RDFa Metadata in LaTeX*

Michael Kohlhase
FAU Erlangen-Nürnberg
http://kwarc.info/kohlhase

November 25, 2018

Abstract

The `rdfmeta` package allows mark up Ontology-based Metadata in LaTeX documents that can be harvested by automated tools or exported to PDF.

Contents

Experimental! do not use!

*Version v0.2 (last revised 2016/04/07)

1 Introduction

The `rdfmeta` package allows mark up extensible metadata in sTeX documents, so that that it can be harvested by automated tools or exported to PDF. It is also intended to support the new metadata infrastructure for the OMDoc format [**Kohlhase:OMDoc1.2**] introduced in OMDoc1.3 [**Kohlhase:OMDoc1.3**] (see [**LK:MathOntoAuthDoc09**] for the relevant ideas and and [**KohKohLan:ssffld10:biblatex**] for an application).

Metadata are annotated as key value pairs in the semantic environments provided by sTeX. In most markup formats, the metadata vocabularies are fixed by the language designer. In sTeX, the `rdfmeta` package allows the user to extend the metadata vocabulary.

```
 \importmodule[../ontologies/cert]{certification}
 ...
 \section[id=userreq,hasState=$\statedocrd{\tuev}$]{User Requirements}
 ...
```

```
<imports from="../ontologies/cert.omdoc#certification"/>
...
<omgroup xml:id="userreq">
  <metadata>
    <link rel="../ontologies/cert.omdoc#certification:hasState">
      <dc:title>User Requirements</dc:title>
      <resource rel="../ontologies/cert.omdoc#certification/statedocrd"
                resource="../ontologies/cert.omdoc#certification/tuev"/>
    </link>
  </metadata>
...
</omgroup>
```

Example 1: Metadata for Certification

Take, for instance, the case where we want to use metadata for the certification status of document fragments. In Figure **??** we use the `hasState` key to say that a section has been approved by the TÜV, a specific certification agency. There are two concerns here. First, the `hasState` key has to be introduced and given a meaning, and same for the (complex) value `\statedocrd{\tuev}`. This meaning is given in the `certifiation` ontology which we imported via the `\importmodule` command. The ontology can be marked up in sTeX (see Figure **??**), with the exception that we use the `\keydef` macro for the definition of the `hasState` relation so that it also defines the key. For the details of this see the next section.

2 User Interface

We now document the specifics of the environments and macros provided by the `rdfmeta` package from a user perspective.

2.1 Package Options

showmeta

The `rdfmeta` package takes the option: `showmeta`. If this is set, then the metadata keys are shown (see [**Kohlhase:metakeys:ctan**] for details and customization options).

sectioning

The remaining options can be used to specify metadata upgrades of standard keys. The `sectioning` option upgrades the `\part`, `\chapter`, `\section`, `\subsection`, `\subsubsection`, `\paragraph` macros (and of course their starred variants).

2.2 Extending Macros and Environments by Metadata Keys

\keydef

The main user-visible feature of the `rdfmeta` package is the `keydef` macro. It takes two arguments, a "key group identifier" and a key name. In a nutshell, every STEX command that takes metadata keys comes with a "key group identifier" that identifies the set of admissible keys; see [**Kohlhase:metakeys:ctan**] for details on this concept. Figure **??** gives an overview over the key groups and their identifiers in STEX.

Semantically, `\keydef{`⟨*keygroup*⟩`}{`⟨*key*⟩`}` defines a symbol just like the `\symdef` macro from the `modules` package [**KohAmb:smmssl:ctan**]. But it also extends the syntax of STEX itself: it adds a key ⟨*key*⟩ to ⟨*keygroup*⟩, which allows to state the corresponding metadata as a key/value pair in the STEX macro or environment. Following the ideas from [**LK:MathOntoAuthDoc09**], the metadata is transformed to RDFa metadata [**w3c:WD-rdfa-core-20100803**] in OMDoc, where the identifiers of relations are exactly the symbols introduced by the corresponding `\keydef`.

In our example in Figure **??** we have defined a key `hasstate` in the `omtext` key group[1] and a symbol `hasstate` via `\addkey{omtext}{hasstate}`. Furthermore, we have defined the meaning of the relation expressed by the `hasstate` symbol informally and specified some possible objects for the relation (that could of course have been done in other modules as well). We have made use of this metadata ontology and the new key `hasState` in the example in Figure **??**.

2.3 Redefinitions of Common LATEX Macros and Environments

The `rdfmeta` package redefines common LATEX commands (e. g. the sectioning macros) so that they include optional KeyVal arguments that can be extended by `\keydef` commands. With this extension, we can add RDFa metadata to any existing LATEX document and generate linked data (XHTML+RDFa documents) via the LATEXML translator.

[1] For the `\omtext` environment and key group see [**Kohlhase:smmtf:ctan**]

Key Group Identifier	Macros	Package/Class
dcm@person	DCMPerson	dcm.sty
dcm@institution	DCMInstitution	dcm.sty
dcm@sect	section	dcm.sty
assig	assignment	hwexam.cls
inclassig	includeassignment	hwexam.cls
quizheading	quizheading	hwexam.cls
testheading	quizheading	hwexam.cls
module	module	modules.sty
termdef	termdef	modules.sty
view	view	modules.sty
omgroup	omgroup	omdoc.sty
ignore	ignore	omdoc.sty
omtext	omtext, definition, axiom, assertion, example, inlinedef	omtext.sty, statements.sty
phrase	phrase	omtext.sty
problem	problem	problem.sty
inclprob	includeproblem	problem.sty
req	requirement	reqdoc.sty
spf	sproof, spfcases, spfcase, spfstep, spfcomment	sproof.sty
termref	termref	statements.sty
symboldec	symboldec	statements.sty

Figure 1: Key Group Identifiers in STEX

```
\documentclass{omdoc}
\usepackage{stex,rdfmeta,amstext}
\begin{document}
\begin{module}[id=certification]
% \metalanguage[../owl2onto/owl2]{OWL2}
  \keydef{omtext}{hasState}
  \keydef{omgroup}{hasState}
  \symdef{hasState}{\text{hasState}}
  \symdef{statedocrd}[1]{rd. #1}
  \symdef{tuev}{\text{T\"UV}}
  \begin{omgroup}[id=foo,hasState=test]{Definitions}
  \begin{definition}[for=hasState]
    A document \defii{has}{state} $x$, iff the project manager decrees it so.
  \end{definition}
  \begin{definition}[for=statedocrd,hasState=$\statedocrd\tuev$]
    A document has state \defi[name=statedocrd]{rd. $x$}, iff it has been submitted to $x$ for
    certification.
  \end{definition}
  \begin{definition}[for=tuev,hasState=$\statedocrd\tuev$]
    The $\tuev$ (Technischer \"Uberwachungs Verein) is a national
    certification agency in Germany.
  \end{definition}
\end{omgroup}
\end{module}
\end{document}

%%% Local Variables:
%%% mode: latex
%%% TeX-master: t
%%% End:
```

Example 2: A simple Ontology on Certification

2.4 Extending Packages with `rdfmeta`

The `rdfameta` package also exposes its internal infrastructure for extending the redefinitions. Note that the upgrade macros can only be used in LaTeX packages, as the macro names contain `@`. Consequently, this section is only addressed at package developers who want to extend existing (i.e. not written by them) packages with flexible metadata functionality.

`\rdfmeta@upgrade` `\rdfmeta@upgrade` is the basic upgrade macro. It takes an optional keyval argument an a command sequence $\langle cseq \rangle$ as a proper argument and (if that is defined), redefines $\backslash\langle cseq \rangle$ to take a keyval argument. There is a variant

`\rdfmeta@upgrade*` `\rdfmeta@upgrade*` that has to be used to upgrade macros that have a starred form (e.g. `\section` and friends). Note that `\rdfmeta@upgrade*` upgrades both forms (e.g. `\section` and `\section*`).

`\rdfmeta@upgrade` uses four keys to specify the behavior in the case the the macro to be upgraded already has an optional argument. For concreteness, we introduce them using the `\section` macro from standard LaTeX as an example. `\section` has an optional argument for the "short title", which

`optarg` will appear in the table of contents. The `optarg` key can be used to specify a key for the existing optional argument. Thus, after upgrading it via

`\rdfmeta@upgrade*[optarg=short]{section}`, we can use the updated form `\section[short=`⟨*toctitle*⟩`]{`⟨*title*⟩`}` instead of the old `\section[`⟨*toctitle*⟩`]{`⟨*title*⟩`}`. Actually, this still has a problem: the `\section*` would also be given the `short` key and would be passed an optional argument (which it does not accept). To remedy this we can set the `optargstar` key to `no`. In summary, the correct upgrade command for `\section` and `\section*` would be

optargstar

```
\rdfmeta@upgrade*[optarg=short,optargstar=no]{section}
```

The `\rdfmeta@upgrade*` macro also initializes a metadata key-group (a named set of keys and their handlers; see [**Kohlhase:metakeys:svn**] for details) for the section macro with an `id` key for identification (see [**Kohlhase:sref***] for details). Often, the name of the key-group is the same as the command sequence, so we take this as the default, if we want to specify a different metadata key-group name, we can do so with the `keygroup` key in `\rdfmeta@upgrade*`.

keygroup

idlabel

If `idlabel` is set to ⟨*prefix*⟩, then the LaTeX label is set to the value ⟨*prefix*⟩`.`⟨*id*⟩, where ⟨*id*⟩ is the value given in the RDFa `id` key. This allows to use the normal LaTeX referencing mechanism in addition to the semantic referencing mechanism provided by the `sref` package [**Kohlhase:sref:ctan**].

2.5 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:github:on**].

1. Currently the coverage of the redefinitions of standard commands in the `rdfmeta` package is minimal; we will extend this in the future.
2. The `\rdfmeta@upgrade` macro only works with single arguments, this should be easy to fix with `\case` for the argument string.
3. I am not sure `\rdfmeta@upgrade` works with environments.
4. it would be convenient, if we had a macro `\keydefs`, which takes a list of keygroups, so that we can define keys in multiple groups in one go, e.g. `\keydefs{omtext,omgroup}{hasState}` in Figure **??**. But the obvious "solution"

   ```
   \newcommand\keydefs[2]{\@for\@I:=#1\do{\keydef{#1}{#2}}}
   ```

   does not work for me.

3 The Implementation

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).[1]

EdN:1

```
1 ⟨*package⟩
2 \newif\if@rdfmeta@sectioning\@rdfmeta@sectioningfalse
3 \DeclareOption{sectioning}{\@rdfmeta@sectioningtrue}
4 \DeclareOption*{\PassOptionsToPackage{\CurrentOption}{sref}
5                                 \PassOptionsToPackage{\CurrentOption}{modules}}
6 \ProcessOptions
```

The first measure is to ensure that the right packages are loaded. From the from STEX collection, we need the `sref` package (see [**Kohlhase:sref:ctan**]) for handling keys, the `modules` package for exporting the `\keydef` (see [**KohAmb:smmssl:ctan**]).

```
7 \RequirePackage{sref}
8 \RequirePackage{modules}
```

`\rdfmeta@loaded` and we define a macro `rdfmeta@loaded` just for the purpose of determining whether the `rdfmeta` package is loaded.

`\rdfmeta@loaded`

```
9 \newcommand\rdfmeta@loaded{yes}
```

`\rdfmeta@sectioning` And another macro `rdfmeta@sectioning` to determine wether the sectioning macros have been redefined.

`\rdfmeta@loaded`

```
10 \if@rdfmeta@sectioning\newcommand\rdfmeta@sectioning{yes}\fi
```

3.2 Key Definitions

`\keydef` The `\keydef` macro is rather simple, we just add a key to the respective environment and extend the export token register for the current module by an `\addmetakey` instruction.

```
11 \newcommand\keydef[2]{\addmetakey{#1}{#2}%
12 \expandafter\g@addto@macro\this@module{\addmetakey{#1}{#2}}}
```

`\listkeydef` The `\listkeydef` macro is analogous, but uses `\addmetalistkey` instead. instruction.

```
13 \newcommand\listkeydef[2]{\addmetalistkey{#1}{#2}%
14 \expandafter\g@addto@macro\this@module{\addmetalistkey{#1}{#2}}}
```

[1] EdNote: need an implementation for LaTeXML

3.3 RDFa upgrade Facilities

We first define the keys for the `\rdfmeta@upgrade` macro.

```
\def\@yes@{yes}
\addmetakey*{upgrade}{idlabel}
\addmetakey*{upgrade}{optarg}
\addmetakey*[yes]{upgrade}{optargstar}
\addmetakey*{upgrade}{keygroup}
```

`\rdfmeta@upgrade` This upgrade macro gives extended functionality according to the optional keys. The top-level invocation just differentiates on whether a star is following:

```
\def\rdfmeta@upgrade{\@ifstar\rdfmeta@upgrade@star\rdfmeta@upgrade@nostar}
```

Both cases are almost the same, they only differ in the third line where they call `\rdfmeta@upgrade@base` or `\rdfmeta@upgrade@base@star` defined above. In particular, both take the arguments originally intended for `\rdfmeta@upgrade`.

```
\newcommand\rdfmeta@upgrade@nostar[2][]{\metasetkeys{upgrade}{#1}%
\ifx\upgrade@keygroup\@empty\def\@@group{#2}\else\def\@@group{\upgrade@keygroup}\fi
\rdfmeta@upgrade@base{#2}{\@nameuse{\@@group @\upgrade@optarg}}}
```

They set the metakeys from the second argument, then set `\@@group` to be the intended group (if the `keygroup` key was specified, it takes precedence over the default `#2`).

```
\newcommand\rdfmeta@upgrade@star[2][]{\metasetkeys{upgrade}{#1}%
\ifx\upgrade@keygroup\@empty\def\@@group{#2}\else\def\@@group{\upgrade@keygroup}\fi
\rdfmeta@upgrade@base@star{#2}{\@nameuse{\@@group @\upgrade@optarg}}}
```

`\rdfmeta@upgrade@base` This auxiliary macro and is invoked as `\rdfmeta@upgrade@base{`⟨*cseq*⟩`}{`⟨*optarg*⟩`}`, where ⟨*cseq*⟩ is a command sequence name. It checks if `\`⟨*cseq*⟩ is defined (if not it does nothing), saves the old behavior of `\`⟨*cseq*⟩ as `\rdfmeta@`⟨*cseq*⟩`@old`, and then redefines `\`⟨*cseq*⟩ to take a keyval argument and passes ⟨*optarg*⟩ as the optional argument.

```
\newcommand\rdfmeta@upgrade@base[2]{\@ifundefined{#1}{}%
{\message{redefining macro #1,}
\ifx\upgrade@idlabel\@empty\srefaddidkey{#1}\else\srefaddidkey[prefix=\upgrade@idlabel]{#1}\fi%
\expandafter\let\csname rdfmeta@#1@old\expandafter\endcsname\csname #1\endcsname%
\expandafter\renewcommand\csname #1\endcsname[2][]%
{\metasetkeys{#1}{##1}\@nameuse{rdfmeta@#1@old}[#2]{##2}}
\addmetakey*\@@group{\upgrade@optarg}}}
```

`\rdfmeta@upgrade@base@star` This is a variant of `\rdfmeta@upgrade@base`, which also takes care of the starred variants of a macro.

```
\newcommand\rdfmeta@upgrade@base@star[2]{\@ifundefined{#1}{}%
{\message{redefining macros #1 and #1*,}
\ifx\upgrade@idlabel\@empty\srefaddidkey{#1}\else\srefaddidkey[prefix=\upgrade@idlabel]{#1}\fi%
\expandafter\let\csname rdfmeta@#1@old\expandafter\endcsname\csname #1\endcsname%
```

In this case, we cannot just use `\newcommand` for dealing with the optional argument because the star is between the command sequence and the arguments. So we make a case distinction on the presence of the star. `\rdfmeta@`⟨*cseq*⟩`@old`.

```
\expandafter\renewcommand\csname #1\endcsname%
{\@ifstar{\@nameuse{rdfmeta@#1@star}}{\@nameuse{rdfmeta@#1@nostar}}}%
```

the macros `\rdfmeta@`⟨*cseq*⟩`@star` and `\rdfmeta@`⟨*cseq*⟩`@nostar` that are defined in terms of `\rdfmeta@`⟨*cseq*⟩`@old` handle the necessary cases. The second one is simple:

```
\expandafter\newcommand\csname rdfmeta@#1@nostar\endcsname[2][]%
{\metasetkeys{#1}{##1}\edef\@test{#2}%
\ifx\@test\@empty\@nameuse{rdfmeta@#1@old}{##2}%
\else\@nameuse{rdfmeta@#1@old}[#2]{##2}\fi}%
```

For `\rdfmeta@`⟨*cseq*⟩`@star` we have to take care of the optional argument of the old macro: if the `optargstar` key was set, then we pass the second argument of `\rdfmeta@upgrade@base` as an optional argument to it as above.

```
\ifx\upgrade@optargstar\@yes@%
\expandafter\newcommand\csname rdfmeta@#1@star\endcsname[2][]%
{\metasetkeys{#1}{##1}\@nameuse{rdfmeta@#1@old}*[#2]{##2}}%
\else%
\expandafter\newcommand\csname rdfmeta@#1@star\endcsname[2][]%
{\metasetkeys{#1}{##1}\@nameuse{rdfmeta@#1@old}*{##2}}%
\fi%
\addmetakey*\@@group{\upgrade@optarg}}}
```

3.4 Redefinitions

If the `sectioning` macro is set, we redefine the respective commands

```
\if@rdfmeta@sectioning
\message{redefining sectioning commands!}
\rdfmeta@upgrade*[optarg=short,optargstar=no]{part}
\rdfmeta@upgrade*[optarg=short,optargstar=no]{chapter}
\rdfmeta@upgrade*[optarg=short,optargstar=no]{section}
\rdfmeta@upgrade*[optarg=short,optargstar=no]{subsection}
\rdfmeta@upgrade*[optarg=short,optargstar=no]{subsubsection}
\rdfmeta@upgrade*[optarg=short,optargstar=no]{paragraph}
\fi
```

⟨/package⟩