`reqdoc.sty`: Semantic Markup for Requirements Specification Documents*

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

October 23, 2015

Abstract

This package provides an infrastructure for semantically enhanced requirements specifications used in software engineering. This allows to embed structural information into documents that can be used by semantic document management systems e.g. for management of change and requirements tracing.

*Version v0.3 (last revised 2012/11/23)

Contents

1 Introduction

In software engineering, the development process is accompanied with a trail of structured documents, user specifications, architecture specifications, test reports, etc. All of these documents[1]

EdN:1

For an example of a requirement document see the file `requirements.tex` provided in this package. [2]

EdN:2

2 The User Interface

2.1 Package Options

`recorddeps` The `reqdoc` package takes the package option `recorddeps`. If this is given, then the package generates an external file with dependencies that can be used by external systems like the `locutor` system[3], see Section **??**. If the `showmeta` is set, then the metadata keys are shown (see [**Kohlhase:metakeys:ctan**] for details and customization options).

EdN:3

`showmeta`

2.2 Requirements

The `reqdoc` package supplies two forms of writing down requirements that mainly differ in their presentation. We can have requirement lists and requirement tables.

`requirements` The `requirements` environment marks up a list of requirements. It takes an optional key/value list as an argument: if `numbering` is set to `yes` (the default), then the requirements are numbered for referencing it visually; the label is created using the prefix specified in the key `prefix`.

`requirement` The individual requirements are specified by the `requirement` environment, which takes an optional key/value list as an argument: the `id` key allows to specify a symbolic label for cross-referencing, the `prio` key allows to specify a priority of the requirement, the `reqs` key allows to specify a comma-separated list of labels of requibments this one depends on or refines. Finally, the visual label of the requirement can be fixed by the `num` key[4].

EdN:4

`reqtable` The `reqtable` environment is a varian of the `\requirements` environment that shows the requirements in a tabular form that gives a better overview; its optional key/value argument works the same. The respective requirements are marked up with the `\reqline` macro, which takes three arguments. The first one is an optional key/value specification and corresponds to be one on the `requirement` environment. The second one contains the actual text of the requirements and the third one a comment.

`\reqline`

`\importreqs` Note that if we want to refer to requirements from a document ⟨*doc*⟩, then we will need to know about their representations and can import the necessary information via `\importreqs{`⟨*doc*⟩`}`.

3 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. none reported yet

[1] EdNote: continue

[2] EdNote: need to bring this in line with the sref package

[3] EdNote: add citation here

[4] EdNote: this is not implemented yet

4 The Implementation

The reqdoc package generates to 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

```
⟨*ltxml⟩
 package LaTeXML::Package::Pool;
use strict;
use LaTeXML::Package;
⟨/ltxml⟩
```

4.1 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).[5] EdN:5

```
⟨*package⟩
\newif\if@deps\@depsfalse
\DeclareOption{recorddeps}{\@depstrue}
\DeclareOption*{\PassOptionsToPackage{\CurrentOption}{statements}}
\ProcessOptions
```

Then we load a couple of packages

```
\RequirePackage{statements}
\RequirePackage{longtable}
⟨/package⟩
⟨*ltxml⟩
DeclareOption('recorddeps', '');
DeclareOption(undef,sub {PassOptions('statements','sty',ToString(Digest(T_CS('\CurrentOption')))); });
ProcessOptions();
⟨/ltxml⟩
```

Then we register the namespace of the requirements ontology

```
⟨*ltxml⟩
RegisterNamespace('r'=>"http://omdoc.org/ontology/requirements#");
RegisterDocumentNamespace('r'=>"http://omdoc.org/ontology/requirements#");
⟨/ltxml⟩
```

4.2 Requirements

requirements

and now the requirements environment, it is empty at the moment[6] EdN:6

```
⟨*package⟩
\newif\ifreqsnum\reqsnumfalse
\addmetakey{reqs}{numbering}
\addmetakey[R]{reqs}{prefix}
\def\reqs@no{no}
\newenvironment{requirements}[1][]%
{\metasetkeys{reqs}{#1}\ifx\reqs@numbering\reqs@no\reqsnumfalse\else\reqsnumtrue\fi}{}
⟨/package⟩
⟨*ltxml⟩
DefEnvironment('{requirements} OptionalKeyVals:reqs',
       "<omdoc:omgroup type='itemize'>#body</omdoc:omgroup>");
⟨/ltxml⟩
```

[5] EdNote: need an implementation for LaTeXML

[6] EdNote: think about this again!

We define a group of keywords using the `\addmetakey` command from the `metakeys` package [**Kohlhase:metakeys:ctan**]. The group below, named as `req`, consists of three keywords `id`, `prio` and `refs`.

```
⟨*package⟩
\addmetakey{req}{id}
\addmetakey{req}{prio}
\addmetakey{req}{refs}
\addmetakey{req}{num}
\addmetakey*{req}{title}
\newcounter{reqnum}[section]
```

This function cycles over a comma-separated list and does the references

```
\def\req@do@refs#1#2{\let\@tmpop=\relax\@for\@I:=#1\do{\@tmpop\req@do@ref{\@I}\let\@tmpop=#2}}
```

EdN:7 The `\req@do@ref` command creates a hyperlink from [7]

```
\def\req@do@ref#1{\sref@hlink@ifh{#1}{\req@ref{#1}{number}}}
```

this function defines a requirement aspect the first arg is the label, the second one the aspect to be defined and the third one the value expand `csname` before `xdef`

The command `\req@def@aux` creates the name of a command, which is determined by the text given between `\csname` and `\endcsname`, and defines this command globally to function as `#3`. We use the command `\expandafter` in the definition of `\req@def@aux` to execute the command `\xdef` after `\csname` is executed.

```
\def\req@def@aux#1#2#3{\expandafter\xdef\csname req@#1@#2\endcsname{#3}}
```

this function takes the same arguments and writes the command to the aux file

```
\def\req@write@aux#1#2#3{\protected@write\@auxout{}{\string\req@def@aux{#1}{#2}{\thesection.#3}}}
```

and finally this function does both

```
\def\req@def#1#2#3{\req@def@aux{#1}{#2}{#3}\req@write@aux{#1}{#2}{#3}}
```

this function references an aspect of a requirement.

```
\def\req@ref#1#2{\csname req@#1@#2\endcsname}
```

these functions print the priority, label, and references (if specified)

```
\def\print@req@prio{\ifx\req@prio\@empty\else(Priority: \req@prio)\fi}
\def\print@req@label{\sref@target@ifh\req@id{\reqs@prefix\arabic{reqnum}: }}
\def\print@req@refs{\ifx\req@refs\@empty\else\hfill [from~\req@do@refs{\req@refs}{,}]\fi}
```

EdN:8 [8] First argument is a list of key-value pairs which are assigned to req. Increase the counter reqnum, i.e., increase the requirement number. Remember the number for reference. Print the requirement label (with the requirement number) Print the priority? Print the requirement (given as arg 2) Print the references We define a new command `\reqnote` to annotate the notes given for a requirement. The command `\reqnote` simply prints the note, which is given by the user as a text, in the form `Note: <text>`.

`requirement`

```
\newenvironment{requirement}[1][]%
{\metasetkeys{req}{#1}\stepcounter{reqnum}
\ifreqsnum\ifx\req@id\@empty\else\req@def\req@id{number}\thereqnum\fi
\noindent\textbf{\print@req@label}\fi
\newcommand\reqnote[1]{\par\noindent Note: ##1}
\print@req@prio}
{\medskip\print@req@refs}
⟨/package⟩
⟨*ltxml⟩
DefEnvironment('{requirement} OptionalKeyVals:req',
      "<omdoc:omtext ?&GetKeyVal(#1,'id')(xml:id='&GetKeyVal(#1,'id')')() r:dummy='to ensure the namespace'>"
```

[7] EdNote: What is req at ref? It has appeared for the first time.
[8] EdNote: What are number and 0?

```
      . "<omdoc:meta property='texttype' content='r:requirement'/>"
     .  "?&GetKeyVal(#1,'refs')(<omdoc:link rel='r:dependsOn' href='#&GetKeyVal(#1,'refs')'/>)()"
      .  "#body"
      ."</omdoc:omtext>");
DefConstructor('\reqnote{}',
       "<omdoc:note type='requirement'>#1</omdoc:note>");
⟨/ltxml⟩
```

requment

```
⟨*package⟩
\def\st@reqment@initialize{}\def\st@reqment@terminate{}
\define@statement@env{reqment}
\def\st@reqment@kw{Requirement}
\theorembodyfont{\upshape}
\newtheorem{STreqmentEnv}[STtheoremAssEnv]{\st@reqment@kw}
⟨/package⟩
```

reqtable

```
⟨*package⟩
\newenvironment{reqtable}[1][]{\metasetkeys{reqs}{#1}
\begin{center}\begin{longtable}{|l|l|p{6cm}|p{5cm}|l|}\hline
\# & Prio & Requirement & Notes & Refs\\\hline\hline}
{\end{longtable}\end{center}}
⟨/package⟩
⟨*ltxml⟩
DefEnvironment('{reqtable} OptionalKeyVals:reqs',
       "<omdoc:omgroup type='itemize'>#body</omdoc:omgroup>");
⟨/ltxml⟩
```

\reqline

```
⟨*package⟩
\newcommand\reqline[3][]%
{\metasetkeys{req}{#1}\stepcounter{reqnum}
\req@def\req@id{number}\thereqnum% remember the number for reference
\textbf{\sref@target@ifh\req@id{\reqs@prefix\arabic{reqnum}}}&
\req@prio &#2&#3&\req@do@refs\req@refs{,}\tabularnewline\hline}
⟨/package⟩
⟨*ltxml⟩
DefConstructor('\reqline OptionalKeyVals:req{}{}',
       "<omdoc:omtext type='requirement'><omdoc:CMP>#2</omdoc:CMP></omdoc:omtext>"
      ."<omdoc:omtext type='note'><omdoc:CMP>#3</omdoc:CMP></omdoc:omtext>");
⟨/ltxml⟩
```

\importreqs The \importreqs macro reports a dependency to the dependencies file. and then reads the aux file specified in the argument.

```
⟨*package⟩
\newcommand\importreqs[1]{\req@dep@write{"#1.tex"}{IMPORTREQS}\makeatletter\input{#1.aux}\makeatother}
⟨/package⟩
⟨*ltxml⟩
DefConstructor('\importreqs {}',"<omdoc:imports from='#1'/>");
⟨/ltxml⟩
```

\rinput The \rinput macro[9] inputs the file and protocols this in the dependencies file. Note that this only takes place on the top level; i.e. the \@ifdeps switch is set to false. EdN:9

```
⟨*package⟩
\newcommand\rinput[1]{\req@dep@write{"#1.tex"}{[dt="input"]}\bgroup\@depsfalse\input{#1}\egroup}
```

[9] EdNote: this should go somewhere up; probably merge with sinput; which should also go into the stex package.

```
⟨/package⟩
⟨*ltxml⟩
DefMacro('\rinput','\input');
⟨/ltxml⟩
```

4.3 Recording the dependencies for Change Management

The macros in this section record dependencies in a special file to be used in change management by the `locutor` system. This is still not optimal, since we do not know the actual path.

```
⟨*package⟩
\if@deps\newwrite\req@depfile
\immediate\openout\req@depfile=\jobname.deps
\AtEndDocument{\closeout\req@depfile}
⟨/package⟩
```

EdN:10 we redefine the `\importmodule` command, so that it does the reporting. [10]

```
⟨*package⟩
\renewcommand{\importmodule}[2][]{\req@dep@write{"#1.tex"}{[dt="importmodule"]}\def\@test{#1}%
\ifx\@test\@empty\else\requiremodules{#1}{sms}\fi
\expandafter\gdef\csname module#2@path\endcsname{#1}
\activate@defs{#2}\export@defs{#2}}
\fi
⟨/package⟩
```

```
⟨*package⟩
\def\req@dep@write#1#2{\if@deps\protected@write\req@depfile{}{#1 #2}\fi}
⟨/package⟩
```

4.4 Finale

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

```
⟨ltxml⟩1;
```

[10] EdNote: MK: this probably does not work after the refactoring of importmodule; rework.