

# metakeys.sty: A generic framework for extensible Metadata in L<sup>A</sup>T<sub>E</sub>X\*

Michael Kohlhasse  
Jacobs University, Bremen  
<http://kwarc.info/kohlhasse>

November 7, 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

<b>1</b>	<b>The User Interface</b>	<b>2</b>
1.1	Package Options . . . . .	2
1.2	Adding Metadata Keys to Commands . . . . .	2
1.3	Showing Metadata Keys/Values . . . . .	3
<b>2</b>	<b>Limitations</b>	<b>4</b>
<b>3</b>	<b>The Implementation</b>	<b>4</b>
3.1	Package Options . . . . .	4
3.2	Adding Metadata Keys . . . . .	5
3.3	Showing Metadata Keys/Values . . . . .	7
3.4	Using better defaults than empty . . . . .	8
3.5	Finale . . . . .	8

---

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

# 1 The User Interface

Many of the  $\text{\TeX}$  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 **rdmeta** package which allows to use these for flexible, user-extensible metadata relations (see [Koh15] 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 1.3)

## 1.2 Adding Metadata Keys to Commands

Key/value pairs in  $\text{\TeX}$  are organized in **key groups**: every  $\text{\TeX}$  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 supplies the `\addmetakey` macro to add a new key to a key group: If  $\langle group \rangle$  is the name of a key group  $\langle key \rangle$  is a metadata keyword name, then

`\addmetakey[\langle default \rangle]{\langle group \rangle}{\langle key \rangle}[\langle dval \rangle]`

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

`\addmetakey[false]{\langle group \rangle}{\langle key \rangle}[true]`

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

The keys registered for a metadata group can be used for defining macros with a key/value arguments via the `\metasetkeys` macro, see for instance the definition in Figure 1. This macro is used exactly like the `\setkeys` macro from the **keyval** package [Car99], 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 another 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 1 would allow 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<sup>1</sup> an invocation of

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

is formatted to

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

id:f4711 type:bar

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

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.

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` `\hidemetakeys` macro that reverses this. The `\showmetakeys` macro re-enables metadata presentation.

<sup>1</sup>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` GitHub repository [sTeX].

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.

The general preamble for `LATEXML`:

```
1 <*ltxml>
2 # -*- CPERL -*-
3 package LaTeXML::Package::Pool;
4 use strict;
5 use LaTeXML::Package;
6 </ltxml>
```

### 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

```
7 <*package>
8 \newif\ifmetakeys@showmeta\metakeys@showmetafalse
9 \DeclareOption{showmeta}{\metakeys@showmetatrue}
10 \DeclareOption*{}
11 \ProcessOptions
12 </package>
13 <*ltxml>
14 DeclareOption('showmeta', '');
15 DeclareOption(undef, '');
16 ProcessOptions();
17 </ltxml>
```

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

```
18 <*package>
19 \RequirePackage{keyval}[1997/11/10]
20 \RequirePackage{etoolbox}
21 </package>
22 <*ltxml>
23 RequirePackage('keyval');
24 </ltxml>
```

## 3.2 Adding Metadata Keys

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

```
25 <*package>
26 \newrobustcmd\addmetakey{\@ifstar\addmetakey@star\addmetakey@nostar}%
27 </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>`.

```
28 <*package>
29 \newrobustcmd\addmetakey@star[3][]{%
30   \@ifnextchar[%
31     \addmetakey@star@aux[#1]{#2}{#3}%
32   }{%
33     \addmetakey@star@aux[#1]{#2}{#3}[]%
34   }%
35 }%
36 \def\addmetakey@star@aux[#1]#2#3[#4]{%
37   \metakeys@ext@clear@keys{#2}{#3}{#1}%
38   \metakeys@initialize@showkeys{#2}%
39   \define@key{#2}{#3}{#4}{%
40     \csxdef{#2@#3}{##1}%
41   }%
42 }%
```

`\addmetakey@nostar` `\addmetakey@nostar` takes care of the no-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`.

```
43 \newrobustcmd\addmetakey@nostar[3][]{%
44   \metakeys@ext@showkeys{#2}{#3}%
45   \addmetakey@star[#1]{#2}{#3}%
46 }%
47 </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.

```
48 <*package>
49 \newrobustcmd\metasetkeys[2]{%
50   \@nameuse{clear@#1@keys}%
51   \setkeys{#1}{#2}%
52   \ifmetakeys@showmeta%
53     \edef\@keys{\@nameuse{#1@showkeys}}%
54     \metakeys@show@keys{#1}{%
55       \for\@I:=\@keys\do{%
```

```

56      \metakeys@show@keyval{#1}{\@I}%
57      }%
58      }%
59      \fi%
60 }%
61 </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.

```

62 <*package>
63 \newrobustcmd\metakeys@ext@clear@keys[3]{%
64   \@metakeys@ext@clear@keys{#1}{#1@#2}{#3}%
65 }%
66 \newrobustcmd\@metakeys@ext@clear@keys[3]{%
67   \@ifundefined{clear@#1@keys}{%
68     \csgdef{clear@#1@keys}{\csgdef{#2}{#3}}%
69   }%
70   {\expandafter\gappto\csname clear@#1@keys\endcsname{\csgdef{#2}{#3}}}%
71 }%
72 </package>

```

`\addmetalistkey`

```

73 <*package>
74 \newrobustcmd\addmetalistkey{%
75   \@ifstar\addmetalistkey@star\addmetalistkey@nostar%
76 }%
77 \newrobustcmd\addmetalistkey@star[3][]{%
78   \metakeys@ext@clear@keys{#2}{#3}{#1}%
79   \metakeys@initialize@showkeys{#2}%
80   \csgdef{#2@#3s}{}%
81   \define@key{#2}{#3}[#1]{%
82     \ifcsemt{#2@#3s}{%
83       \csgdef{#2@#3s}{##1}%
84     }{%
85       \csxdef{#2@#3s}{\csuse{#2@#3s},##1}%
86     }%
87   }%
88 }%
89 \newrobustcmd\addmetalistkey@nostar[3][]{%
90   \metakeys@ext@showkeys{#2}{#3}%
91   \addmetalistkey@star[#1]{#2}{#3}%
92 }%
93 </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.
94 <*package>
95 \newrobustcmd\metakeys@initialize@showkeys[1]{%
96   \ifundefined{#1@showkeys}{%
97     \csdef{#1@showkeys}{}%
98   }{}%
99 }%

\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.
100 \newrobustcmd\metakeys@ext@showkeys[2]{%
101   \ifundefined{#1@showkeys}{%
102     \csdef{#1@showkeys}{#2}%
103   }{%
104     \csedef{#1@showkeys}{\csuse{#1@showkeys},#2}%
105   }%
106 }%

\metakeys@show@key \metakeys@show@key{<key>}{<value>}} shows the a single key value pair, as a de-
fault we just write <key>:<value>.
107 \newrobustcmd\metakeys@show@key[2]{\metakeys@show@key{#2}{#1}}%
108 \newrobustcmd\metakeys@show@key[2]{%
109   \edef\@test{#2}%
110   \ifx\@test\empty\else #1:#2\quad\fi%
111 }%

\metakeys@show@keys \metakeys@show@keys{<group>}{<keys>}} shows the metadata, by default we dis-
regard the <group> and show <keys> in a marginpar.
112 \newrobustcmd\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>.
113 \newrobustcmd\metakeys@show@keyval[2]{%
114   \expandafter\@metakeys@show@key\csname #1#2\endcsname{#2}%
115 }%
116 </package>

\showmetakeys
117 <*package>
118 \newrobustcmd\showmetakeys{\metakeys@showmetatrue}%
119 </package>
120 <*ltxml>
121 DefConstructor(' \showmetakeys', '');
122 </ltxml>

```

`\hidemetakeys`

```
123 <*package>
124 \newrobustcmd\hidemetakeys{\metakeys@showmetafalse}%
125 </package>
126 <*!xml>
127 DefConstructor('\'hidemetakeys',');
128 </!xml>
```

### 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.

```
129 <*package>
130 \newrobustcmd\addmetakeynew[3] [] {%
131   \metakeys@ext@clear@keys{#2}{#3}{#1}%
132   \define@key{#2}{#3}{%
133     \csgdef{#2@#3}{#1}%
134   }%
135 }
```

EdN:1\metakeys@unspecified An internal macro for unspecified values. It is used to initialize keys.<sup>1</sup>

```
136 \newrobustcmd\metakeys@unspecified{an metakeys-defined key left unspecified}%
```

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

```
137 \newrobustcmd\metakeysifus[4] {%
138   \message{testing #1@#2=\csname#1@#2\endcsname}%
139   \expandafter\ifx\csname #1@#2\endcsname\metakeys@unspecified{#3}\else{#4}\fi%
140 }%
141 </package>
```

### 3.5 Finale

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

```
142 <!xml>1;
```

---

<sup>1</sup>EdNOTE: MK: we could probably embed an package error or warning in here



## References

- [Car99] David Carlisle. *The keyval package*. Part of the T<sub>E</sub>X distribution. The Comprehensive T<sub>E</sub>X Archive Network. 1999. URL: <https://www.tug.org/texlive/devsrc/Master/texmf-dist/doc/latex/graphics/keyval.pdf>.
- [Koh15] Michael Kohlhase. *RDFa Metadata in L<sup>A</sup>T<sub>E</sub>X*. Tech. rep. Comprehensive T<sub>E</sub>X Archive Network (CTAN), 2015. URL: <http://www.ctan.org/tex-archive/macros/latex/contrib/stex/rdfmeta/rdfmeta.pdf>.
- [sTeX] *KWARC/sTeX*. URL: <https://svn.kwarc.info/repos/stex> (visited on 05/15/2015).