The pmdb Package

D. P. Story
Email: dpstory@acrotex.net

processed December 27, 2019

Contents

1 ⟨*package⟩

1 Introduction

This package addresses the issue of a poor-man's database. Educators who use LaTeX to construct exams and homework sometimes have a collection of problems. Each problem is in its own TEX file. The educator creates a document and \inputs several of these packaged questions. This package attempts to provide some "user interface" to the questions and provides a mechanism of selecting which questions are to be included in the document.

What does this package do? For a document that inputs content using the LaTeX command \input, the same content can be input using the command \pmInput (capitalized). When content is input by \pmInput, a check box is created in the margin at the insertion point of the content. The check boxes so created can be checked or cleared. When the user clicks on push button provided

by this package, a list of all *selected* `\input` statements are listed in the JavaScript console. This list can then be copied and pasted into another document the author is developing. If you Ctrl+Click on a check box, the associated content is opened in the default TEX editor. In this way, the document author can see a typeset of the content and decide whether the content should be included in the developing document.

The DB stage: When you have a collection of questions (or content) in various files and want to use this package to `\input` them into your document, the following comments are apropos:

PDF creators: Any PDF creator current in the LaTeX world is valid for use with this package.

PDF viewers: The ideal viewer is Acrobat, however, Adobe Reader and PDF-XChange Editor can also be used. In the case of Adobe Reader, there is an annoying security dialog box that appears each time you use the Ctrl+Click feature of the check box; the Ctrl+Click feature does not WORK with PDF-XChange Editor.

The production stage: After the document has been assembled (using pmdb), and you build your final document (perhaps an exerquiz quiz), the end user can use an appropriate PDF reader. If an exerquiz quiz is used, then a minimum of Adobe Acrobat Reader is required.

2 The main code

```
\edef\th@dquoteCat{\the\catcode`\"}
\catcode`\"=12\relax
\newif\ifpmdbmode \pmdbmodetrue
```

2.1 Package options and package requirements

`dbmode` The default option is `dbmode`. When in effect, check boxes appear in the margins at each `\pmInput` point.

```
\DeclareOption{dbmode}{\pmdbmodetrue}
```

`!dbmode` A convenient way to turn off the creation of the check boxes is to simply place an exclamation point (!) in front of the `dbmode` option.

```
\DeclareOption{!dbmode}{\pmdbmodefalse}
```

`tight` When this option is taken, the checkboxes are tight against the text box area.

```
\newif\ifpmdbtight \pmdbtightfalse
\DeclareOption{tight}{\pmdbtighttrue}
```

`!tight` The default for the package, the checkboxes are placed to extreme left (or right) in the margins.

```
\DeclareOption{!tight}{\pmdbtightfalse}
\ProcessOptions
\RequirePackage{eforms}
```

2.2 Special attention to Thor

One motivation for this package is to support the thorshammer package, to that end we make the following assignment, if Thor is not present. This is to prevent stoppage: if your are inputting a `\RespBoxEssay` question that is accompanied by the `\essayQ` command, defined in thorshammer.

```
\def\pmdb@ckThor{\@ifundefined{essayQ}{\let\essayQ\@gobble}{}}
\AtBeginDocument{\pmdb@ckThor}
```

2.3 Form field creation

Some Booleans and counters

```
\newif\ifpmdbFP \pmdbFPfalse
\newif\ifpmdbDQs \pmdbDQsfalse
\newcount\pmdb@Cnt
```

2.3.1 Check box creation

`\cbSelectInput{`⟨*path*⟩`}` creates a check box with a tool tip of ⟨*path*⟩ The mouse up action `\ccBoxMU` fixes up relative paths, and defines a Ctrl+Click action. When the check box is so clicked, the ⟨*path*⟩ is opened in the default TEX editor. The ⟨*path*⟩ can be relative or absolute. This command is used within `\insertCkBx`; its ⟨*path*⟩ argument is passed to it from `\insertCkBx`.

```
\def\pmCBPresets#1{\def\pm@CBPresets{#1}}
\pmCBPresets{}
\def\cbSelectInput#1{\checkBox[\TU{#1}\presets{\pm@CBPresets}
    \cmd{\bParams{#1}{\the\pmdb@Cnt}\eParams}
    \AAmouseup{\ccBoxMU}
  ]{pmdbCkBx.\the\pmdb@Cnt}{11bp}{11bp}{On}%
  \global\advance\pmdb@Cnt\@ne}
```

`\insertCkBx{`⟨*method*⟩`}` The argument of this macro describes the method of inserting the check-box. The default definition works well for a straightforward document, where you are inputting ordinary LaTeX code (such as sections or chapters).

```
\def\insertCkBx#1{\def\@insertCkBx##1{#1}}
```

`\pmAlignCB` `\normalCBMargins` `\altCBMargins` Placement of check boxes in the margin. `\pmAlignCB` controls the marginpar placement. `\normalCBMargins` places it according to the rules of `\marginpar`; `\altCBMargins` alternates the margin placement, forces the check box to the extreme left (on odd pages) and extreme right (on even pages).

```
\def\setCBsMarg{%
  \ifpmdbtight
    \if@reversemargin
      \def\pmAlignCBAlt{\ifodd\value{page}\leavevmode\hfill\else\fi}\else
      \def\pmAlignCBAlt{\ifodd\value{page}\else\hfill\fi}\fi
  \else
    \if@reversemargin
      \def\pmAlignCBAlt{\ifodd\value{page}\hfil\else\hfil\fi}\else
```

```
      \def\pmAlignCBAlt{\ifodd\value{page}\hfill\else\fi}\fi
  \fi
}
\def\altCBMargins{\let\pmAlignCB\pmAlignCBAlt}
\def\pmAlignCB{%
  \if@reversemargin
    \ifpmdbtight\hfill\else\fi
  \else
    \ifpmdbtight\else\hfill\fi
  \fi
}
\@ifundefined{chapter}{}{\AtBeginDocument{\setCBsMarg\altCBMargins}}
```

This is the default declaration. It works well when you are inputting content that goes into horizontal mode. We insert the check box at the beginning of the first paragraph. When you are inputting files that come into a list environment, this method does not work satisfactory.

`\InputParas` declares that the next `\pmInput` macros are for paragraph content. This is the default state of the package.

pmInput states

```
\def\InputParas{%\let\pmAlignCB\relax
  \insertCkBx{\ifpmdbmode
  \everypar{\marginpar{\pmAlignCB\cbSelectInput{##1}}\global\everypar{}}\fi}}
\InputParas
```

`\InputQuizItems` declares that the next `\pmInput` macros are for items in a `quiz` environment of exerquiz

```
\newcount\saveQNo \saveQNo\z@
\def\pmHook@qzItems{%
  \let\item@pmOld\item
  \def\item@pnNew{\item@pmOld\itemhook\let\item\item@pmOld}%
  \let\item\item@pnNew}
\def\InputQuizItems{\let\pmHook\pmHook@qzItems
  \saveQNo\z@
  \insertCkBx{\def\cbInQzMargin{\cbSelectInput{##1}}}%
  \ItemHook{\leavevmode\ifpmdbmode
  \ifnum\saveQNo<\value{eqquestionnoi}%
    \marginpar{\pmAlignCB\cbInQzMargin}\fi
  \saveQNo=\arabic{eqquestionnoi}\fi}}
```

`\InputItems` declares that the next `\pmInput` macros are for items in an list environment 2019/12/09 v0.4

```
\def\pmHook@item{\let\item@pmOld\item
  \def\item@pmNew{%
    \ifx\pmiarg\@empty
      \ifx\pm@Brk\ef@YES
        \def\pm@next{\item@pmOld[]}\else
        \let\pm@next\item@pmOld
      \fi
```

```
    \else
      \def\pm@next{\item@pmOld[\pmiarg]}%
    \fi\pm@next\itemhook\let\item\item@pmOld}%
  \let\item\item@pmNew
}
\def\ItemHook#1{\def\itemhook{#1}}
\def\InputItems{\let\pmHook\pmHook@item
  \insertCkBx{\def\cbInQzMargin{\cbSelectInput{##1}}}%
  \ItemHook{\leavevmode\ifpmdbmode
  \marginpar{\pmAlignCB\cbInQzMargin}\fi}}
```

Place check box and input ⟨*path*⟩

`\ckBxInput{`⟨*path*⟩`}` Places the check box and inputs the ⟨*path*⟩.

```
\let\pmHook\relax
\def\ckBxInput#1{\@insertCkBx{#1}%
  \ifpmdbDQs\def\donext{\pmHook\input{"#1"}}\else
    \def\donext{\pmHook\input{#1}}\fi
  \donext}
```

2.3.2 Push button creation

This package provides two form fields that are used for the DB step.

`\displayChoices[`⟨*options*⟩`]{`⟨*wd*⟩`}{`⟨*ht*⟩`}` inserts a push button whose action is to display the selections in the console window. The argument ⟨*wd*⟩ can be empty, in which case, the width of the field is determined from the `\CA` key.

```
\def\displayChoiceCA#1{\def\displayChoice@CA{#1}}
\def\displayChoiceTU#1{\def\displayChoice@TU{#1}}
\displayChoiceCA{Display Choices}
\displayChoiceTU{Lists all choices in the console window}
\newcommand{\displayChoices}[3][]{\pushButton[\TU{\displayChoice@TU}
  \CA{\displayChoice@CA}#1\AAmouseup{\sldInputs}\protect\AA
]{sldInputs}{#2}{#3}}
```

`\clrChoices[`⟨*options*⟩`]{`⟨*wd*⟩`}{`⟨*ht*⟩`}` inserts a push button whose action is to clear all check boxes (and underlying JavaScript variables) created by this package. The argument ⟨*wd*⟩ can be empty, in which case, the width of the field is determined from the `\CA` key.

```
\def\clrChoicesCA#1{\def\clrChoices@CA{#1}}
\def\clrChoicesTU#1{\def\clrChoices@TU{#1}}
\clrChoicesCA{Clear Choices}
\clrChoicesTU{Clears all check boxes created by pmdb}
\newcommand{\clrChoices}[3][]{\pushButton[\TU{\clrChoices@TU}
  \CA{\clrChoices@CA}#1\AAmouseup{\clrAction}\protect\AA
]{sldInputs}{#2}{#3}}
```

2.4 Defining the \pmInput command

`\pmInput[`⟨*arg*⟩`]{`⟨*path*⟩`}` is the main user-interface for inputting a file; here, the macros name

is \pmInput, ultimately it calls \input with the same path. Paths with spaces must be enclosed in double quotes (\pmInput{my cool problem.tex}) and the extensions must always be used.

```
\def\pmInput{\@ifnextchar[%]
  {\let\pm@Brk\ef@YES\inputConta}
  {\let\pm@Brk\ef@NO\inputConta}}
\let\pm@Brk\ef@NO
\def\inputConta{\bgroup\@makeother\"\inputContb}
\newcommand\inputContb[2][]{\egroup\def\pmiarg{#1}\inputConti#2;;}
```

Determine if double quotation marks are used.

```
\def\inputConti{\@ifnextchar"%
  {\global\pmdbDQstrue\removedqs}
  {\global\pmdbDQsfalse\removesemis}}
\def\removedqs"#1";;{\inputContii{#1}}
\def\removesemis#1;;{\inputContii{#1}}
```

Determine if this is a full path, we do this by searching for a colon (:). Following the search for the colon, pass on to the final step of \doinput.

```
\def\inputContii#1{\isItFullPath#1:\@nil\doinput{#1}}
```

A command to detect presence of a colon.

```
\def\isItFullPath#1:#2\@nil{%
  \def\@rgii{#2}\ifx\@rgii\@empty
    \global\pmdbFPfalse\else
    \global\pmdbFPtrue\fi}
```

Final step, if the switch \ifpmdbmode is true, we insert the check box \ckBxInput; otherwise, we pass ⟨*path*⟩ to \input.

```
\def\doinput#1{\ifpmdbmode\def\donext{\ckBxInput{#1}}\else
  \ifpmdbDQs\def\donext{\input{"#1"}}\else
    \def\donext{\input{#1}}\fi\fi
  \donext}
```

During the development of this package, the original command name used was \Input. There are a few users that use this old definition; the command \Input is defined in other package, in particular in the srcltx. So we allow the use of \Input if \Input is not otherwise defined.

```
\def\pmInputWarni{\PackageWarningNoLine{pmdb}{The command \string\Input\space
  is already defined.\MessageBreak
  The checkboxes may not appear in the margins.\MessageBreak
  Use the supported command \string\pmInput\space instead}}
\def\pmInputWarnii{\PackageWarningNoLine{pmdb}{Letting
  \string\Input\space to \string\pmInput. You are \MessageBreak
  encouraged to use the supported\MessageBreak
  command \string\pmInput\space instead}}
\def\pmInputChk{\@ifundefined{Input}{\let\Input\pmInput\pmInputWarnii}
  {\pmInputWarni}}
\AtBeginDocument{\pmInputChk}
```

3 Field JavaScript

\ccBoxMU This is the JavaScript action of the check box, used in \cbSelectInput

```
\begin{defineJS}[\makeesc\@\catcode`\%=14\relax]{\ccBoxMU}
@ifpmdbFP%
event.target.userName=("@p(1)");
@else%
```

This part of the code is Windows specific. Don't know enough about Mac OS to form the proper path.

```
// device independent path
var pos=this.path.lastIndexOf("/");
var thispath=this.path.substring(0,pos+1);
```

/<drive>/user/documents/.../myfolder/

```
pos=this.path.indexOf("/",1);
var drive=thispath.substring(0,pos);
var platform=app.platform;
if (platform=="WIN")
```

/<drive>:/user/documents/.../myfolder/

```
  thispath=drive+":/"+thispath.substring(pos+1);
event.target.userName=thispath+("@p(1)");
@fi%
if (event.modifier){
  var _to=app.setTimeOut("_restoreCCState('pmdbCkBx.@p(2)');",.05);
  try {
  aebTrustedFunctions(this,aebLaunchURL,
    {cURL: "file://"+event.target.userName});
  } catch(e) {
    console.show();
    console.println("The Ctrl+Click action is not supported, %
installation of aeb\_pro.js is required.");
  }
} else {
  if (event.target.isBoxChecked(0)){
    _oSPaths["pmdbCkBx.@p(2)"]=%
[("@p(1)"),@ifpmdbDQs true@else false@fi];
    _aInputs[@p(2)]=true;
    _numInputs++;
  }else{
    _oSPaths["pmdbCkBx.@p(2)"]=null;
    _aInputs[@p(2)]=false;
    _numInputs--;
  }
  event.target["_boxState"]=!!event.target.isBoxChecked(0);
}
\end{defineJS}
```

\sldInputs This is the mouse up action for a push button. It lists all selected content and displays them in the console window of Acrobat/Reader, used in \displayChoices.

```
\begin{defineJS}[\catcode`\%=14\relax]{\sldInputs}
console.clear();console.show();
if (_numInputs==0) console.println("No inputs selected");
else {
  for(var i=0;i<_aInputs.length;i++){
    if (!!_aInputs[i]){
      if(_oSPaths["pmdbCkBx."+i][1])
        console.println(%
'\\\\input\{\\"'+(_oSPaths["pmdbCkBx."+i][0])+'\\"\}');
      else
        console.println(%
'\\\\input\{'+(_oSPaths["pmdbCkBx."+i][0])+'\}');
    }
  }
}
\end{defineJS}
```

`\clrAction` Mouse up action to clear the check boxes and to re-initialize internal internal JS variables. Used in `\clrChoices`.

```
\begin{defineJS}{\clrAction}
var _oSPaths=new Object;
var _aInputs=new Array;
var _numInputs=0;
this.resetForm("pmdbCkBx");
\end{defineJS}
```

4 Document JavaScript

```
\begin{insDLJS}{mrki}{Supporting JavaScript for pmdb}
var _oSPaths=new Object;
var _aInputs=new Array;
var _numInputs=0;
function _restoreCCState(fName){
  var f=this.getField(fName);
  var _ccState=%
(typeof f["_boxState"]=="undefined")?false:f["_boxState"];
  f["_boxState"]=_ccState;
  return f.checkThisBox(0,_ccState);
}
\end{insDLJS}
```

```
\catcode`\"=\th@dquoteCat
```

⟨/package⟩

5 Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.

6 Change History