An Infrastructure for formatting Assignments*

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

July 19, 2010

Abstract

The `assignment` packge allows individual course assignment sheets and compond assignment documents using problem files marked up with the `problem` package.

Contents

*Version v0.9a (last revised 2010/06/25)

1 Introduction

The `assignment` package supplies an infrastructure that allows to format nice-looking assignment sheets by simply including problems from problem files marked up with the `problem` package. It is designed to be compatible with `problems.sty`, and inherits some of the functionality.

2 The User Interface

2.1 Assignments

`assignment` This package supplies the `assignment` environment that groups problems into assignment sheets. It takes an optional KeyVal argument with the keys `number` (for the assignment number; if none is given, 1 is assumed as the default or — in multi-assignment documents — the ordinal of the `assignment` environment), `title` (for the assignment title; this is referenced in the title of the assignment sheet), `type` (for the assignment type; e.g. "quiz", or "homework"), `given` (for the date the assignment was given), and `due` (for the date the assignment is due).

`number` `title` `type` `given` `due`

`\CourseTitle` `\Coursetitle{`⟨*title*⟩`}` can be used to specify the title of the course. It is usually specified in the driver file.

`\AssignmentType` Similarly, `\AssignmentType{`⟨*type*⟩`}` can be used to specify the default assignment type.

`solutions` The `assignment` package takes the options `solutions`, `notes`, `hints`, `pts`, `min`, and `boxed` that are just passed on to the `problems` package (cf. its documentation for a description of the intended behavior).

`notes` `hints` `pts` `min` `boxed`

2.2 Typesetting Exams

`multiple` Furthermore, the `assignment` package takes the option `multiple` that allows to combine multiple assigment sheets into a compound document (the assignment sheets are treated as section, there is a table of contents, etc.).

`test` Finally, there is the option `test` that modifies the behavior to facilitate formatting tests. Only in `test` mode, the macros `\testspace`, `\testnewpage`, and `\testemptypage` have an effect: they generate space for the students to solve the given problems. Thus they can be left in the LaTeX source.

`\testspace` `\testspace` takes an argument that expands to a dimension, and leaves vertical space accordingly. `\testnewpage` makes a new page in `test` mode, and `\testemptypage` generates an empty page with the cautionary message that this page was intentionally left empty.

`\testnewpage` `\testemptypage`

`\testheading` Finally, the `\testheading` takes a keyword argument where the keys `duation` specifies a string that specifies the duration of the test, `min` specifies the equivalent in number of minutes, and `reqpts` the points that are required for a perfect grade.

```
\testheading{duration=one hour,min=60,reqpts=27}
```

formats to

NAME:

MATRICULATION NUMBER:

You have one hour(sharp) for the test;
Write the solutions to the sheet.
The scheduled time for solving this exam is 58 minutes, leaving you 2 minutes for revising your exam.
You can reach 30 points if you solve all problems. You will only need 27 points for a perfect score, i.e. 3 points are bonus points.

Different problems test different skills and knowledge, so do not get stuck on one problem.

	To be used for grading, do not write here								
prob.	1.1	2.1	2.2	2.3	3.1	3.2	3.3	Sum	grade
total	4	4	6	6	4	4	2	30	
reached									

Example 1: A generated test heading.

2.3 Including Assignments

`\includeassignment`

The `\includeassignment` macro can be used to include an assignment 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 `assignment` environment in the included file). The keys `number`, `title`, `type`,`given`, and `due` are just as for the `assignment` environment and (if given) overwrite the ones specified in the `assignment` environment in the included file.

`number`
`title`
`type`
`given`
`due`

3 The Implementation

3.1 Package Options

The first step is to declare (a few) package options that handle whether certain information is printed or not. Some come with their own conditionals that are set by the options, the rest is just passed on to the `problems` package.

```
⟨*package⟩
\newif\iftest\testfalse
\newif\ifsolutions\solutionsfalse
\DeclareOption{test}{\testtrue\solutionsfalse}
\newif\ifmultiple\multiplefalse
\DeclareOption{multiple}{\multipletrue}
\DeclareOption*{\PassOptionsToPackage{\CurrentOption}{problem}}
\ProcessOptions
⟨/package⟩
```

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

```
⟨*package⟩
\RequirePackage{keyval}[1997/11/10]
\RequirePackage{problem}
⟨/package⟩
```

Here comes the equivalent header information for LaTeXML, we also initialize the package inclusions. Since LaTeXML does not handle options yet, we have nothing to do.

```
⟨*ltxml⟩
# -*- CPERL -*-
package LaTeXML::Package::Pool;
use strict;
use LaTeXML::Package;
RequirePackage('problem');
⟨/ltxml⟩
```

Then we register the namespace of the requirements ontology

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

3.2 Assignments

We will prepare the keyval support for the `assignment` environment.

```
⟨*package⟩
\omdaddkey{assig}{number}
\omdaddkey{assig}{title}
\omdaddkey{assig}{type}
\omdaddkey{assig}{given}
\omdaddkey{assig}{due}
⟨/package⟩
```

The next macro just sets the internal token register to the intended string.

```
⟨*package⟩
\def\AssignmentType#1{\gdef\assig@default@type{#1}}
\def\assig@default@type{Assignment}
⟨/package⟩
⟨*ltxml⟩
⟨/ltxml⟩
```

The next three macros are intermediate functions that handle the case gracefully, where the respective token registers are undefined.

```
⟨*package⟩
\def\Assig@Type{\ifx\assig@type\@empty\assig@default@type\else\assig@type\fi}
\def\Assig@Title{\ifx\assig@title\@empty\else\assig@title~\fi}
```

The `\given@due` macro prints information about the given and due status of the assignment. Its arguments specify the brackets.

```
\def\given@due#1#2{%
\ifx\assig@given\@empty\else\ifx\assig@due\@empty\else{#1}\fi\fi%
\ifx\assig@given\@empty\else{Given {\assig@given}}\fi%
\ifx\assig@given\@empty\else\ifx\assig@due\@empty\else{, }\fi\fi%
\ifx\assig@due\@empty\else{Due {\assig@due}}\fi%
\ifx\assig@given\@empty\else{\ifx\assig@due\@empty\else{#2}\fi}\fi}
⟨/package⟩
```

With them, we can define the central `assignment` environment. This has two forms (separated by `\ifmultiple`) in one we make a title block for an assignment sheet, and in the other we make a section heading and add it to the table of contents.

`assignment`

```
⟨*package⟩
\newenvironment{assignment}[1][]% keyval args
{\omdsetkeys{assig}{#1}% collect the keys and correct them from the outside
\@ifundefined{incl@assig@title}{}{\def\assig@title{\incl@assig@title}}
\@ifundefined{incl@assig@type}{}{\def\assig@type{\incl@assig@type}}
\@ifundefined{incl@assig@num}{}{\def\assig@num{\incl@assig@num}}
\@ifundefined{incl@assig@due}{}{\def\assig@due{\incl@assig@due}}
\@ifundefined{incl@assig@given}{}{\def\assig@given{\incl@assig@given}}
\@ifundefined{assig@num}{\stepcounter{section}}%
{\setcounter{section}{\assig@num}\setcounter{problem}{0}}
\ifmultiple%
\section*{\Assig@Type~\arabic{section}:~\Assig@Title\given@due{\\(}{)}}
  \addcontentsline{toc}{section}{\Assig@Type~{\arabic{section}}:~\Assig@Title}
\else
\begin{center}
\Large\Course@Title\\
{\bf{\Assig@Type}~{\arabic{section}}:~\Assig@Title\strut\\\large{\given@due()}}
\end{center}
\fi%ifmultiple
```

```
}{}
⟨/package⟩
⟨*ltxml⟩
DefEnvironment('{assignment} OptionalKeyVals:assig',
  "<omdoc:omgroup ?&KeyVal(#1,'id')(xml:id='&KeyVal(#1,'id')')() "
  .  "assig:dummy='for the namespace'"
  . "?#locator(stex:srcref='#locator')()>"
  .  "<omdoc:metadata ?#locator(stex:srcref='#locator')()>"
  .    "<dc:title ?#locator(stex:srcref='#locator')()>"
  .       "Assignment ?&KeyVal(#1,'num')(&KeyVal(#1,'num').)()"
  .       "?&KeyVal(#1,'title')((&KeyVal(#1,'title')))"
  .    "</dc:title>"
  .    "?&KeyVal(#1,'given')(<omdoc:meta property='assig:given'>&KeyVal(#1,'given')</omdoc:meta
  .    "?&KeyVal(#1,'due')(<omdoc:meta property='assig:due'>&KeyVal(#1,'due')</omdoc:meta>)()"
  .    "?&KeyVal(#1,'pts')(<omdoc:meta property='assig:pts'>&KeyVal(#1,'pts')</omdoc:meta>)()"
  .  "</omdoc:metadata>"
  .  "#body"
  ."</omdoc:omgroup>\n");
⟨/ltxml⟩
```

3.3 Including Assignments

The next command is essentially a glorified `\include` statement, it just sets some internal macros first that overwrite the local points,

```
⟨*package⟩
\define@key{incl@assig}{number}{\def\incl@assig@num{#1}}
\define@key{incl@assig}{title}{\def\incl@assig@title{#1}}
\define@key{incl@assig}{type}{\def\incl@assig@type{#1}}
\define@key{incl@assig}{given}{\def\incl@assig@given{#1}}
\define@key{incl@assig}{due}{\def\incl@assig@due{#1}}
\newcommand{\includeassignment}[2][]{\bgroup\setkeys{incl@assig}{#1}\include{#2}\egroup}
⟨/package⟩
⟨*ltxml⟩
DefMacro('\includeassignment [] {}','\input{#2}');
⟨/ltxml⟩
⟨*package⟩
\def\CourseTitle#1{\gdef\Course@Title{#1}}
⟨/package⟩
⟨*ltxml⟩
DefConstructor('\CourseTitle{}','');
⟨/ltxml⟩
```

3.4 Typesetting Exams

```
⟨*package⟩
\omdaddkey{testheading}{min}
\omdaddkey{testheading}{duration}
\omdaddkey{testheading}{reqpts}
```

```
\def\testheading#1{\omdsetkeys{testheading}{#1}
{\noindent\large NAME: \\[1ex] MATRICULATION NUMBER:\\[2ex]
 {\textbf{You have
\ifx\test@heading@duration\@empty\testheading@min minutes\else\testheading@duration\fi
(sharp) for the test}};\\ Write the solutions to the sheet.}\par\noindent

\newcount\check@time\check@time=\testheading@min
\advance\check@time by -\theassignment@totalmin
The scheduled time for solving this exam is {\theassignment@totalmin} minutes,
leaving you {\the\check@time} minutes for revising your exam.

\newcount\bonus@pts\bonus@pts=\theassignment@totalpts
\advance\bonus@pts by -\testheading@reqpts
You can reach {\theassignment@totalpts} points if you solve all problems. You will only need
{\testheading@reqpts} points for a perfect score, i.e.\ {\the\bonus@pts} points are
bonus points. \vfill
\begin{center}
  {\Large\em
%  You have ample time, so take it slow and avoid rushing to mistakes!\\[2ex]
  Different problems test different skills and knowledge, so do not get stuck on
  one problem.}\vfill\par\correction@table
\end{center}\newpage}
⟨/package⟩
⟨*ltxml⟩
DefConstructor('\testheading{}','');
⟨/ltxml⟩
⟨*package⟩
\def\testspace#1{\iftest\vspace*{#1}\fi}
\def\testnewpage{\iftest\newpage\fi}
\def\testemptypage{\iftest\begin{center}This page was intentionally left
    blank for extra space\end{center}\vfill\eject\else\fi}
⟨/package⟩
⟨*ltxml⟩
DefConstructor('\testspace{}','');
DefConstructor('\testnewpage','');
DefConstructor('\testemptypage','');
⟨/ltxml⟩
```

`\@problem` This macro acts on a problem's record in the `*.aux` file. Here we redefine it to generate the correction table.

```
⟨*package⟩
\def\@problem#1#2#3{\stepcounter{assignment@probs}
\def\@test{#2}\ifx\@test\@empty\else\addtocounter{assignment@totalpts}{#2}\fi
\def\@test{#3}\ifx\@test\@empty\else\addtocounter{assignment@totalmin}{#3}\fi
\xdef\correction@probs{\correction@probs & #1}%
\xdef\correction@pts{\correction@pts & #2}
\xdef\correction@reached{\correction@reached &}}
⟨/package⟩
```

`\correction@table` This macro generates the correction table

```
⟨*package⟩
\newcounter{assignment@probs}
\newcounter{assignment@totalpts}
\newcounter{assignment@totalmin}
\def\correction@probs{prob.}%
\def\correction@pts{total}%
\def\correction@reached{reached}%
\stepcounter{assignment@probs}
\def\correction@table{\begin{tabular}{|l|*{\theassignment@probs}{c|}|p{3cm}|}\hline%
&\multicolumn{\theassignment@probs}{c||}%
{\footnotesize To be used for grading, do not write here} &\\\hline
\correction@probs & Sum & grade\\\hline
\correction@pts &\theassignment@totalpts & \strut\hspace{3cm}\strut\\\hline
\correction@reached & & \\[.7cm]\hline
\end{tabular}}
⟨/package⟩
```

3.5 Leftovers

at some point, we may want to reactivate the logos font, then we use

```
here we define the logos that characterize the assignment
\font\bierfont=../assignments/bierglas
\font\denkerfont=../assignments/denker
\font\uhrfont=../assignments/uhr
\font\warnschildfont=../assignments/achtung

\def\bierglas{{\bierfont\char65}}
\def\denker{{\denkerfont\char65}}
\def\uhr{{\uhrfont\char65}}
\def\warnschild{{\warnschildfont\char 65}}
\def\hardA{\warnschild}
\def\longA{\uhr}
\def\thinkA{\denker}
\def\discussA{\bierglas}
```

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

```
⟨ltxml⟩1;
```