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

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

November 1, 2015

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 MathHub 4

3 Limitations **5**

4 The Implementation **6**
4.1 Problems and Solutions 7
4.2 Including Problems 11
4.3 Reporting Metadata 12
4.4 Support for MathHub 13
4.5 Providing IDs Elements 13
4.6 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 solutions[1]. 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

solutions
notes
hints
pts
min

The problem package takes the options solutions (should solutions be output?), notes (should the problem notes be presented?), hints (do we give the hints?), pts (do we display the points awarded for solving the problem?), min (do we display the estimated minutes for problem soling). If theses are specified, then the corresponding auxiliary parts of the problems are output, otherwise, they remain invisible.

boxed
test

The boxed option specifies that problems should be formatted in framed boxes so that they are more visible in the text. Finally, the test 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.

showmeta

Finally, if the showmeta is set, then the metadata keys are shown (see [Koh15] for details and customization options).

2.2 Problems and Solutions

problem
id
pts
min
title

The main environment provided by the problem package is (surprise surprise) the problem environment. It is used to mark up problems and exercises. The environment takes an optional KeyVal argument with the keys id as an identifier that can be reference later, pts for the points to be gained from this exercise in homework or quiz situations, min for the estimated minutes needed to solve the problem, and finally title 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.

solution
solutions
id
for
height
test

The solution environment can be to specify a solution to a problem. If the solutions option is set or \solutionstrue is set in the text, then the solution will be presented in the output. The solution environment takes an optional KeyVal argument with the keys id for an identifier that can be reference for to specify which problem this is a solution for, and height that allows to specify the amount of space to be left in test situations (i.e. if the test option is set in the \usepackage statement).

hint
note

, the hint and exnote environments can be used in a problem environment to

[1] for 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=elefants,pts=10,min=2,title=Fitting Elefants]
    How many Elefants 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=elefants,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 1 (Fitting Elefants)
How many Elefants 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.

`\startsolutions`
`\stopsolutions`

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.

`title`
`min`
`pts`

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 sTEX content is hosed 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 ⟨*group*⟩`/`⟨*repo*⟩, where ⟨*group*⟩ is a MathHub-unique repository group and ⟨*repo*⟩ a repository name that is ⟨*group*⟩-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 sTEX 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 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 GitHub repository [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 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.

First the general setup for LaTeXML

```
⟨*ltxml⟩
# -*- CPERL -*-
package LaTeXML::Package::Pool;
use strict;
use LaTeXML::Package;
⟨/ltxml⟩
%    \begin{macrocode}
%
% \subsection{Package Options}\label{sec:impl: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.
%
%    \begin{macrocode}
⟨*package⟩
\DeclareOption{showmeta}{\PassOptionsToPackage{\CurrentOption}{metakeys}}
\newif\ifexnotes\exnotesfalse
\DeclareOption{notes}{\exnotestrue}
\newif\ifhints\hintsfalse
\DeclareOption{hints}{\hintstrue}
\newif\ifsolutions\solutionsfalse
\DeclareOption{solutions}{\solutionstrue}
\newif\ifpts\ptsfalse
\DeclareOption{pts}{\ptstrue}
\newif\ifmin\minfalse
\DeclareOption{min}{\mintrue}
\newif\ifboxed\boxedfalse
\DeclareOption{boxed}{\boxedtrue}
\DeclareOption*{\PassOptionsToPackage{\CurrentOption}{omtext}}
\ProcessOptions
⟨/package⟩
```

On the LaTeXML side we only make sure that the switches are defined. Since LaTeXML currently does not process package options, we have nothing to do.

```
⟨*ltxml⟩
RawTeX('
\newif\ifexnotes\exnotesfalse
\newif\ifhints\hintsfalse
\newif\ifsolutions\solutionsfalse
\newif\ifpts\ptsfalse
\newif\ifmin\minfalse
```

```
\newif\ifboxed\boxedfalse
');
DeclareOption('notes', '');
DeclareOption('hints', '');
DeclareOption('solutions', '');
DeclareOption('pts', '');
DeclareOption('min', '');
DeclareOption('boxed', '');
DeclareOption(undef,sub {PassOptions('omtext','sty',ToString(Digest(T_CS('\CurrentOption'))));
ProcessOptions();
⟨/ltxml⟩
```

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

```
⟨*package⟩
\RequirePackage{omtext}
\RequirePackage{comment}
\RequirePackage{mdframed}
\RequirePackage[base]{babel}
⟨/package⟩
⟨*ltxml⟩
RequirePackage('omtext');
⟨/ltxml⟩
```

Then we register the namespace of the requirements ontology

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

`\prob@*@kw` For multilinguality, we define internal macros for keywords that can be specialized in `*.ldf` files.

```
⟨*package⟩
\AfterBabelLanguage{ngerman}{\input{problems-ngerman.ldf}}
\def\prob@problem@kw{Problem}
\def\prob@solution@kw{Solution}
⟨/package⟩
```

4.1 Problems and Solutions

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

```
⟨*package⟩
\srefaddidkey[prefix=prob.]{problem}
\addmetakey{problem}{pts}
\addmetakey{problem}{min}
\addmetakey*{problem}{title}
\addmetakey{problem}{refnum}
```

Then we set up a counter for problems.

```
\newcounter{problem}
```

\prob@label We provide the macro \prob@label to redefine later to get context involved.

```
\newcommand\prob@label[1]{#1}
```

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

```
\def\prob@number{\ifx\inclprob@refnum\@empty%
\ifx\problem@refnum\@empty\prob@label\theproblem%
\else\prob@label\problem@refnum\fi%
\else\prob@label\inclprob@refnum\fi}
```

\prob@title We consolidate the problem title into a reusable internal macro as well. \prob@title takes three arguments the first is the fallback when no title is given at all, the second and third go around the title, if one is given.

```
\newcommand\prob@title[3]{%
\ifx\inclprob@title\@empty% if there is no outside title
\ifx\problem@title\@empty{#1}\else{#2\problem@title{#3}}\fi
\else{#2}\inclprob@title{#3}\fi}% else show the outside title
```

With these the problem header is a one-liner

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

```
\def\prob@heading{\prob@problem@kw~\prob@number\prob@title{ }{ (}{)\strut\\}%
\sref@label@id{\prob@problem@kw~\prob@number}}
```

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

```
\newenvironment{problem}[1][]{\metasetkeys{problem}{#1}\sref@target%
\@in@omtexttrue% we are in a statement (for inline definitions)
\stepcounter{problem}\record@problem%
\def\current@section@level{\prob@problem@kw}%
\par\noindent\textbf\prob@heading\show@pts\show@min\rm\noindent\ignorespaces}
{\smallskip}
\ifboxed\surroundwithmdframed{problem}\fi
⟨/package⟩
```

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

```
⟨*ltxml⟩
DefEnvironment('{problem} OptionalKeyVals:problem',
 "<omdoc:exercise ?&GetKeyVal(#1,'id')(xml:id='&GetKeyVal(#1,'id')')()>"
.    "?&GetKeyVal(#1,'title')(<dc:title>&GetKeyVal(#1,'title')</dc:title>)()"
 .    "?&GetKeyVal(#1,'min')("
   .       "<omdoc:meta property='prob:solvedinminutes' prob:dummy='for the namespace'>"
 .          "&GetKeyVal(#1,'min')"
   .       "</omdoc:meta>)()"
```

```
  .    "?&GetKeyVal(#1,'pts')("
    .       "<omdoc:meta property='prob:points' prob:dummy='for the namespace'>"
  .         "&GetKeyVal(#1,'pts')"
    .       "</omdoc:meta>)()"
    .   "#body"
    ."</omdoc:exercise>",
    afterDigest => sub {
      my ($stomach,$kv)=@_;
      my $kvi = LookupValue('inclprob');
      my @keys = qw(id title min pts);
      my @vals = $kvi && map($kvi->getValue($_), @keys);
      foreach my $i(0..$#vals) {
        $kv->setValue($keys[$i],$vals[$i]) if $vals[$i];
      }
      return;});#$
⟨/ltxml⟩
```

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

```
⟨*package⟩
\def\record@problem{\protected@write\@auxout{}%
{\string\@problem{\prob@number}%
{\ifx\inclprob@pts\@empty\problem@pts\else\inclprob@pts\fi}%
{\ifx\inclprob@min\@empty\problem@min\else\inclprob@min\fi}}}
⟨/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).

```
⟨*package⟩
\def\@problem#1#2#3{}
⟨/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.

```
⟨*package⟩
\srefaddidkey{soln}
\addmetakey{soln}{for}
\addmetakey{soln}{height}
\addmetakey{soln}{creators}
\addmetakey{soln}{contributors}
\addmetakey{soln}{srccite}
%    \begin{macrocode}
% the next step is to define a helper macro that does what is needed to start a solution.
%    \begin{macrocode}
\newcommand\@startsolution[1][]{\metasetkeys{soln}{#1}%
\@in@omtexttrue% we are in a statement.
\ifboxed\else\hrule\fi\smallskip\noindent{\textbf\prob@solution@kw: }\begin{small}%
\def\current@section@level{\prob@solution@kw}%
```

```
\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.

```
\newcommand\startsolutions{\specialcomment{solution}{\@startsolution}%
{\ifboxed\else\hrule\medskip\fi\end{small}}%
\ifboxed\surroundwithmdframed{solution}\fi}
⟨/package⟩
⟨*ltxml⟩
DefConstructor('\startsolutions','');
⟨/ltxml⟩
```

\stopsolutions

```
⟨*package⟩
\newcommand\stopsolutions{\excludecomment{solution}}
⟨/package⟩
⟨*ltxml⟩
DefConstructor('\stopsolutions','');
⟨/ltxml⟩
```

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

```
⟨*package⟩
\ifsolutions\startsolutions\else\stopsolutions\fi
⟨/package⟩
```

the LaTeXML binding for the solutions is straightforward.

```
⟨*ltxml⟩
DefKeyVal('soln','id','Semiverbatim');
DefKeyVal('soln','height','Semiverbatim');
DefKeyVal('soln','for','Semiverbatim');
DefKeyVal('soln','creators','Semiverbatim');
DefKeyVal('soln','contributors','Semiverbatim');
DefEnvironment('{solution} OptionalKeyVals:soln',
       "<omdoc:solution ?&GetKeyVals(#1,'for')(for='&GetKeyVal(#1,'for')')()>"
     .   "#body"
     . "</omdoc:solution>");
⟨/ltxml⟩
```

```
⟨*package⟩
\ifexnotes
\newenvironment{exnote}[1][]%
{\par\smallskip\hrule\smallskip\noindent\textbf{Note: }\small}
{\smallskip\hrule}
\else%ifexnotes
\excludecomment{exnote}
\fi%ifexnotes
\ifhints
\newenvironment{hint}[1][]%
```

```
{\par\smallskip\hrule\smallskip\noindent\textbf{Hint: }\small}
{\smallskip\hrule}
\newenvironment{exhint}[1][]%
{\par\smallskip\hrule\smallskip\noindent\textbf{Hint: }\small}
{\smallskip\hrule}
\else%ifhints
\excludecomment{hint}
\excludecomment{exhint}
\fi%ifhints
⟨/package⟩
⟨*ltxml⟩
DefEnvironment('{exnote}',"<omdoc:hint>#body</omdoc:hint>");
DefEnvironment('{hint}',"<omdoc:hint>#body</omdoc:hint>");
DefConstructor('\pts{}',"");
DefConstructor('\min{}',"");
⟨/ltxml⟩
```

4.2 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.

```
⟨*package⟩
\addmetakey{inclprob}{pts}
\addmetakey{inclprob}{min}
\addmetakey*{inclprob}{title}
\addmetakey{inclprob}{refnum}
\addmetakey{inclprob}{mhrepos}
\clear@inclprob@keys%initially
\newcommand\includeproblem[2][]{\metasetkeys{inclprob}{#1}%
\input{#2}\clear@inclprob@keys}
⟨/package⟩
⟨*ltxml⟩
DefKeyVal('prob','pts','Semiverbatim');
DefKeyVal('prob','min','Semiverbatim');
DefKeyVal('prob','title','Semiverbatim');
DefKeyVal('prob','refnum','Semiverbatim');
DefConstructor('\includeproblem OptionalKeyVals:prob Semiverbatim',
   "<omdoc:exercise tref='#2'>"
.  "?&GetKeyVal(#1,'title')(<dc:title>&GetKeyVal(#1,'title')</dc:title>)()"
.  "?&GetKeyVal(#1,'min')("
 .     "<omdoc:meta property='prob:solvedinminutes' prob:dummy='for the namespace'>"
.       "&GetKeyVal(#1,'min')"
 .     "</omdoc:meta>)()"
.  "?&GetKeyVal(#1,'pts')("
 .     "<omdoc:meta property='prob:points' prob:dummy='for the namespace'>"
.       "&GetKeyVal(#1,'pts')"
 .     "</omdoc:meta>)()"
 ."</omdoc:exercise>",
```

```
 afterDigest => sub{
   my ($stomach,$kv) = @_;
   AssignValue('inclprob',$kv) if $kv;
 });
⟨/ltxml⟩
```

```
⟨*ltxml⟩
Tag('omdoc:exercise',afterOpen=>\&numberIt);
Tag('omdoc:solution',afterOpen=>\&numberIt);
Tag('omdoc:hint',afterOpen=>\&numberIt);
⟨/ltxml⟩
```

4.3 Reporting Metadata

```
⟨*package⟩
\def\pts#1{\ifpts\marginpar{#1 pt}\fi}
\def\min#1{\ifmin\marginpar{#1 min}\fi}
⟨/package⟩
⟨*ltxml⟩
⟨/ltxml⟩
```

```
⟨*package⟩
\AtEndDocument{\ifpts\message{Total: \arabic{pts} points}\fi
\ifmin\message{Total: \arabic{min} minutes}\fi}
⟨/package⟩
⟨*ltxml⟩
⟨/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.

```
⟨*package⟩
\newcounter{pts}
\def\show@pts{\ifx\inclprob@pts\@empty%
\ifx\problem@pts\@empty\else%
\ifpts\marginpar{\problem@pts pt\smallskip}\addtocounter{pts}{\problem@pts}\fi%
\fi\else% inclprob@pts nonempty
\ifpts\marginpar{\inclprob@pts pt\smallskip}\addtocounter{pts}{\inclprob@pts}\fi%
\fi}
```

and now the same for the minutes

`\show@min`

```
\newcounter{min}
\def\show@min{\ifx\inclprob@min\@empty%
\ifx\problem@min\@empty\else%
\ifmin\marginpar{\problem@min min}\addtocounter{min}{\problem@min}\fi%
\fi\else%
\ifmin\marginpar{\inclprob@min min}\addtocounter{min}{\inclprob@min}\fi
\fi}
⟨/package⟩
```

4.4 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.

```
⟨*package⟩
\newcommand\includemhproblem[2][]{\metasetkeys{inclprob}{#1}%
\edef\mh@@repos{\mh@currentrepos}%
\ifx\inclprob@mhrepos\@empty\else\mhcurrentrepos\inclprob@mhrepos\fi%
\input{\MathHub{\mh@currentrepos/source/#2}}%
\mhcurrentrepos\mh@@repos\clear@inclprob@keys}
⟨/package⟩
⟨*ltxml⟩
sub includemhproblem {
  my ($gullet,$keyval,$arg2) = @_;
  my $repo_path;
  if ($keyval) {
    $repo_path = ToString(GetKeyVal($keyval,'mhrepos')); }
  if (! $repo_path) {
    $repo_path = ToString(Digest(T_CS('\mh@currentrepos'))); }
  else {
    $keyval->setValue('mhrepos',undef); }
  my $mathhub_base = ToString(Digest('\MathHub{}'));
  my $finalpath = $mathhub_base.$repo_path.'/source/'.ToString($arg2);
  return Invocation(T_CS('\includeproblem'), $keyval, T_OTHER($finalpath)); }#$
DefKeyVal('inclprob','mhrepos','Semiverbatim');
DefMacro('\includemhproblem OptionalKeyVals:inclprob {}', \&includemhproblem);
⟨/ltxml⟩
```

4.5 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`.

```
⟨*ltxml⟩
Tag('omdoc:exercise',afterOpen=>\&numberIt,afterClose=>\&locateIt);
Tag('omdoc:solution',afterOpen=>\&numberIt,afterClose=>\&locateIt);
Tag('omdoc:hint',afterOpen=>\&numberIt,afterClose=>\&locateIt);
⟨/ltxml⟩
```

4.6 Finale

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

```
⟨ltxml⟩1;
```