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

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

March 20, 2019

Abstract

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

Contents

*Version v1.3 (last revised 2019/03/20)

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 [Koh16] 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.

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`

2.3 Multiple Choice Blocks

`mcb`
`\mcc`

Multiple choice blocks can be formatted using the `mcb` environment, in which single choices are marked up with `\mcc[`⟨*keyvals*⟩`]{`⟨*text*⟩`}` macro, which takes an optional key/value argument ⟨*keyvals*⟩ for choice metadata and a required argument ⟨*text*⟩ for the proposed answer text. The following keys are supported

`T`
`F`
`Ttext`
`Ftext`
`feedback`

- `T` for true answers, `F` for false ones,
- `Ttext` the verdict for true answers, `Ftext` for false ones, and
- `feedback` for a short feedback text given to the student.

See Figure **??** for an example

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.

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].

```
\begin{problem}[title=Functions]
  What is the keyword to introduce a function definition in python?
  \begin{mcb}
    \mcc[T]{def}
    \mcc[F,feedback=that is for C and C++]{function}
    \mcc[F,feedback=that is for Standard ML]{fun}
    \mcc[F,Ftext=Nooooooooo,feedback=that is for Java]{public static void}
  \end{mcb}
\end{problem}
```

Problem 2 (Functions)
What is the keyword to introduce a function definition in python?

1. def
2. function
3. fun
4. public static void

Problem 3 (Functions)
What is the keyword to introduce a function definition in python?

1. def
   Yes!
2. function
   No, that is for C and C++
3. fun
   No, that is for Standard ML
4. public static void
   Nooooooooo, that is for Java

Example 3: A Problem with a multiple choice block

1. none reported yet

4 The Implementation

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.

```
⟨*package⟩
\newif\if@problem@mh@\@problem@mh@false
\DeclareOption{mh}{\@problem@mh@true}
\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
```

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

```
\if@problem@mh@\RequirePackage{problem-mh}\fi
\RequirePackage{omtext}
\RequirePackage{comment}
\RequirePackage{mdframed}
\RequirePackage[base]{babel}
```

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

```
\AfterBabelLanguage{ngerman}{\input{problem-ngerman.ldf}}
\AfterBabelLanguage{arabic}{\input{problem-arabic.ldf}}
\def\prob@problem@kw{Problem}
\def\prob@solution@kw{Solution}
```

4.2 Problems and Solutions

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

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

Then we set up a counter for problems.

\numberproblemsin

```
\newcounter{problem}
\newcommand\numberproblemsin[1]{\@addtoreset{problem}{#1}}
```

\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

```
\newcommand\prob@number{%
\ifx\inclprob@refnum\@empty% if there is no outside refnumb
\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\rmfamily\noindent\ignorespaces}
{\smallskip}
\ifboxed\surroundwithmdframed{problem}\fi
```

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

```
\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}}}
```

`\@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).

```
\def\@problem#1#2#3{}
```

`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.

```
\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}
```

`\stopsolutions`

```
\newcommand\stopsolutions{\excludecomment{solution}}
```

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

```
\ifsolutions\startsolutions\else\stopsolutions\fi
```

```
\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
```

4.3 Multiple Choice Blocks

EdN:1

mcb [1]

```
\newenvironment{mcb}
{\begin{enumerate}}
{\end{enumerate}}
```

we define the keys for the mcc macro

```
\srefaddidkey{mcc}
\addmetakey{mcc}{feedback}
\addmetakey[T]{mcc}{T}
\addmetakey[F]{mcc}{F}
\addmetakey[Yes]{mcc}{Ttext}
\addmetakey[No]{mcc}{Ftext}
```

\mcc

```
\newcommand\mcc[2][]{%
\metasetkeys{mcc}{#1}%
\item #2%
\ifsolutions\\%
\ifcsstring{mcc@T}{T}{}{\mcc@Ttext}%
\ifcsstring{mcc@F}{F}{}{\mcc@Ftext}%
\ifx\mcc@feedback\@empty!\else, \mcc@feedback\fi%
\fi} %solutions
```

4.4 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.

```
\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}
```

[1] EdNote: MK: maybe import something better here from a dedicated MC package

4.5 Reporting Metadata

```
\def\pts#1{\ifpts\marginpar{#1 pt}\fi}
\def\min#1{\ifmin\marginpar{#1 min}\fi}

\AtEndDocument{\ifpts\message{Total: \arabic{pts} points}\fi
\ifmin\message{Total: \arabic{min} minutes}\fi}
```

`\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.

```
\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⟩

Change History