\,dm}^{-3}$

`\moLar`
$\mathrm{mol\,L}^{-1}$

`\Molar`
M

`\MolMass`
$\mathrm{g\,mol}^{-1}$

`\normal`
N

`\torr`
torr

By the way: `\mmHg` mmHg already is defined by siunitx.

8. Internal Modules

8.1. The tikz Module

The `tikz` module loads the tikz package [Tan13] and the Ti*k*Z library `calc`.

8.1.1. For Users

The `tikz` module defines a few arrow tips:

`el`
An arrow tip: `\tikz\draw[-el](0,0)--(1,0);`

`left el`
An arrow tip: `\tikz\draw[-left el](0,0)--(1,0);`

`right el`
An arrow tip: `\tikz\draw[-right el](0,0)--(1,0);`

Introduced in version 5.3

The `tikz` module also loads the libraries `calc` and `decorations.pathmorphing`. It uses those libraries for defining a new decoration `wave`.

```
\begin{tikzpicture}
  \draw[decorate,decoration=wave]
  (0,0) -- (2,0) ;
\end{tikzpicture}
```


8.1.2. For Module Writers

The `tikz` module provides some macros for common Ti*k*Z functions. This allows to use expl3's powerful function variants for expansion control.

`\c_chemmacros_other_colon_tl`
A constant tokenlist which contains a colon with category code 12 (other). This is useful since Ti*k*Z sometimes expects an other colon and in an expl3 programming environment `:` has category code 11 (letter).

`\chemmacros_tikz_picture:nn` {⟨*options*⟩} {⟨*code*⟩}
Defined as `\tikzpicture[{#1}] #2 \endtikzpicture`.

`\chemmacros_tikz:nn` {⟨*options*⟩} {⟨*code*⟩}
Defined as `\tikz[{#1}]{#2}`.

`\chemmacros_tikz_draw:n` {⟨*options*⟩}
Defined as `\draw[{#1}]`.

`\chemmacros_tikz_node:n` {⟨*options*⟩}
Defined as `\node[{#1}]`.

`\chemmacros_tikz_shade:n` {⟨*options*⟩}
Defined as `\shade[{#1}]`.

`\chemmacros_tikz_shadedraw:n` {⟨*options*⟩}
Defined as `\shadedraw[{#1}]`.

`\chemmacros_tikz_node_in_draw:n` {⟨*options*⟩}
Defined as `node[{#1}]`.

Table 4: Predefined xfrac text instances.

font family	text	superscript
cmr	2/3	2/3
lmr	2/3	2/3
LinuxLibertineT-TLF	⅔	⅔
LinuxLibertineT-TOsF	⅔	⅔

8.2. The xfrac Module

The xfrac module loads the package xfrac [L3Pb]. For the following explanations it will be helpful if you know about said package and how it works first. This module is a support module that defines the macro

\chemfrac[⟨type⟩]{⟨numerator⟩}{⟨denominator⟩}
⟨type⟩ can either be text or superscript.

This macro calls a certain instance of the xfrac text template, depending on the option ⟨type⟩ and the current font family. If used \chemfrac looks if an instance

chemmacros-frac-\f@family-⟨type⟩

exists. If yes this instance is used, if no the instance chemmacros-frac-default-⟨type⟩ is used. The default instances are the same as the ones for cmr.

The xfrac module defines instances some font families, they are listed and demonstrated in table 4. The superscript type fractions *look* larger than the text types. The reason is that the superscript types are typically used with a smaller font size. Let's take a look at an example where both instances are used:

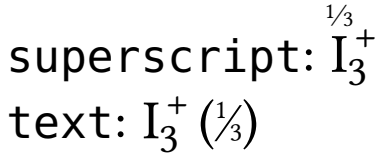
```
\chemsetup[redox]{pos=top}
\code{superscript}:
\ch{"\ox{1/3,I}" {}3+}

\chemsetup[redox]{pos=side}
\code{text}: \ox{1/3,I3+}

\huge
\chemsetup[redox]{pos=top}
\code{superscript}:
\ch{"\ox{1/3,I}" {}3+}

\chemsetup[redox]{pos=side}
\code{text}: \ox{1/3,I3+}
```

superscript: $\overset{1/3}{I_3^+}$
text: I_3^+ (⅓)



If you define instances for other families please feel free to submit them to me (see section A.2 starting on page 62) so they can be added to the xfrac module.

Part IV.
Appendix

A. Own Modules

A.1. How To

If you have additional functionality which you think might be useful as a CHEMMACROS module then you can easily write one yourself. The module must be a file in a path where TEX can find it following a certain naming scheme. The file for a module `foo` *must be named* `chemmacros.module.foo.code.tex`.

`\ChemModule*{`⟨*name*⟩`}{`⟨*description*⟩`}[`⟨*minimal compatibility version*⟩`]`
Register module ⟨*name*⟩. The optional argument ⟨*minimal compatibility version*⟩ ensures that this module is only loaded if the option `compatibility` has a high enough version number. If it is omitted the module can be loaded in each version 5.0 or higher.

The first line in the file then should look similar to this:

```
1 \ChemModule{foo}{2015/07/14 description of foo}
```

This registers module `foo` which means CHEMMACROS will accept this file as a valid module.

Since CHEMMACROS is written using expl3 `\ChemModule` starts an expl3 programming environment. If you don't want that but rather want to write your module using traditional LATEX 2ε methods then use the starred variant:

```
1 \ChemModule*{foo}{2015/07/14 description of foo}
```

In both variants `@` has category code 11 (letter).

Since new modules very likely might rely on code provided first in a certain version of CHEMMACROS you might want to make sure that your module only is loaded when the compatibility mode is high enough to provide the features you want:

```
1 \ChemModule{foo}{2015/10/14 description of foo}[5.2]
```

You should be aware that your module *will not be loaded* with `\usechemmodule{all}`! The pseudo-module `all` contains a manually maintained list of the modules that are loaded by it.

If you decide to write your module `foo` using expl3 and add options you want to be able to set using `\chemsetup[foo]{`⟨*options*⟩`}` please make sure you define them the following way:

```
\keys_define:nn {chemmacros/foo} {
  ...
}
```

Also (especially if you consider submitting the module, see section A.2) please follow the expl3 naming conventions for variables and functions, *i. e.*, use `chemmacros` as expl3 module name:

```
\tl_new:N \l__chemmacros_my_internal_variable_tl
\tl_new:N \l_chemmacros_my_public_variable_tl
\cs_new:Npn \__chemmacros_my_internal_function:n #1 { ... }
\cs_new_protected:Npn \chemmacros_my_public_function:n #1 { ... }
\NewDocumentCommand \publicfunction {m}
  { \chemmacros_my_public_function:n {#1} }
```

You will find more details on the naming conventions in `interface3.pdf` which most likely is available on your system:

```
~ $ texdoc interface3
```

If you haven't read section 6.1 starting on page 24 about the `base` module, yet, please have a look. There some macros for module writers are described. Also other modules define macros for module writers which may be useful.

A.2. Submitting a Module

If you have written a module and feel it might be useful for other users please feel free to contact me and submit the module. I will surely take at look at both functionality and code and if I feel that it adds value to CHEMMACROS I will add it to the package. Requirement for this is that the module is licensed with the LaTeX Project Public License (v1.3 or later) and that I take over maintenance (according to the "maintainer" status of the LPPL).

Please do *not* submit your module via pull request but send me the files directly. In the best case you also have a short piece of documentation.

B. Suggestions, Bug Reports, Support

Support If you need support or help with anything regarding CHEMMACROS please use the usual support forums

- `http://www.golatex.de/` or
- `http://texwelt.de/wissen/` if you speak German,
- `http://www.latex-community.org/forum/` or

- `http://tex.stackexchange.com/` if you speak English

or go the *dedicated support forum*

- `http://www.mychemistry.eu/forums/forum/chemmacros/`

where you can be sure that I will see the question.

Suggestions If you have any suggestions on how CHEMMACROS could be improved, adding missing features *etc.*, please feel free to contact me via `contact@mychemistry.eu`.

Bug reports If you find any bugs, *i. e.*, errors (something not working as described, conflicts with other packages, …) then please go to `https://github.com/cgnieder/chemmacros/issues/` and open a new issue describing the error including a minimal working example.

C. References

[Ars13] Donald Arseneau. relsize. version 4.1, Mar. 29, 2013.
URL: `http://mirror.ctan.org/macros/latex/contrib/relsize`.

[Bra13] Johannes Braams, current maintainer: Javier Bezos.
babel. version 3.9f, May 16, 2013.
URL: `http://mirror.ctan.org/macros/latex/required/babel/`.

[Cha13] François Charette, current maintainer: Arthur Reutenauer.
polyglossia. version 1.33.4, June 27, 2013.
URL: `http://mirror.ctan.org/macros/latex/contrib/polyglossia/`.

[CM04] David Carlisle and Frank Mittelbach. bm. version 1.1c, Feb. 26, 2004.
URL: `http://mirror.ctan.org/macros/latex/required/bm/`.

[Coh+08] E. Richard Cohan et al.
"Quantities, Symbols and Units in Physical Chemistry", IUPAC Green Book.
3rd Edition. 2nd Printing. IUPAC & RSC Publishing, Cambridge, 2008.

[Con+05] Neil G. Connelly et al. *"Nomenclature of Inorganic Chemistry", IUPAC Red Book.*
IUPAC & RSC Publishing, Cambridge, 2005. ISBN: 0-85404-438-8.

[Hen15] Martin Hensel. mhchem. version 4.02, July 23, 2015.
URL: `http://mirror.ctan.org/macros/latex/contrib/mhchem/`.

[Koh15] Markus Kohm. KOMA-Script. version 3.18, July 2, 2015.
URL: `http://mirror.ctan.org/macros/latex/contrib/koma-script/`.

[L3Pa] The LaTeX3 Project Team. l3kernel. version SVN 6210, Oct. 14, 2015.
URL: `http://mirror.ctan.org/macros/latex/contrib/l3kernel/`.

[L3Pb] The LaTeX3 Project Team. l3packages. version SVN 6210, Oct. 14, 2015.
URL: `http://mirror.ctan.org/macros/latex/contrib/l3packages/`.

[Leh15] Philipp Lehman, current maintainer: Joseph Wright.
etoolbox. version 2.2a, Aug. 2, 2015.
URL: `http://mirror.ctan.org/macros/latex/contrib/etoolbox/`.

[MRW13] Lars Madsen, Will Robertson, and Joseph Wright.
mathtools. version 1.13, Feb. 12, 2013.
URL: `http://mirror.ctan.org/macros/latex/contrib/mh/`.

[MSoo] Frank Mittelbach and Rainer Schöpf. amstext. version 2.01, June 29, 2000. URL: http://mirror.ctan.org/macros/latex/required/amstext/.

[Nie13] Clemens Niederberger. translations. version 1.1a, Sept. 30, 2013. URL: http://mirror.ctan.org/macros/latex/contrib/translations/.

[Nie15a] Clemens Niederberger. chemformula. version 4.11, June 30, 2015. URL: http://mirror.ctan.org/macros/latex/contrib/chemformula/.

[Nie15b] Clemens Niederberger. chemgreek. version 1.0a, July 1, 2015. URL: http://mirror.ctan.org/macros/latex/contrib/chemgreek/.

[Nie15c] Clemens Niederberger. chemnum. version 1.1a, May 13, 2015. URL: http://mirror.ctan.org/macros/latex/contrib/chemnum/.

[Nie15d] Clemens Niederberger. elements. version 0.1, June 14, 2015. URL: http://mirror.ctan.org/macros/latex/contrib/elements/.

[Ped04] Bjørn Pedersen. bpchem. version 1.06, Nov. 25, 2004. URL: http://mirror.ctan.org/macros/latex/contrib/bpchem/.

[PPR04] R. Panico, W. H. Powell, and J-C. Richer. *"Nomenclature of Organic Chemistry, Sections A, B, C, D, E, F, and H", IUPAC Blue Book.* draft. Oct. 7, 2004. URL: http://old.iupac.org/reports/provisional/abstract04/BB-prs310305/CompleteDraft.pdf (visited on 07/07/2013).

[Tan13] Till Tantau. TikZ/pgf. version 3.0.0, Dec. 13, 2013. URL: http://mirror.ctan.org/graphics/pgf/.

[Tel15] Christian Tellechea. chemfig. version 1.2c, Nov. 20, 2015. URL: http://mirror.ctan.org/macros/generic/chemfig/.

[Wri13] Joseph Wright. chemstyle. version 2.0m, July 3, 2013. URL: http://mirror.ctan.org/macros/latex/contrib/chemstyle/.

[Wri15] Joseph Wright. siunitx. version 2.6h, July 17, 2015. URL: http://mirror.ctan.org/macros/latex/contrib/siunitx/.

D. Index

N

O

P

Q

R