

`problem.sty`: An Infrastructure for formatting Problems*

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

April 27, 2014

Abstract

The `problem` package supplies an infrastructure that allows specify problems and to reuse them efficiently in multiple environments.

Contents

1	Introduction	2
2	The User Interface	2
2.1	Package Options	2
2.2	Problems and Solutions	2
2.3	Starting and Stopping Solutions	4
2.4	Including Problems	4
2.5	Reporting Metadata	4
2.6	Support for <code>MathHub</code>	4
3	Limitations	5
4	The Implementation	6
4.1	Package Options	6
4.2	Problems and Solutions	7
4.3	Including Problems	10
4.4	Reporting Metadata	11
4.5	Support for <code>MathHub</code>	12
4.6	Providing IDs Elements	12
4.7	Finale	13

*Version v1.1 (last revised 2013/12/12)

1 Introduction

The `problem` package supplies an infrastructure that allows specify problem. Problems are text fragments that come with auxiliary functions: hints, notes, and solutions¹. Furthermore, we can specify how long the solution to a given problem is estimated to take and how many points will be awarded for a perfect solution.

Finally, the `problem` package facilitates the management of problems in small files, so that problems can be re-used in multiple environment.

2 The User Interface

2.1 Package Options

<code>solutions</code>	The <code>problem</code> package takes the options <code>solutions</code> (should solutions be output?),
<code>notes</code>	<code>notes</code> (should the problem notes be presented?), <code>hints</code> (do we give the hints?),
<code>hints</code>	<code>pts</code> (do we display the points awarded for solving the problem?), <code>min</code> (do we
<code>pts</code>	display the estimated minutes for problem soling). If theses are specified, then the
<code>min</code>	corresponding auxiliary parts of the problems are output, otherwise, they remain
	invisible.
<code>boxed</code>	The <code>boxed</code> option specifies that problems should be formatted in framed boxes
<code>test</code>	so that they are more visible in the text. Finally, the <code>test</code> option signifies that
	we are in a test situation, so this option does not show the solutions (of course),
	but leaves space for the students to solve them.
<code>showmeta</code>	Finally, if the <code>showmeta</code> is set, then the metadata keys are shown (see [Koh13]
	for details and customization options).

2.2 Problems and Solutions

<code>problem</code>	The main environment provided by the <code>problem</code> package is (surprise surprise) the <code>problem</code> environment. It is used to mark up problems and exercises. The
<code>id</code>	environment takes an optional KeyVal argument with the keys <code>id</code> as an identifier
<code>pts</code>	that can be reference later, <code>pts</code> for the points to be gained from this exercise in
<code>min</code>	homework or quiz situations, <code>min</code> for the estimated minutes needed to solve the
<code>title</code>	problem, and finally <code>title</code> for an informative title of the problem. For an example
	of a marked up problem see Figure 1 and the resulting markup see Figure 2.
<code>solution</code>	The <code>solution</code> environment can be to specify a solution to a problem. If the
<code>solutions</code>	<code>solutions</code> option is set or <code>\solutionstrue</code> is set in the text, then the solution
	will be presented in the output. The <code>solution</code> environment takes an optional
<code>id</code>	KeyVal argument with the keys <code>id</code> for an identifier that can be reference <code>for</code> to
<code>for</code>	specify which problem this is a solution for, and <code>height</code> that allows to specify the
<code>height</code>	amount of space to be left in test situations (i.e. if the <code>test</code> option is set in the
<code>test</code>	<code>\usepackage</code> statement).
<code>hint</code>	, the <code>hint</code> and <code>exnote</code> environments can be used in a <code>problem</code> environment to
<code>note</code>	

¹for the moment multiple choice problems are not supported, but may well be in a future version

```

\usepackage[solutions,hints,pts,min]{problem}
\begin{document}
  \begin{problem}[id=elephants,pts=10,min=2,title=Fitting Elephants]
    How many Elephants can you fit into a Volkswagen beetle?
  \begin{hint}
    Think positively, this is simple!
  \end{hint}
  \begin{exnote}
    Justify your answer
  \end{exnote}
  \begin{solution}[for=elephants,height=3cm]
    Four, two in the front seats, and two in the back.
  \end{solution}
  \end{problem}
\end{document}

```

Example 1: A marked up Problem

Problem 2.1 (Fitting Elephants)

How many Elephants can you fit into a Volkswagen beetle?

Hint: Think positively, this is simple!

Note: Justify your answer

Solution: Four, two in the front seats, and two in the back.

Example 2: The Formatted Problem from Figure 1

give hints and to make notes that elaborate certain aspects of the problem.

2.3 Starting and Stopping Solutions

Sometimes we would like to locally override the `solutions` option we have given to the package. To turn on solutions we use the `\startsolutions`, to turn them off, `\stopsolutions`. These two can be used at any point in the documents.

2.4 Including Problems

`\includeproblem` The `\includeproblem` macro can be used to include a problem from another file. It takes an optional `KeyVal` argument and a second argument which is a path to the file containing the problem (the macro assumes that there is only one problem in the include file). The keys `title`, `min`, and `pts` specify the problem title, the estimated minutes for solving the problem and the points to be gained, and their values (if given) overwrite the ones specified in the `problem` environment in the included file.

2.5 Reporting Metadata

The sum of the points and estimated minutes (that we specified in the `pts` and `min` keys to the `problem` environment or the `\includeproblem` macro) to the log file and the screen after each run. This is useful in preparing exams, where we want to make sure that the students can indeed solve the problems in an allotted time period.

The `\min` and `\pts` macros allow to specify (i.e. to print to the margin) the distribution of time and reward to parts of a problem, if the `pts` and `pts` package options are set. This allows to give students hints about the estimated time and the points to be awarded.

2.6 Support for MathHub

Much of the \LaTeX content is hosted on MathHub (<http://MathHub.info>), a portal and archive for flexiformal mathematics. MathHub offers GIT repositories (public and private escrow) for mathematical documentation projects, online and offline authoring and document development infrastructure, and a rich, interactive reading interface. The `modules` package supports repository-sensitive operations on MathHub.

Note that MathHub has two-level repository names of the form $\langle group \rangle / \langle repo \rangle$, where $\langle group \rangle$ is a MathHub-unique repository group and $\langle repo \rangle$ a repository name that is $\langle group \rangle$ -unique. The file and directory structure of a repository is arbitrary – except that it starts with the directory `source` because they are Math Archives in the sense of [Hor+11]. But this structure can be hidden from the \LaTeX author with MathHub-enabled versions of the `modules` macros.

`\includemhproblem` The `\includemhproblem` macro is a variant of `\importmodule` with repository support. Instead of writing

```
\defpath{MathHub}{/user/foo/lmh/MathHub}
\includeproblem[pts=7]{\MathHub{fooMH/bar/source/baz/foobar}}
```

we can simply write (assuming that `\MathHub` is defined as above)

```
\includemhproblem[fooMH/bar]{baz/foobar}
```

Note that the `\importmhmodule` form is more semantic, which allows more advanced document management features in `MathHub`.

If `baz/foobar` is the “current module”, i.e. if we are on the `MathHub` path `...MathHub/fooMH/bar...`, then stating the repository in the first optional argument is redundant, so we can just use

```
\includemhproblem{baz/foobar}
```

Of course, neither `LATEX` nor `LATEXML` know about the repositories when they are called from a file system, so we can use the `\mhcurrentrepos` macro from the `modules` package to tell them. But this is only needed to initialize the infrastructure in the driver file. In particular, we do not need to set it in each module, since the `\importmhmodule` macro sets the current repository automatically.

Caveat if you want to use the `MathHub` support macros (let’s call them `mh`-variants), then every time a module is imported or a document fragment is included from another repos, the `mh`-variant `\importmhmodule` must be used, so that the “current repository” is set accordingly. To be exact, we only need to use `mh`-variants, if the imported module or included document fragment use `mh`-variants.

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 TRAC` [sTeX].

1. none reported yet

4 The Implementation

The `problem` package generates two files: the \LaTeX package (all the code between `*package` and `/package`) and the \LaTeX XML 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.

4.1 Package Options

The first step is to declare (a few) package options that handle whether certain information is printed or not. They all come with their own conditionals that are set by the options.

```
1 <*package>
2 \DeclareOption{showmeta}{\PassOptionsToPackage{\CurrentOption}{metakeys}}
3 \newif\ifexnotes\exnotesfalse\DeclareOption{notes}{\exnotestruer}
4 \newif\ifhints\hintsfalse\DeclareOption{hints}{\hintstruer}
5 \newif\ifsolutions\solutionfalse\DeclareOption{solutions}{\solutionstruer}
6 \newif\ifpts\ptsfalse\DeclareOption{pts}{\ptstruer}
7 \newif\ifmin\minfalse\DeclareOption{min}{\mintruer}
8 \newif\ifboxed\boxedfalse\DeclareOption{boxed}{\boxedtruer}
9 \ProcessOptions
10 </package>
```

On the \LaTeX XML side we only make sure that the switches are defined

```
11 <*ltxml>
12 RawTeX('
13 \newif\ifexnotes\exnotesfalse
14 \newif\ifhints\hintsfalse
15 \newif\ifsolutions\solutionfalse
16 \newif\ifpts\ptsfalse
17 \newif\ifmin\minfalse
18 \newif\ifboxed\boxedfalse
19 ');
20 </ltxml>
```

Then we make sure that the necessary packages are loaded (in the right versions).

```
21 <*package>
22 \RequirePackage{comment}
23 \RequirePackage{sref}
24 \RequirePackage{mdframed}
25 %\RequirePackage{marginnote}
26 </package>
```

Here comes the equivalent header information for \LaTeX XML, we also initialize the package inclusions. Since \LaTeX XML currently does not process package options, we have nothing to do.

```
27 <*ltxml>
28 # -*- CPERL -*-
29 package LaTeXML::Package::Pool;
```

```

30 use strict;
31 use LaTeXML::Package;
32 RequirePackage('sref');
33 </txml>

    Then we register the namespace of the requirements ontology
34 <*txml>
35 RegisterNamespace('prob'=>"http://omdoc.org/ontology/problems#");
36 RegisterDocumentNamespace('prob'=>"http://omdoc.org/ontology/problems#");
37 </txml>

```

4.2 Problems and Solutions

We now prepare the KeyVal support for problems. The key macros just set appropriate internal macros.

```

38 <*package>
39 \srefaddidkey[prefix=prob.]{problem}
40 \addmetakey{problem}{pts}
41 \addmetakey{problem}{min}
42 \addmetakey*{problem}{title}
43 \addmetakey{problem}{refnum}

```

Then we set up a box and a counter for problems

```

44 \newcounter{problem}[section]

```

`\prob@number` We consolidate the problem number into a reusable internal macro

```

45 \def\prob@number{\ifx\inclprob@refnum\@empty%
46 \ifx\problem@refnum\@empty\thesection.\theproblem\else\problem@refnum\fi%
47 \inclprob@refnum\fi}

```

We consolidate the problem header line into a separate internal macro that can be reused in various settings.

`\prob@heading` We consolidate the problem header line into a separate internal macro that can be reused in various settings.

```

48 \def\prob@heading{Problem \prob@number%
49 \ifx\sref@id\@empty\else{\sref@label@id{Problem \thesection.\theproblem}}\fi%
50 \ifx\inclprob@title\@empty% if there is no outside title
51 \ifx\problem@title\@empty{: \quad}\else{\quad(\problem@title)\hfill\\}\fi
52 \else\quad(\inclprob@title)\hfill\\}\fi}% else show the outside title

```

With this in place, we can now define the `problem` environment. It comes in two shapes, depending on whether we are in boxed mode or not. In both cases we increment the problem number and output the points and minutes (depending) on whether the respective options are set.

`problem`

```

53 \newenvironment{problem}[1] [] {\metasetkeys{problem}{#1}\sref@target%
54 \stepcounter{problem}\record@problem%

```

```

55 \def\currentsectionlevel{problem\xspace}%
56 \def\Currentsectionlevel{Problem\xspace}%
57 \par\noindent\textbf{\prob@heading\show@pts\show@min\rm\ignorespaces}
58 {\smallskip}
59 \ifboxed\surroundwithmdframed{problem}\fi
60 \end{package}

```

Note that we allow hints and solutions in the body of a `problem` environment so we have to allow the `omdoc:CM` and `ltx:p` elements to autoopen and autoclose.

```

61 \begin{xml}
62 \DefEnvironment('problem' OptionalKeyVals:problem',
63 "<omdoc:exercise ?&GetKeyVal(#1,'id')(xml:id='&GetKeyVal(#1,'id')')()>"
64 .   "?&GetKeyVal(#1,'title')(<dc:title>&GetKeyVal(#1,'title')</dc:title>())"
65 .   "?&GetKeyVal(#1,'min')("
66 .   .   "<omdoc:meta property='prob:solvedinminutes' prob:dummy='for the namespace'>"
67 .   .   "&GetKeyVal(#1,'min')("
68 .   .   "</omdoc:meta>())"
69 .   .   "?&GetKeyVal(#1,'pts')("
70 .   .   "<omdoc:meta property='prob:points' prob:dummy='for the namespace'>"
71 .   .   "&GetKeyVal(#1,'pts')("
72 .   .   "</omdoc:meta>())"
73 .   .   "#body"
74 .   .   "</omdoc:exercise>",
75   afterDigest => sub {
76     my ($stomach,$kv)=@_;
77     my $kvi = LookupValue('inclprob');
78     my @keys = qw(id title min pts);
79     my @vals = $kvi && map($kvi->getValue($_), @keys);
80     foreach my $i(0..$#vals) {
81       $kv->setValue($keys[$i],$vals[$i]) if $vals[$i];
82     }
83     return;};#$
84 \end{xml}

```

`\record@problem` This macro records information about the problems in the `*.aux` file.

```

85 \begin{package}
86 \def\record@problem{\protected@write\@auxout{}%
87 {\string\@problem{\prob@number}%
88 {\ifx\inclprob@pts\empty\problem@pts\else\inclprob@pts\fi}%
89 {\ifx\inclprob@min\empty\problem@min\else\inclprob@min\fi}}
90 \end{package}

```

`\@problem` This macro acts on a problem's record in the `*.aux` file. It does not have any functionality here, but can be redefined elsewhere (e.g. in the `assignment` package).

```

91 \begin{package}
92 \def\@problem#1#2#3{}
93 \end{package}

```


`solution` The `solution` environment is similar to the `problem` environment, only that it is independent of the boxed mode. It also has it's own keys that we need to define first.

```

94 <*package>
95 \srefaddidkey{soln}
96 \addmetakey{soln}{for}
97 \addmetakey{soln}{height}
98 % \begin{macrocode}
99 % the next step is to define a helper macro that does what is needed to start a solution.
100 % \begin{macrocode}
101 \newcommand\@startsolution[1][\metasetkeys{soln}{#1}%
102 \ifboxed\else\hrule\fi\smallskip{\bf Solution: }\begin{small}%
103 \def\currentsectionlevel{solution\hspace}%
104 \def\Currentsectionlevel{Solution\hspace}%
105 \ignorespaces}

```

`\startsolutions` for the `\startsolutions` macro we use the `\specialcomment` macro from the `comment` package. Note that we use the `\@startsolution` macro in the start codes, that parses the optional argument.

```

106 \newcommand\startsolutions{\specialcomment{solution}{\@startsolution}%
107 {\ifboxed\else\hrule\fi\end{small}}}%
108 \ifboxed\surroundwithmdframed{solution}\fi
109 </package>
110 <*ltxml>
111 DefConstructor('\startsolutions','');
112 </ltxml>

```

`\stopsolutions`

```

113 <*package>
114 \newcommand\stopsolutions{\excludecomment{solution}}
115 </package>
116 <*ltxml>
117 DefConstructor('\stopsolutions','');
118 </ltxml>

```

so it only remains to start/stop solutions depending on what option was specified.

```

119 <*package>
120 \ifsolutions\startsolutions\else\stopsolutions\fi
121 </package>

```

the LaTeXML binding for the `solutions` is straightforward.

```

122 <*ltxml>
123 DefKeyVal('soln','id','Semiverbatim');
124 DefKeyVal('soln','height','Semiverbatim');
125 DefKeyVal('soln','for','Semiverbatim');
126 DefEnvironment('{solution} OptionalKeyVals:soln',
127 " <omdoc:solution ?&GetKeyVals(#1,'for')(for='&GetKeyVal(#1,'for'))(> "

```

```

128 . "#body"
129 . "</omdoc:solution>");
130 </ltxml>

131 <*package>
132 \ifexnotes
133 \newenvironment{exnote}[1] []%
134 {\par\noindent\hrule\smallskip{\bf Note: }\small}
135 {\smallskip\hrule}
136 \else%ifexnotes
137 \excludecomment{exnote}
138 \fi%ifexnotes
139 \ifhints
140 \newenvironment{hint}[1] []%
141 {\par\noindent\hrule\smallskip{\bf Hint: }\small}
142 {\smallskip\hrule}
143 \newenvironment{exhint}[1] []%
144 {\par\noindent\hrule\smallskip{\bf Hint: }\small}
145 {\smallskip\hrule}
146 \else%ifhints
147 \excludecomment{hint}
148 \excludecomment{exhint}
149 \fi%ifhints
150 </package>
151 <*ltxml>
152 DefEnvironment('{exnote}', "<omdoc:hint>#body</omdoc:hint>");
153 DefEnvironment('{hint}', "<omdoc:hint>#body</omdoc:hint>");
154 DefConstructor('\pts{ }', "");
155 DefConstructor('\min{ }', "");
156 </ltxml>

```

4.3 Including Problems

`\includeproblem` The `\includeproblem` command is essentially a glorified `\input` statement, it sets some internal macros first that overwrite the local points. Importantly, it resets the `inclprob` keys after the input.

```

157 <*package>
158 \addmetakey{inclprob}{pts}
159 \addmetakey{inclprob}{min}
160 \addmetakey*{inclprob}{title}
161 \addmetakey{inclprob}{refnum}
162 \addmetakey{inclprob}{mhrepos}
163 \clear@inclprob@keys
164 \newcommand\includeproblem[2] [] {\metasetkeys{inclprob}{#1}%
165 \input{#2}\clear@inclprob@keys}
166 </package>
167 <*ltxml>
168 DefKeyVal('prob', 'pts', 'Semiverbatim');
169 DefKeyVal('prob', 'min', 'Semiverbatim');
170 DefKeyVal('prob', 'title', 'Semiverbatim');

```

```

171 DefKeyVal('prob','refnum','Semiverbatim');
172 DefConstructor('\includeproblem OptionalKeyVals:prob Semiverbatim',
173   "<omdoc:exercise tref='#2'>"
174 .   "?&GetKeyVal(#1,'title')(<dc:title>&GetKeyVal(#1,'title')</dc:title>())"
175 .   "?&GetKeyVal(#1,'min')("
176 .     "<omdoc:meta property='prob:solvedinminutes' prob:dummy='for the namespace'>"
177 .       "&GetKeyVal(#1,'min')("
178 .         "</omdoc:meta>())"
179 .   "?&GetKeyVal(#1,'pts')("
180 .     "<omdoc:meta property='prob:points' prob:dummy='for the namespace'>"
181 .       "&GetKeyVal(#1,'pts')("
182 .         "</omdoc:meta>())"
183 .   "</omdoc:exercise>",
184   afterDigest => sub{
185     my ($stomach,$kv) = @_;
186     AssignValue('inclprob',$kv) if $kv;
187   });
188 </ltxml>

189 <*ltxml>
190 Tag('omdoc:exercise',afterOpen=>\&numberIt);
191 Tag('omdoc:solution',afterOpen=>\&numberIt);
192 Tag('omdoc:hint',afterOpen=>\&numberIt);
193 </ltxml>

```

4.4 Reporting Metadata

```

194 <*package>
195 \def\pts#1{\ifpts\marginpar{#1 pt}\fi}
196 \def\min#1{\ifmin\marginpar{#1 min}\fi}
197 </package>
198 <*ltxml>
199 </ltxml>

200 <*package>
201 \AtEndDocument{\ifpts\message{Total: \arabic{pts} points}\fi}
202 \ifmin\message{Total: \arabic{min} minutes}\fi}
203 </package>
204 <*ltxml>
205 </ltxml>

```

`\show@pts` The `\show@pts` shows the points: if no points are given from the outside and also no points are given locally do nothing, else show and add. If there are outside points then we show them in the margin.

```

206 <*package>
207 \newcounter{pts}
208 \def\show@pts{\ifx\inclprob@pts\empty%
209 \ifx\problem@pts\empty\else%
210 \ifpts\marginpar{\problem@pts pt\smallskip}\addtocounter{pts}{\problem@pts}\fi%
211 \fi\else% inclprob@pts nonempty
212 \ifpts\marginpar{\inclprob@pts pt\smallskip}\addtocounter{pts}{\inclprob@pts}\fi%

```

```
213 \fi}
```

and now the same for the minutes

```
\show@min
```

```
214 \newcounter{min}
215 \def\show@min{\ifx\inclprob@min\@empty%
216 \ifx\problem@min\@empty\else%
217 \ifmin\marginpar{\problem@min min}\addtocounter{min}{\problem@min}\fi%
218 \fi\else%
219 \ifmin\marginpar{\inclprob@min min}\addtocounter{min}{\inclprob@min}\fi
220 \fi}
221 \</package>
```

4.5 Support for MathHub

`\includemhproblem` The `\includemhproblem` saves the current value of `\mh@currentrepos` in a local macro `\mh@@repos`, resets `\mh@currentrepos` to the new value if one is given in the optional argument, and after importing resets `\mh@currentrepos` to the old value in `\mh@@repos`.

```
222 <*package>
223 \newcommand\includemhproblem[2][\metasetkeys{inclprob}{#1}]%
224 \edef\mh@@repos{\mh@currentrepos}%
225 \ifx\inclprob@mhrepos\@empty\else\mhcurrentrepos\inclprob@mhrepos\fi%
226 \input{\MathHub{\mh@currentrepos/source/#2}}%
227 \mhcurrentrepos\mh@@repos\clear@inclprob@keys}
228 \</package>
229 <*ltxml>
230 sub includemhproblem {
231   my ($gullet,$keyval,$arg2) = @_;
232   my $repo_path;
233   if ($keyval) {
234     $repo_path = ToString(GetKeyVal($keyval,'mhrepos')); }
235   if (! $repo_path) {
236     $repo_path = ToString(Digest(T_CS('\mh@currentrepos'))); }
237   else {
238     $keyval->setValue('mhrepos',undef); }
239   my $mathhub_base = ToString(Digest('\MathHub{'}));
240   my $finalpath = $mathhub_base.$repo_path.'/source/'.ToString($arg2);
241   return Invocation(T_CS('\includeproblem'), $keyval, T_OTHER($finalpath)); }##
242 DefKeyVal('inclprob','mhrepos','Semiverbatim');
243 DefMacro('\includemhproblem OptionalKeyVals:inclprob {}', \&includemhproblem);
244 \</ltxml>
```

4.6 Providing IDs Elements

To provide default identifiers, we tag all elements that allow `xml:id` attributes by executing the `numberIt` procedure from `omdoc.sty.ltxml`.

```
245 <*/txml>
246 Tag('omdoc:exercise',afterOpen=>\&numberIt,afterClose=>\&locateIt);
247 Tag('omdoc:solution',afterOpen=>\&numberIt,afterClose=>\&locateIt);
248 Tag('omdoc:hint',afterOpen=>\&numberIt,afterClose=>\&locateIt);
249 </txml>
```

4.7 Finale

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

```
250 <txml>1;
```