`metakeys.sty`: A generic framework for extensible Metadata in LaTeX*

Michael Kohlhase
Jacobs University, Bremen
`http://kwarc.info/kohlhase`

April 14, 2015

Abstract

The `metakeys` package is part of the sTeX collection, a version of TeX/LaTeX that allows to markup TeX/LaTeX documents semantically without leaving the document format, essentially turning TeX/LaTeX into a document format for mathematical knowledge management (MKM).

This package supplies the infrastructure for extending sTeX macros with OMDoc metadata. This package is mainly intended for authors of sTeX extension packages.

Contents

*Version v0.9 (last revised 2012/10/26)

1 The User Interface

Many of the sTEX macros and environments take an optional first argument which uses key/value pairs to specify metadata relations of the marked up objects. The `metakeys` package supplies the infrastructure managing these key/value pairs. It also forms the basis for the `rdfmeta` package which allows to use these for flexible, user-extensible metadata relations (see [**Kohlhase:rdfmeta:ctan**] for details).

1.1 Package Options

`showmeta` The `metakeys` package takes a single option: `showmeta`. If this is set, then the metadata keys defined by the `\addmetakey` are shown (see **??**)

1.2 Adding Metadata Keys to Commands

Key/value pairs in sTEX are organized in **key groups**: every sTEX macro and environment that takes a key/value argument has an associated key group, and only keys that are registered in this group can be utilized. The `metakeys` package `\addmetakey` supplies the `\addmetakey` macro to add a new key to a key group: If ⟨*group*⟩ is the name of a key group ⟨*key*⟩ is a metadata keyword name, then

`\addmetakey[`⟨*default*⟩`]{`⟨*group*⟩`}{`⟨*key*⟩`}[`⟨*dval*⟩`]`

registers ⟨*key*⟩ in the metadata group ⟨*group*⟩, with an optional values ⟨*default*⟩ and ⟨*dval*⟩ for ⟨*key*⟩. ⟨*default*⟩ is the default value for ⟨*key*⟩, if it is not specified, and ⟨*dval*⟩ is the value ⟨*key*⟩ gets, if ⟨*key*⟩ is given without specifying a value. These two defaults are often used as

`\addmetakey[false]{`⟨*group*⟩`}{`⟨*key*⟩`}[true]`

Then, the value of ⟨*key*⟩ is `false` if ⟨*key*⟩ is not given and `true`, if ⟨*key*⟩ is specified without value. This is often the best way if we want to use ⟨*key*⟩ as an indicator to have a feature of name ⟨*key*⟩ (we can test that with `\ifx\`⟨*group*⟩`@`⟨*key*⟩`\@true`, if we prepared the macro `\def\@true{true}` earlier).

The keys registered for a metadata group can be used for defining macros `\metasetkeys` with a key/value arguments via the `\metasetkeys` macro, see for instance the the definition in Figure **??**. This macro is used exactly like the `\setkeys` macro from the `keyval` package [**Carlisle:tkp99**], but integrates custom initialization and draft display functionality. This usage is mostly for package designers. There is another: If a macro or environment cannot be extended by an optional argument, e.g. because anther package already does so (e.g. the `document` environment is extended – by redefining it – by various packages, which causes problems), the `\metasetkeys` macro can be used directly.

`\addmetalistkey` The `\addmetalistkey` macro is a variant of `\addmetakey` that adds a list-valued metadata key. The `\addmetalistkey{foo}{val}` in Figure **??** would allows to use multiple occurrences of the `val` keys in the metadata argument of `\foo`, the values of the `val` keys are collected as a comma-separated list in the token

register `\foo@vals`. Note that the `val` key can also deal with comma-separated lists for convenience.

With these definitions in a used package[1] an invocation of

```
\foo[type=bar,id=f4711,val=4,val=7,val={1,1}]
```

id:f4711 type:bar

is formatted to

> I have seen a *foo* of type `bar` with identifier `f4711` and values 4, and 7, and 1, and 1!

```
\addmetakey{foo}{id}
\addmetakey{foo}{type}
\addmetakey[yes]{foo}{visible}
\addmetalistkey{foo}{val}
\def\@yes{yes}
\newcommand\foo[1][]{\metasetkeys{foo}{#1}
\ifx\foo@visible\@yes % testing for visibility
I have seen a \emph{foo} of type \texttt{\foo@type} with identifier
\texttt\foo@id and values \texttt\foo@vals.
\let\@join=\relax\def\@thejoin{, and }
\@for\@I:=\foo@vals\do{\@join\@I\let\@join=\@thejoin}!
\fi}
```

Example 1: Defining a macro with metadata

1.3 Showing Metadata Keys/Values

If the `showmeta` package option is set, the `metakeys` package sets an internal switch that shows the values of all keys specified with the `\addmetakey` macro. The default behavior is to write the key/value pairs into the margin as ⟨*key*⟩:⟨*value*⟩. Package designers can customize this behavior by redefining the `\metakeys@show@key` and `\metakeys@show@keys` macro.

`\metakeys@show@key` `\metakeys@show@keys`

`\metakeys@show@key{`⟨*key*⟩`}{`⟨*value*⟩`}` shows the a single key value pair, and `\metakeys@show@keys{`⟨*group*⟩`}{`⟨*keys*⟩`}` shows the a list of keys metadata, by default we disregard the ⟨*group*⟩ and show ⟨*keys*⟩ in a marginpar.

`\addmetakey*`

For keys that should not be shown in this manner, the `\addmetakey` macro has a variant `\addmetakey*`. Its behavior is exactly the same, only that it keeps the key from being shown by the `showmeta` option.

`\hidemetakeys` `\showmetakeys`

Note that setting the `showmeta` option will enable metadata presentation on the whole document. But sometimes we want to disable that, e.g. inside figures, where `\marginpar` is not allowed. Therefore the `metakeys` package provides the `\hidemetakeys` macro that reverses this. The `\showmetakeys` macro re-enables metadata presentation.

[1] Recall that the `@` character is only allowed in packages, where comma-separated lists can be iterated over e.g. by the `\@for` macro.

2 Limitations

In this section we document known limitations. If you want to help alleviate them, please feel free to contact the package author. Some of them are currently discussed in the sTEX TRAC [**sTeX:online**].

1. none reported yet

3 The Implementation

The `metakeys` package generates two files: the LaTeX package (all the code between ⟨*package⟩ and ⟨/package⟩) and the LaTeXML bindings (between ⟨*ltxml⟩ and ⟨/ltxml⟩). We keep the corresponding code fragments together, since the documentation applies to both of them and to prevent them from getting out of sync.

3.1 Package Options

We declare some switches which will modify the behavior according to the package options. Generally, an option `xxx` will just set the appropriate switches to true (otherwise they stay false). First we have the general options

```
⟨*package⟩
\newif\ifmetakeys@showmeta\metakeys@showmetafalse
\DeclareOption{showmeta}{\metakeys@showmetatrue}
```

Finally, we need to declare the end of the option declaration section to LaTeX.

```
\ProcessOptions
```

We build on the `keyval` package which we first need to load. For LaTeXML, we also initialize the package inclusions.

```
\RequirePackage{keyval}[1997/11/10]
⟨/package⟩
⟨*ltxml⟩
# -*- CPERL -*-
package LaTeXML::Package::Pool;
use strict;
use LaTeXML::Package;
DeclareOption('showmeta', '');
ProcessOptions();
RequirePackage('keyval');
⟨/ltxml⟩
```

3.2 Adding Metadata Keys

`\addmetakey` The `\addmetakey` macro looks at the next character and invokes helper macros accordingly.

```
⟨*package⟩
\newcommand\addmetakey{\@ifstar\addmetakey@star\addmetakey@nostar}
⟨/package⟩
```

`\addmetakey@star` `\addmetakey@star` takes care of the starred form of `\addmetakey`. An invocation of `\addmetakey@star{`⟨*default*⟩`}{`⟨*group*⟩`}{`⟨*key*⟩`}` macro first extends the `\metakeys@clear@`⟨*group*⟩`@keys` macro then defines the key ⟨*key*⟩ with the `\define@key` macro from the `keyval` package. This stores the key value given in the local macro `\`⟨*group*⟩`@`⟨*key*⟩.

```
⟨*package⟩
\newcommand\addmetakey@star[3][]%
{\@ifnextchar[{\addmetakey@star@aux[#1]{#2}{#3}}{\addmetakey@star@aux[#1]{#2}{#3}[]}}
\def\addmetakey@star@aux[#1]#2#3[#4]{\metakeys@ext@clear@keys{#2}{#3}{#1}%
\metakeys@initialize@showkeys{#2}%
\define@key{#2}{#3}[#4]{\expandafter\xdef\csname #2@#3\endcsname{##1}}}
```

`\addmetakey@nostar` `\addmetakey@nostar` takes care of the starred form of `\addmetakey` by first extending the `\metakeys@`⟨*group*⟩`@showkeys` macro which contains those keys that should be shown and then calling `\addmetakey@star`.

```
\newcommand\addmetakey@nostar[3][]%
{\metakeys@ext@showkeys{#2}{#3}\addmetakey@star[#1]{#2}{#3}}
⟨/package⟩
```

`\metasetkeys` The `\metasetkeys{`⟨*group*⟩`}` clears/presets the key of ⟨*group*⟩ via `\clear@`⟨*group*⟩`@keys`, (if the `showmeta` option is set) shows them, and then sets the keys via `keyvals` `\setkeys` command.

```
⟨*package⟩
\newcommand\metasetkeys[2]{\@nameuse{clear@#1@keys}\setkeys{#1}{#2}%
\ifmetakeys@showmeta%
\edef\@@keys{\@nameuse{#1@showkeys}}%
\metakeys@show@keys{#1}{\@for\@I:=\@@keys\do{\metakeys@show@keyval{#1}{\@I}}}%
\fi}
⟨/package⟩
```

`\metakeys@ext@clear@keys` `\metakeys@ext@clear@keys{`⟨*group*⟩`}{`⟨*key*⟩`}{`⟨*default*⟩`}` extends (or sets up if this is the first `\addmetakey` for ⟨*group*⟩) the `\clear@`⟨*group*⟩`@keys` macro to set the default value ⟨*default*⟩ for ⟨*key*⟩. The `\clear@`⟨*group*⟩`@keys` macro is used in the generic `\metasetkeys` macro below. The variant `\@metakeys@ext@clear@keys` is provided for use in the `sref` package.

```
⟨*package⟩
\newcommand\metakeys@ext@clear@keys[3]{\@metakeys@ext@clear@keys{#1}{#1@#2}{#3}}
\newcommand\@metakeys@ext@clear@keys[3]{\@ifundefined{clear@#1@keys}%
{\expandafter\gdef\csname clear@#1@keys\endcsname%
{\expandafter\gdef\csname #2\endcsname{#3}}}%
{\expandafter\g@addto@macro\csname clear@#1@keys\endcsname%
{\expandafter\gdef\csname #2\endcsname{#3}}}}
⟨/package⟩
```

`\addmetalistkey`

```
⟨*package⟩
\newcommand\addmetalistkey{\@ifstar\addmetalistkey@star\addmetalistkey@nostar}
\newcommand\addmetalistkey@star[3][]{\metakeys@ext@clear@keys{#2}{#3}{#1}%
```

```
\metakeys@initialize@showkeys{#2}%
\expandafter\gdef\csname #2@#3s\endcsname{}
\define@key{#2}{#3}[#1]{%
\expandafter\ifx\csname #2@#3s\endcsname\@empty\expandafter\gdef\csname #2@#3s\endcsname{##1}%
\else\expandafter\xdef\csname #2@#3s\endcsname{\csname #2@#3s\endcsname,##1}%
\fi}}
\newcommand\addmetalistkey@nostar[3][]%
{\metakeys@ext@showkeys{#2}{#3}\addmetalistkey@star[#1]{#2}{#3}}
⟨/package⟩
```

3.3 Showing Metadata Keys/Values

`\metakeys@initialize@showkeys` `\metakeys@initialize@showkeys{`⟨*group*⟩`}` sets up the `\`⟨*group*⟩`@showkeys` macro which is is used to store the keys to be shown of the metadata in in the generic `\setmetakeys` macro below.

```
⟨*package⟩
\newcommand\metakeys@initialize@showkeys[1]%
{\@ifundefined{#1@showkeys}{\expandafter\def\csname #1@showkeys\endcsname{}}{}}%
```

`\metakeys@ext@showkeys` `\metakeys@ext@showkeys{`⟨*group*⟩`}{`⟨*key*⟩`}` extends (or sets up) the `\`⟨*group*⟩`@showkeys` macro which is is used to store the keys to be shown of the metadata in in the generic `\setmetakeys` macro below.

```
\newcommand\metakeys@ext@showkeys[2]{\@ifundefined{#1@showkeys}%
{\expandafter\def\csname #1@showkeys\endcsname{#2}}%
{\expandafter\edef\csname #1@showkeys\endcsname{\csname #1@showkeys\endcsname,#2}}}
```

`\metakeys@show@key` `\metakeys@show@key{`⟨*key*⟩`}{`⟨*value*⟩`}` shows the a single key value pair, as a default we just write ⟨*key*⟩`:`⟨*value*⟩.

```
\newcommand\@metakeys@show@key[2]{\metakeys@show@key{#2}{#1}}
\newcommand\metakeys@show@key[2]{\edef\@test{#2}\ifx\@test\@empty\else #1:#2\quad\fi}
```

`\metakeys@show@keys` `\metakeys@show@keys{`⟨*group*⟩`}{`⟨*keys*⟩`}` shows the metadata, by default we disregard the ⟨*group*⟩ and show ⟨*keys*⟩ in a marginpar.

```
\newcommand\metakeys@show@keys[2]{\marginpar{{\scriptsize #2}}}
```

`\metakeys@show@keyval` `\metakeys@show@keyval{`⟨*group*⟩`}\meta{key}` shows the key/value pair of a given key ⟨*key*⟩.

```
\newcommand\metakeys@show@keyval[2]%
{\expandafter\@metakeys@show@key\csname #1@#2\endcsname{#2}}
⟨/package⟩
```

`\showmetakeys`

```
⟨*package⟩
\newcommand\showmetakeys{\metakeys@showmetatrue}
⟨/package⟩
⟨*ltxml⟩
DefConstructor('\showmetakeys','');
⟨/ltxml⟩
```

\hidemetakeys

```
⟨*package⟩
\newcommand\hidemetakeys{\metakeys@showmetafalse}
⟨/package⟩
⟨*ltxml⟩
DefConstructor('\hidemetakeys','');
⟨/ltxml⟩
```

3.4 Using better defaults than empty

\addmetakeynew \addmetakeynew is an experimental version of \addmetakey which gives \omd@unspecified as an optional argument, so that it is used as the default value here and then test for it in \omfidus. But unfortunately, this does not work yet.

```
⟨*package⟩
\newcommand\addmetakeynew[3][]{\metakeys@ext@clear@keys{#2}{#3}{#1}%
\define@key{#2}{#3}{\expandafter\gdef\csname #2@#3\endcsname{##1}}}
```

EdN:1 \metakeys@unspecified An internal macro for unspecified values. It is used to initialize keys.[1]

```
\newcommand\metakeys@unspecified{an metakeys-defined key left unspecified}
```

\metakeysifus This just tests for equality of the first arg with \metakeys@unspecified

```
\newcommand\metakeysifus[4]{\message{testing #1@#2=\csname#1@#2\endcsname}%
\expandafter\ifx\csname #1@#2\endcsname\metakeys@unspecified{#3}\else{#4}\fi}
⟨/package⟩
```

3.5 Finale

Finally, we need to terminate the file with a success mark for perl.

```
⟨ltxml⟩1;
```

[1] EdNote: MK: we could probably embed an package error or warning in here