The dps Package

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

processed June 9, 2020

Contents

1 ⟨∗package⟩

1 Introduction

The dps Package (Das Puzzle Spiel) provides the commands to create a matching game and associated puzzle. As the user answers each question, another part of the puzzle is filled in. After the user has correctly answered all the questions, the message contained in the puzzle is fully visible. The user answers the question by first clicking the checkbox of that question, reading and solving the question, then by finding the correct answer listed amongst the answer columns. This game was inspired by one of the handout work sheets of my son's eighth grade pre-algebra class.

2 Options

We bring in the xkeyval Package so we can gather our options using it's commands, rather than the default keyval commands.

2 `\RequirePackage{xkeyval}`

`nonrandomized` The default behavior is to randomize the questions and answers. With this option, the questions and answers are listed in the order they were written in the source file; good for debugging, and testing the document. We also have the option `!nonrandomized` `!nonrandomized` to 'cancel' `nonrandomized`.

```
3 \DeclareOptionX{nonrandomized}{\werandomizefalse}
4 \DeclareOptionX{!nonrandomized}{\werandomizetrue}
```

`viewmode` Used for developing the puzzle. When `viewmode` is optioned, the puzzle and letters in the puzzle are seen. By adjusting the argument of `\insertPuzzle` you can get the distribution of the puzzle that you want. See also the comments preceding the `!viewmode` definition of `\makeTextField` below. The `!viewmode` option cancels the `viewmode` option.

```
5 \DeclareOptionX{viewmode}{\viewModetrue\previewtrue}
6 \DeclareOptionX{!viewmode}{\viewModefalse\previewfalse}
```

`showletters` When this option is taken, in the answer columns, the letters that the answers correspond to appear; and in the puzzle, the question number that corresponds to that letter. Generally, this is used when web is put in `forpaper` mode, but can be `!showletters` used in "screen" mode. The convenience option of `!showletters` is also provided.

```
7 \DeclareOptionX{showletters}{\showletterstrue}
8 \DeclareOptionX{!showletters}{\showlettersfalse}
```

showanswerlabels

We give the user the option of showing the label for the answer, in the case of for screen presentation.

```
\DeclareOptionX{showanswerlabels}
{%
    \ifeqforpaper\else\def\dpsAitemOptArg{}\fi
}
\DeclareOptionX{!showanswerlabels}{%
  \def\dpsAitemOptArg{[]}}
\def\dpsAitemOptArg{[]}
```

showanswerkey

!showanswerkey

When this option is taken, the solution key appears in the footer. If the graphicx package is loaded, the answer key is rotated 180 degrees. The answer key is always computed and saved in the macro `\AnswerKey`. Selecting `showanswerkey` also activates the `showletters` option. The convenience option `!showanswerkey` is also provided.

```
\DeclareOptionX{showanswerkey}{\showsolutiontrue
  \ExecuteOptionsX{showletters}}
\DeclareOptionX{!showanswerkey}{\showsolutionfalse}
```

Options for posing questions. There are two methods of posing longer questions: (1) (`usebtnappr`) Place these questions in the appearance of a button; or (2) (`uselayers`) Place the questions in their own layer (OCG). Only one of these two options is allowed. Neither option is also permitted (short questions only).

usebtnappr

A required option when you want to pose longer questions. We have a complex workflow for placing the questions as a button appearance. This option brings in supporting commands.

```
\DeclareOptionX{usebtnappr}{\usebtnapprtrue
  \ifuseocgappr
    \PackageWarningNoLine{dps}
      {Options usebtnappr and uselayers both used.\MessageBreak
    Will use the uselayers option}%
    \usebtnapprfalse
    \let\dpsInputBtnAppr\relax
  \else
    \def\dpsInputBtnAppr{\InputIfFileExists{usebtnappr.def}{}{}}%
  \fi}
\let\dpsInputBtnAppr\relax
```

uselayers

A required option when you want to pose longer questions. Longer questions are placed in OCG (layers). This option brings in supporting commands.

```
\DeclareOptionX{uselayers}{\useocgapprtrue
  \ifusebtnappr
    \PackageWarningNoLine{dps}
      {Options usebtnappr and uselayers both used.\MessageBreak
      Will use the usebtnappr option}%
      \useocgapprfalse
      \let\dpsInputOcgAppr\relax
  \else
```

```
    \def\dpsInputOcgAppr{\InputIfFileExists{useocgappr.def}{}{}}%
  \fi}
\let\dpsInputOcgAppr\relax
```

savedata

When this option is taken, the solution key appears in the footer. If the graphicx package is loaded, the answer key is rotated 180 degrees. The answer key is always computed and saved in the macro `\AnswerKey`. The negation of `savedata`, `!savedata`, is also provided.

!savedata

```
\DeclareOptionX{savedata}{\savepuzzledatatrue}
\DeclareOptionX{!savedata}{\savepuzzledatafalse}
```

Various switches used by this package

```
\newif\ifwerandomize \werandomizetrue
\newif\ifviewMode \viewModefalse
\newif\ifshowletters\showlettersfalse
\newif\ifshowsolution\showsolutionfalse
\newif\ifsavepuzzledata\savepuzzledatafalse
\newif\ifusebtnappr \usebtnapprfalse
\newif\ifuseocgappr \useocgapprfalse
```

(20/06/03) New default is is `\wrtContenttrue`.

```
\newif\ifwrtContent\wrtContenttrue
```

If a paper option is taken, we show the letters.

```
\ifeqforpaper\showletterstrue\fi
```

lang

The only language localizations are the clever commands that appear in the message box. We offer two language options, `english` (the default) and `german`. There is a custom option for the author to provide his/her own language localizations.

```
\define@choicekey*+{dps.sty}{lang}[\val\nr]{english,german,custom}
{%
  \ifcase\nr\relax
    \def\dps@lang@type{\input{dps_str_us.def}}\or
    \def\dps@lang@type{\input{dps_str_de.def}}\or
    \def\dps@lang@type{\input{dps_str_cus.def}}\else
    \def\dps@lang@type{\input{dps_str_us.def}}\fi
}{\PackageWarning{dps}{Bad choice for lang, permissible values
are english, german and custom. Try again}}
\def\dps@lang@type{\input{dps_str_us.def}}
\AtEndOfPackage{\dps@lang@type}
\ProcessOptionsX
\edef\dps@restoreCats{%
  \catcode`\noexpand\"=\the\catcode`\"\relax
  \catcode`\noexpand\,=\the\catcode`\,\relax
  \catcode`\noexpand\_=\the\catcode`\_\relax
}
\@makeother\"\@makeother\,\@makeother\_
```

3 Required Packages

```
\RequirePackage{web}
\RequirePackage{eforms}
\ifxetex\makeXasPDOff\fi
\RequirePackage{graphicx}
```

In addition to the web and eforms packages, the following are used in the macro package.

```
\RequirePackage{verbatim}
\RequirePackage{calc}
\RequirePackage{multicol}
\RequirePackage{multido}
\hypersetup{pdfencoding=pdfdoc}
```

Input JavaScript for the `usebtnappr` option

```
\ifusebtnappr
  \def\x{\AtEndOfPackage{\dpsInputBtnAppr}}%
  \expandafter\x\fi
```

Input JavaScript for the `uselayers` option

```
\ifuseocgappr
  \def\x{\AtEndOfPackage{\dpsInputOcgAppr}}%
\expandafter\x\fi
```

For `usebtnappr`, we require icon-appr to embed the graphics, and make them appearances of buttons.

```
\ifusebtnappr
  \def\dps@RP{\RequirePackage{icon-appr}[2020/06/05]}
  \expandafter\dps@RP
\fi
```

(20/06/03) If the file `icons-pglst.sav` is present, that means the author has already compiled `icons.tex`, so we can set `\wrtContentfalse`. For the `usebtnappr` option: If `\ifwrtContent` is true (`icons.tex` has not produced the `icons-pglst.sav` file yet), we set `\savepuzzledatatrue`; otherwise, if `\ifwrtContent` is false, we set `\savepuzzledatafalse`.

```
\def\dps@ckForpglst{\IfFileExists{icons-pglst.sav}
  {\global\wrtContentfalse}{}
  \ifusebtnappr
    \ifwrtContent
      \global\savepuzzledatatrue
    \else
      \global\savepuzzledatafalse
    \fi
  \fi
}
\ifusebtnappr
\def\dps@emitEOP{\AtEndOfPackage{\dps@ckForpglst}}
\expandafter\dps@emitEOP\fi
```

Input random.tex. Input and make modifications.

```
\@ifundefined{nextrandom}{\input{random.tex}}{}
```

We modify \nextrandom to save the startup seed.

```
\def\dps@nextrandom{%
  \def\nextrandom{\begingroup
   \ifnum\randomi<\@ne % then initialize with time
      \global\randomi\time
      \global\multiply\randomi388 \global\advance\randomi\year
      \global\multiply\randomi31 \global\advance\randomi\day
      \global\multiply\randomi97 \global\advance\randomi\month
      \message{Randomizer initialized to \the\randomi.}%
      \nextrandom \nextrandom \nextrandom
      \xdef\ds@saveRandomSeed{\the\randomi}%
   \fi
   \count@ii\randomi
   \divide\count@ii 127773 % modulus = multiplier * 127773 + 2836
   \count@\count@ii
   \multiply\count@ii 127773
   \global\advance\randomi-\count@ii % random mod 127773
   \global\multiply\randomi 16807
   \multiply\count@ 2836
   \global\advance\randomi-\count@
   \ifnum\randomi<\z@ \global\advance\randomi 2147483647\relax\fi
   \endgroup
  }
}
\newif\ifnextrandomredefd\nextrandomredefdfalse
```

This package modifies \nextrandom from random.tex; however, other package, most notably, also use the random.tex macros and overwrite this definition of \nextrandom. To work around this problem, this package delays the redefinition of \nextrandom until is is first used in the preamble The command \redefnextrandomAsNeeded appears in the \ds@randomizeList, which is where all randomization occurs.

```
\def\redefnextrandomAsNeeded{\ifnextrandomredefd\else
  \global\let\nextrandom\dps@nextrandom
  \global\nextrandomredefdtrue\fi}
```

\useRandomSeed{⟨*pos-num*⟩} Use the number ⟨*pos-num*⟩ as the initial seed.

```
\def\useRandomSeed#1{\randomi=#1
  \def\ds@saveRandomSeed{#1}}
```

\inputRandomSeed With \inputRandomSeed, you input a seed value earlier saved with the option savedata. That way, you always get the same seed value as you move from the puzzle file to the icons file and back again.

```
\def\inputRandomSeed{\ifwerandomize
  \InputIfFileExists{\jobname_data.sav}{}{}%
  \edef\ds@saveRandomSeed{\the\randomi}\fi}
```

\useLastSeed \inputRandomSeed Input a last seed value that was available at the end of the last compile. (This assumes savedata is an active option.) If the SAV file does not exists, the seed used is based on the data and time.

```
\def\dpsLastSeed#1{\def\dps@LastSeed{#1}}
\def\useLastSeed{\ifwerandomize
  \InputIfFileExists{\jobname_data.sav}{}{}%
  \@ifundefined{dps@LastSeed}{}
    {\randomi=\dps@LastSeed\relax}%
  \edef\ds@saveRandomSeed{\the\randomi}\fi
}
```

4 Switches and things

```
\newif\ifforquestions \forquestionstrue
\newcount\ds@nCnt
\newcount\ds@nMax
\newcount\ds@qNumber\ds@qNumber=0
\newcount\ds@aNumber\ds@aNumber=0
\newcount\ds@probCnt \ds@probCnt=0
\newcount\ds@nCntCols \ds@nCntCols=0
\newcount\ds@getRanNum
\newtoks\ds@listIn \ds@listIn={}
\newtoks\ds@newListIn \ds@newListIn={}
\newtoks\ds@listOut \ds@listOut={}
\newtoks\ds@tmpToks \ds@tmpToks={}
\newtoks\ds@qlistOut \ds@qlistOut={}
\newtoks\ds@alistOut \ds@alistOut={}
\newtoks\ds@PuzzleAppearancetoks \ds@PuzzleAppearancetoks={}
\newtoks\ds@QuesAppearancetoks \ds@QuesAppearancetoks={}
\newtoks\ds@AnsAppearancetoks \ds@AnsAppearancetoks={}
\newwrite \ds@question@write
\setlength{\multicolsep}{\topsep}
\def\csarg#1#2{\expandafter#1\csname #2\endcsname}
\let\dps@One=1 \let\dps@Zero=0
```

5 Building the Puzzle

5.1 Declaring the puzzle

`\DeclarePuzzle{`⟨*puzzle-arg*⟩`}`, where ⟨*puzzle-arg*⟩ is a series of *pairs of arguments*.

```
\DeclarePuzzle{%
  {⟨letter1⟩}{⟨name1⟩}
  {⟨letter2⟩}{⟨name2⟩}
  ...
  ...
  {⟨lettern⟩}{⟨namen⟩}
}
```

The argument ⟨*letter*⟩ represents a letter in the puzzle; ⟨*letter*⟩ plays two roles: (1) it is used to typeset the letters into the document when certain options, such as `viewmode`, are used; (2) it is used as the default value of a text fields that is

created (when the puzzle is built to be interactive). This creates a problem for special characters, such as ü; on one hand the letter is `\"{u}` (when typeset), and is `\string\374` when placed into a text field (`\341`) is the (octal) PDFDocEncoding of u-umlaut. The way around this conundrum is to use `\texorpdfstring`: use ⟨*letter*⟩ to be `{\texorpdfstring{\"{u}}{\string\374}}`.

The second argument pair is ⟨*name*⟩, this is a unique name that is used in the construction of the underlying text field name: the name of the field becomes `puzzle.`⟨*name*⟩. As a result, ⟨*name*⟩ needs to be a JavaScript identifier (or, basically consist of letters and numbers). In the case of special characters such as our umlaut problem, we can assign a name like so:

```
\texorpdfstring{\protect\"{u}}{\ifxetex ü\else\string\374\fi}}{uml}
```

or

```
\tops{\protect\"{u}}{\ifxetex ü\else\string\374\fi}}{uml}
```

where `\tops` is an alias for `\texorpdfstring`. This argument pair is seen several times in the demonstration files. There are two special names, these are `space` and `punc`; as a argument pair, these should appear as follows: `{}{space}` and `{,}{punc}`, respectively. Spaces and punctuation are not normally part of the puzzle to be discovered by answering questions, though they could be.

```
\def\DeclarePuzzle#1{%
  \gdef\puzzleParameters{#1}%
  \let\DPSNamesList\@gobble
  \dps@getNames#1\relax\relax
}
\def\dps@getNames{\begingroup\dps@getNames@i}
\def\dps@getNames@i#1#2{%
  \ifx#2\relax\let\getNextN@me\endgroup
  \else\let\getNextN@me\dps@getNames@i
```

We skip adding to `\DPSNamesList` if `#2` is `space`, `punc`, `cr`, or `#2` has already been added.

```
    \def\@rgii{#2}\ifx\@rgii\ds@myspace\else
    \ifx\@rgii\ds@punc\else
    \ifx\@rgii\ds@cr\else
      \@ifundefined{ds@name@#2}{\g@addto@macro\DPSNamesList{,"#2"}%
        \csarg\let{ds@name@#2}\@empty}{}%
    \fi\fi\fi
  \fi
  \getNextN@me
}
```

`\nPuzzleCols{`⟨*nCols*⟩`}` As a convenience, we provide a way to pass the number of columns for the puzzle to the `\insertPuzzle{`⟨*nCols*⟩`}` command.

```
\def\nPuzzleCols#1{\def\nCols{#1}}
\let\nCols\@empty
```

5.2 The underlying text fields of the puzzle

`\PuzzleAppearance{`⟨*KV-pairs*⟩`}` The command `\PuzzleAppearance` can be used to change the appear-

ance of the text fields for the puzzle, where, ⟨*KV-pairs*⟩ is a set of eforms key-value pairs.

```
\def\PuzzleAppearance#1{\ds@PuzzleAppearancetoks={#1}}
```

`\ds@makeTextField{`⟨*letter*⟩`}{`⟨*name*⟩`}` The command acts on each pair of arguments of the command `\DeclarePuzzle`; it either creates a text field or an underlined space, depending on the options. Control the width of the fields with `\wdPuzzleFields{`⟨*width*⟩`}` and the height with `\htPuzzleFields{`⟨*height*⟩`}`.

`\wdPuzzleFields`
`\htPuzzleFields`

```
\def\wdPuzzleFields#1{\bgroup\setlength\@tempdima{#1}%
  \xdef\wd@fPF{\the\@tempdima}\egroup}
\def\wd@fPF{1.6em}
\def\htPuzzleFields#1{\bgroup\setlength\@tempdima{#1}%
  \xdef\ht@fPF{\the\@tempdima}\egroup}
\def\ht@fPF{11bp}
\def\dps@strut{\rule{0pt}{\ht@fPF}}
```

The definition of `\ds@makeTextField` is a function of the mode the document is in: for paper versus view mode.

```
\ifeqforpaper
  \ifviewMode
    \def\ds@makeTextField#1#2{\def\argii{#2}\ifx\argii
      \ds@myspace\let\ds@ul\relax\else\let\ds@ul\underbar\fi
      \ifshowletters\raisebox{-10pt}{%
      \makebox[0pt][l]{\makebox[\wd@fPF][c]{\footnotesize
      \ds@getProbNumber{#1}{#2}}}}\fi
      \ds@ul{\makebox[\wd@fPF][c]{\dps@strut
        \Hy@pdfstringfalse#1}}}
  \else
    \def\ds@makeTextField#1#2{\def\argii{#2}\ifx\argii\ds@myspace
      \let\ds@ul\relax\else\let\ds@ul\underbar\fi
      \ifshowletters\raisebox{-10pt}{%
      \makebox[0pt][l]{\makebox[\wd@fPF][c]{\footnotesize
      \ds@getProbNumber{#1}{#2}}}}\fi
      \ds@ul{\makebox[\wd@fPF][c]{\dps@strut\hfil}}}
  \fi
\else
  \ifviewMode
    \def\ds@makeTextField#1#2{\ifshowletters\raisebox{-10pt}{%
      \makebox[0pt][l]{\makebox[\wd@fPF][c]{%
      \footnotesize\ds@getProbNumber{#1}{#2}}}}\fi
      \underbar{\makebox[\wd@fPF][c]{\dps@strut
        \Hy@pdfstringfalse#1}}}
  \else
    \def\ds@makeTextField#1#2{\ifshowletters
      \raisebox{-10pt}{\makebox[0pt][l]{\makebox[\wd@fPF][c]{%
      \footnotesize\ds@getProbNumber{#1}{#2}}}}\fi
      \edef\textfieldTmp{\noexpand\textField[\noexpand\Q{1}%
      \noexpand\S{U}\noexpand\Ff\noexpand\FfReadOnly
      \the\ds@PuzzleAppearancetoks\noexpand\DV{#1}]%
      {puzzle.#2}{\wd@fPF}{\ht@fPF}}\Hy@pdfstringtrue\textfieldTmp}
```

```
  \fi
\fi
```

Later in this package the `\ds@buildAnswerKey` is defined and must be expanded at the appropriate moment. The command uses information from all components of the puzzle: the puzzle, the questions, and the answers. So, we must wait until all components have been typeset. This is the purpose of `\dps@emitAK`; its value is increased when each component is typeset. When it reaches a value of 4, it is at that time `\ds@buildAnswerKey` is expanded.

```
\def\dps@emitAK{0}
\def\dps@AddToEmitAK#1{\bgroup
  \@tempcnta=\dps@emitAK\relax
  \advance\@tempcnta by#1\relax
  \xdef\dps@emitAK{\the\@tempcnta}\egroup
}
\def\dps@ckEmitAK{\ifnum\dps@emitAK>\thr@@\expandafter
  \ifshowletters\expandafter\ds@buildAnswerKey\fi\fi}
```

5.3 Inserting the puzzle into the document

`\insertPuzzle` The command element that inserts the puzzle data that has already been declared with `\DeclarePuzzle`. Use `\insertPuzzle` to insert the puzzle at the location desired. The puzzle is placed in a tabular environment. The only argument of this command is the number of columns you want for this tabular environment. For example, `\insertPuzzle{18}` distributes the puzzle so that there are 18 columns per row, one letter per column.

```
\def\insertPuzzle#1{\begingroup\def\@rgi{#1}%
  \ifx\@rgi\@empty
    \ifx\nCols\@empty
      \PackageWarning{dps}{\string\insertPuzzle\space
      needs an argument, use\MessageBreak
      either \string\insertPuzzle{<nCols>} or\MessageBreak
      declare \string\nPuzzleCols{<nCols>} in preamble.\MessageBreak
      Setting <nCols> to 10 for now}\gdef\nCols{10}%
    \fi
  \else
    \xdef\nCols{#1}%
  \fi
  \ifviewMode\Hy@pdfstringfalse\else\Hy@pdfstringtrue\fi
  \let\tops\texorpdfstring\expandafter\dps@Puzzle
  \expandafter{\puzzleParameters}\endgroup
  \dps@AddToEmitAK{1}\dps@ckEmitAK
}
```

`\rowsep` This command is used to adjust the space between row of the tabular environment for the puzzle. The default is `\rowsep{2ex}`.

```
\def\rowsep#1{\gdef\@rowsep{[#1]}\gdef\@rowskip{#1}}
\rowsep{\rowsep@default}
```

```
\def\rowsep@default{2ex}
```

This code create the tabular environment, creating a new row when necessary, and inserts the text field or typesets the puzzle (in the case of `viewmode`).

```
\def\eq@tabSep{&}\def\ds@punc{punc}
```

The command that `\insertPuzzle` calls, the argument is the paired {⟨*letter*⟩}{⟨*name*⟩} data structure. We insert `\relax\relax` to identify the end of the data-structure, then pass on to `\dps@@Puzzle`

```
\def\dps@Puzzle#1{\edef\eq@tabEnd{\noexpand\\\@rowsep}%
  \dps@@Puzzle#1\relax\relax}%
```

`\dps@@Puzzle` begins a `tabular`, then passes the ball to `\@dpsPuzzlei`.

```
\def\dps@@Puzzle{\begin{tabular}
  {@{}*{\nCols}l@{}}\@dpsPuzzlei
}
```

Parse the data structure, taking care to handle `punc` and `cr` correctly. to `\@dpsPuzzlei`.

```
\def\@dpsPuzzlei#1#2{\ifx#2\relax
  \gdef\nextPuzzleChar{\@dpsPuzzleDone}\else
  \gdef\nextPuzzleChar{\@dpsPuzzleii{#1}{#2}}\fi
  \nextPuzzleChar
}
```

We've checked for `\relax` and we're OK to continue. The next pair may be a special pair, we don't make a field if its a special field (`cr` or `punc`).

```
\def\@dpsPuzzleii#1#2{\def\argii{#2}%
  \ifx\argii\ds@punc
  % so something with punc
    \def\@puzzNext{#1\@takeaPeek}%\@setSep@dpsPuzzleii}%
  \else\ifx\argii\ds@cr
  % do something with cr
    \def\@puzzNext{\global\ds@nCntCols\z@
    \eq@tabEnd\@dpsPuzzlei}%
  \else
  % ok to make a field
    \def\@puzzNext{\ds@makeTextField{#1}{#2}\@takeaPeek}%
  \fi\fi
  \@puzzNext
}
\def\ds@cr{cr}
\def\@takeaPeek#1#2{\def\argii{#2}%
  \ifx\argii\ds@punc
```

A punctuation can be followed by the `cr` token, we better check, by taking another peek.

```
    \def\@puzzNext{#1\@takeaPeek}%
  \else
```

We allow a markup to end the tabular line before reaching the number of `\nCols`. This may be needed when there is an long puzzle, and a `\nCols` gives weird row breaks.

```
    \ifx\argii\ds@cr
      \def\@puzzNext{\global\ds@nCntCols\z@
        \eq@tabEnd\@dpsPuzzlei}%
    \else
      \def\@puzzNext{\@setSep@dpsPuzzleii{#1}{#2}}%
    \fi
  \fi
  \@puzzNext
}
\def\@setSep@dpsPuzzleii{\global\advance\ds@nCntCols\@ne
  \ifnum\ds@nCntCols=\nCols\relax
    \expandafter\eq@tabEnd
    \global\ds@nCntCols\z@\else
    \expandafter\eq@tabSep\fi
  \@dpsPuzzlei}
\def\@dpsPuzzleDone{\end{tabular}\ifnum\ds@nCntCols=0\relax
    \ifshowletters\vskip\@rowskip\relax
    \else\vskip3pt\fi
    \vskip-2\baselineskip
  \fi\kern0pt}
```

5.4 The answer key

`\AnswerKey` The command `\AnswerKey` is defined by `\ds@buildAnswerKey`, which is expanded after all components of the puzzle have been typeset. and it (`\AnswerKey`) is available thereafter for manual insertion into the document. If the `showanswerkey` option is taken, it is displayed at the bottom of the page. If the `savedata` option is taken, the answer key is save to a file along with the random seed that generated this answer key. The file name containing the puzzle data is `\jobname_data.sav`.

```
\def\AnswerKey{The answer key is not available.\PackageWarning{dps}
  {The showletters option is required to generate\MessageBreak
    an answer key,}}
\def\ds@buildAnswerKey
{%
  {% local
    \Hy@pdfstringfalse
    \let\tops\texorpdfstring
    \let\protect\@unexpandable@protect
    \count@\z@\toks@={}%
    \loop
      \advance\count@\@ne
      \edef\y{\@nameuse{dps@probLetterii\the\count@}}%
      \edef\x{\@nameuse{dps@probLetterKey\y}}%
      \expandafter\ifx\x\relax\else
        \csarg\xdef{dps@probLetter\the\count@}{\expandafter\noexpand
```

```
          \csname dps@probLetterAlt\y\endcsname}\fi
      \xdef\AnswerKey{\the\toks@
      \the\count@--\@nameuse{dps@probLetter\the\count@}; }%
      \toks@=\expandafter{\AnswerKey}%
    \ifnum\count@ < \ds@qNumber\repeat
  }%
  \ifshowsolution
    \cfooter{\let\tops\texorpdfstring
    \footnotesize\@ifundefined{rotatebox}{\AnswerKey}%
    {\rotatebox{180}{\AnswerKey}}}\fi
}
\def\ds@writePuzzleData{\ifsavepuzzledata
  \newwrite \ds@savedata
  \begingroup
  \immediate\openout \ds@savedata \jobname_data.sav
  \def\msgi{Initial seed}
  \def\msgii{Answer key:}
  \let\verbatim@out\ds@savedata
  \uccode`c=`\%\uppercase{%
    \ifwerandomize
      \dps@IWVO{\string\randomi=\ds@saveRandomSeed\space
        c \msgi}%
      \dps@IWVO{\string\dpsLastSeed{\the\randomi}}\fi
    \ifshowletters\let\tops\texorpdfstring
      \set@display@protect
      \dps@IWVO{c \msgii\space\AnswerKey}\fi
  }
  \immediate\closeout\ds@savedata
  \endgroup
\fi}
\AtEndDocument{\ds@writePuzzleData}
```

`\setdpsfootskip{`⟨*skip*⟩`}` When the `showanswerkey` is in force, `\AnswerKey` is placed in the running footer (`\cfooter`) of `web`. To facilitate the positioning of the running footer, we define a convenience command to set `\web@footskip` used by `web`.

```
\def\setdpsfootskip#1{\bgroup
  \setlength\@tempdima{#1}\ifeqforpaper\else
  \xdef\web@footskip{\the\@tempdima}\fi
  \egroup\InitLayout % a web command
}
\setdpsfootskip{.25in}
```

5.5 Support for a sideshow

`\randomizePicMappings`

A sideshow consists of a tiled graphic that is revealed as the player works the puzzle. The techniques used to build a sideshow depends on the options `usebtnappr` and `uselayers`. The command `\randomizePicMappings`, when expanded in the preamble, will randomize the order the tiles appear in the slideshow; the default is no randomization.

```
\let\DPSIndxList\@empty
\let\DPSNamesList\@empty
\def\randomizePicMappings{\def\bRandPicMaps{true}}
\def\bRandPicMaps{false}
```

`\sortPicMappings` Pictures are randomly placed in the sideshow and the bubble sort is used to sort them out, as a final event when the puzzle is solved. This command does nothing if the `usebtnappr` option is not taken; that is, this is a feature of the `usebtnappr` option.

```
\ifusebtnappr
  \def\sortPicMappings{%
    \InputIfFileExists{sortjs.def}{}{}%
    \OpenAction{\JS{try{if(!hasBeenRandomized)
      {hasBeenRandomized=true;mixupDPS();showDPS();}
      }catch(e){}}}%
  }
\else
  \let\sortPicMappings\relax
\fi
```

`\sideshowPackaged` The side show pictures are packaged into a single PDF, the order of the pages are as expected. The default is they are not packaged. It is assumed the filename is ⟨*basename*⟩`_package.pdf` (When in a package, the pictures must be in a PDF file).

```
\newif\if@isPackaged \@isPackagedfalse
\def\sideshowPackaged{\@isPackagedtrue}
```

6 Questions and Answers

`Composing` This is the environments in which the composing of questions and answers are made. Use the `cQ` and `cA` for this purpose.

```
\newenvironment{Composing}
{\global\ds@qNumber=0 \global\ds@aNumber=0}
{\aftergroup\ds@publishRandomLists}
```

Here are the two environment (`cQ` and `cA`) for composing questions and answers. The argument of each environment corresponds to a form field in the puzzle. Each of these environments are verbatim write environments; they write each question and solution to a separate `.cut` file. These individual files are ultimately input in a random order. A typical pair of environments looks like these two:

`cQ` Sets the content of the question.

```
\newenvironment{cQ}[1]
{%
   \global\advance\ds@qNumber\@ne
   \immediate\openout \ds@question@write \jobname_q\the\ds@qNumber.cut
    \let\verbatim@out\ds@question@write\set@display@protect
    \dps@IWVO{\protect\dpsQ{#1}}%
```

```
    \set@typeset@protect
    \verbatimwrite
}{%
    \endverbatimwrite
    \immediate\closeout \ds@question@write
}
```

The `cA` environment has an optional argument. This optional argument is only used when the document is compiled with the showletters option. The value of the argument is a letter to appear in the answers column. Normally, first entry of the argument pair {⟨*letter*⟩}{⟨*name*⟩} of `\DeclarePuzzle` is used. Cases where you would want to include this optional argument are (1) when giving an answer that does not correspond to a question; (2) the letter is capitalized, suggesting a proper name or the beginning of a sentence, use the optional argument to list the letter in lower case. The latter case is common, for example,

```
\begin{cQ}{H}
  Who wrote this package?
\end{cQ}
\begin{cA}[h]{H}
  D.P. Story
\end{cA}
```

The capital 'H' begins the puzzle, but we don't want the player to see a capital 'H' if the `showletters` option is taken.

`cA` Sets the contents of the answer.

```
\newenvironment{cA}[2][]
{%
   \def\argi{#1}\global\advance\ds@aNumber\@ne
   \immediate\openout \ds@question@write \jobname_a\the\ds@aNumber.cut
    \let\verbatim@out\ds@question@write
    \set@display@protect
    \dps@IWVO{\protect\dpsA\ifshowletters
      \ifx\argi\@empty\else[#1]\fi\fi{#2}}%
    \set@typeset@protect
    \verbatimwrite}
{%
    \endverbatimwrite
    \immediate\closeout \ds@question@write
}
```

`\QuesAppearance` `\AnsAppearance` These two commands can be used to change the appearance of the checkboxes for the questions and answers. When the `forpaper` `web` option is taken, these have no effect.

```
\def\QuesAppearance#1{\ds@QuesAppearancetoks={#1}}
\def\AnsAppearance#1{\ds@AnsAppearancetoks={#1}}
```

The commands `\dpsQ` and `\dpsA` are the ones that produce the checkboxes, and define the JavaScript actions.

\afterQhookA
\OnFocusQhookAA

The two commands \afterQhookA and \OnFocusQhookAA can be redefined for additional JS action, the first is a hook into the mouse up action, and the second is a hook to the on focus action. These two can be redefined as needed, but be sure to preserve the JS functionality.

```
\def\afterQhookA#1{if(PlayerSignIn());}
%\let\afterQhookA\@gobble
\let\OnFocusQhookAA\@gobble
```

\widestFmtdQNum{⟨*text*⟩} Sets the width of the underlying checkbox for the question.

```
\def\widestFmtdQNum#1{\bgroup
  \settowidth{\@tempdima}{#1}%
  \ifxetex\advance\@tempdima2.5bp\else
    \advance\@tempdima.5pt\fi
  \xdef\Qwidth{\the\@tempdima}\egroup}
\widestFmtdQNum{00.}
\def\htOfQ#1{\setlength{\@tempdima}{#1}\ifxetex
  \advance\@tempdima2bp\relax\fi\edef\Qht{\the\@tempdima}}
\htOfQ{13bp}
```

Set the checkboxes and JS action for questions. Allow also, changes to appearance through the \QuesAppearance command.

\dpsQ Sets the checkbox and content of a question.

```
\newcommand{\dpsQ}[1]{\item\relax\ifeqforpaper\else
  \edef\checkboxTmp{\noexpand\checkBox[\the\ds@QuesAppearancetoks
  \noexpand\textSize{0}\noexpand
  \A{\noexpand\JS{activeQuestion = event.target.name;\noexpand
  \r clearRedCrosses();\noexpand\r\noexpand\afterQhookA{#1}}}%
  \noexpand\AA{\noexpand\AAOnFocus{\noexpand\JS{%
  this.resetForm(["ckbxQ"]);\noexpand\r\noexpand
  \OnFocusQhookAA{#1}}}}]{ckbxQ.#1}{\Qwidth}{\Qht}{Yes}}%
  \makebox[0pt][r]{\strut
    \smash{\checkboxTmp}\efKern{-.5pt}{-1.5pt}\enspace}\fi
  \ignorespaces
}
```

Set the checkboxes and JS action for answers. Allow also, changes to appearance through the \AnsAppearance command. This command obeys the showanswerlabels option by re-defining the command \dpsAitemOptArg. The default definition is \def\dpsAitemOptArg{[]}, which cancels the display of the item labels.

\ltrFmtA{⟨*fmt-cmds*⟩} When the showletters option is in effect, the letters appear amongst the answers. the letters may be formatted through \ltrFmtA. The symbolic #1 represents the letter to be formatted; eg, \ltrFmtA{\textbf{#1}} give letters in bold. The default is no formatting.

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

\widestFmtdALtr{⟨*text*⟩} Sets the width of the checkbox around the letter. There is a built in with of 8pt.

```
\def\widestFmtdALtr#1{\bgroup
  \settowidth{\@tempdima}{#1}\ifxetex
    \addtolength{\@tempdima}{2bp+8pt}\else
    \addtolength{\@tempdima}{8pt}\fi
  \xdef\Awidth{\the\@tempdima}\egroup}
\widestFmtdALtr{w}
\def\htOfA#1{\setlength{\@tempdima}{#1}\ifxetex
  \advance\@tempdima2bp\relax\fi\edef\Aht{\the\@tempdima}}
\htOfA{13bp}
```

`\dpsA` Checkbox for the answer.

```
\newcommand{\dpsA}[2][]{\expandafter
  \item\dpsAitemOptArg\relax
  \ifeqforpaper\else
    \edef\checkboxTmp{\noexpand
      \checkBox[\the\ds@AnsAppearancetoks
      \noexpand\textSize{0}\noexpand
      \A{\noexpand\JS{processChoice("#2");}}]%
      {ckbxA.#2}{\Awidth}{\Aht}{Yes}}%
    \makebox[0pt][r]{\strut
      \smash{\checkboxTmp}\enspace}%
  \fi
  \def\argi{#1}%
  \bgroup % dps
    \let\tops\texorpdfstring
    \Hy@pdfstringfalse
    \ifx\argi\@empty
      \global\csarg\let{dps@probLetterKey#2}\relax
    \else
      \csarg\gdef{dps@probLetterKey#2}{#2}%
      \csarg\gdef{dps@probLetterAlt#2}{#1}%
    \fi
  \egroup
  \ifshowletters
    \ifx\argi\@empty
      \let\getLetterNext\relax
      \let\ds@foundLetter\dps@Zero\def\ds@currFN{#2}%
      \def\getLetterNext{\ds@typesetPuzzleLetter}%
      \expandafter\getLetterNext
    \else
      \makebox[0pt][r]{\let\tops\texorpdfstring
        \Hy@pdfstringfalse\@ltrFmtA{#1}\enspace\kern4bp}%
    \fi
  \fi
  \ignorespaces
}

\def\ds@typesetPuzzleLetter{\expandafter
  \typeset@PuzzleLetter\expandafter{\puzzleParameters}}
\def\typeset@PuzzleLetter#1{\typeset@@PuzzleLetter#1\relax\relax}%
```

```
\def\typeset@@PuzzleLetter#1#2{%
  \ifx#1\relax
    \gdef\nextPuzzleLetter{\relax}\else
    \gdef\nextPuzzleLetter{\typeset@PuzzleLetteri{#1}{#2}}\fi
  \nextPuzzleLetter
}
\def\typeset@PuzzleLetteri#1#2{\def\argii{#2}\ifx\argii\ds@currFN
  \ifx\ds@foundLetter\dps@Zero
    \makebox[0pt][r]{\let\tops\texorpdfstring
    \Hy@pdfstringfalse\@ltrFmtA{#1}\enspace\kern4bp}%
    \let\ds@foundLetter\dps@One\fi
    \expandafter\typeset@@PuzzleLetter
  \else
    \expandafter\typeset@@PuzzleLetter
  \fi
}
```

This command is called at `\end{Composing}`, which, in turn, calls the macros `\ds@randomizeQuestionList` and `\ds@randomizeAnswerList` which randomly permutes the `1, 2, ... \the\ds@qNumber` and `1, 2, ... \the\ds@aNumber`, where `\ds@qNumber` and `\ds@aNumber` are the number of questions and answers, respectively.

```
\def\ds@publishRandomLists{%
  \ds@randomizeQuestionList{\the\ds@qNumber}%
  \ds@randomizeAnswerList{\the\ds@aNumber}%
}
\def\ds@myspace{space}
\def\ds@getProbNumber#1#2{%
  \gdef\ds@currentArgi{#1}\gdef\ds@currentArgii{#2}%
  \ifx\ds@currentArgii\ds@myspace\def\ds@probNumNext{\relax}\else
    \def\ds@probNumNext{%
      \@ifundefined{dps@probNum#2}{\global\advance\ds@probCnt\@ne
        \ds@@getProbNumber{\the\ds@probCnt}}
        {\csname dps@probNum#2\endcsname}%
  }%
  \fi
  \ds@probNumNext
}
\def\ds@@getProbNumber#1{%
  {\count@\z@ \let\\=\ds@getNthOne\the\ds@qlistOut}%
  \csname dps@probNum\ds@currentArgii\endcsname
}
\def\ds@getNthOne#1{\advance\count@\@ne
  \ifnum\ds@probCnt=#1\relax
    \csarg\xdef{dps@probNum\ds@currentArgii}{\the\count@}%
    \toks@=\expandafter{\ds@currentArgi}% dps
    \csarg\xdef{dps@probLetter\the\count@}{\the\toks@}% dps
    \csarg\xdef{dps@probLetterii\the\count@}%
      {\ds@currentArgii}%
  \fi
```

```
}
```

`\writeComposingEnv` This is a helper macro. After you declare your puzzle, `\DeclarePuzzle`, you can place this command just after, if needed, like so

```
\writeComposingEnv
```

In the case where `\DeclarePuzzle` is in the preamble; above, we begin the doc and end the doc; assuming you have not developed your questions yet. The command writes to the file `\jobname_comp.def`. This file will be a skeleton of your Composing environment, with correct labeling. Copy and paste it into your document in some appropriate location, and begin writing your questions.

```
\def\writeComposingEnv{%
  \newwrite \ds@composing@write
  \immediate\openout \ds@composing@write \jobname_comp.def
  \let\verbatim@out\ds@composing@write
  \dps@IWVO{\string\begin{Composing}}%
  \dps@IWVO{}%
  \expandafter\write@ComposingEnv\expandafter{\puzzleParameters}%
}
\def\write@ComposingEnv#1{\write@@ComposingEnv#1\relax\relax}%
\newcommand{\ComposingEnvMsg}{\begin{quote}An outline of your
  \texttt{Composing}environment is written to
  \texttt{\jobname\_comp.def}, based on data in the
  argument of your \texttt{\string\DeclarePuzzle} command. Copy and
  paste the contents of this file into your puzzle document following
  \texttt{\string\DeclarePuzzle} then fill in your questions and
  answers. Good luck.\end{quote}}
\def\write@@ComposingEnv#1#2{\ifx#1\relax
  \gdef\nextPuzzlePair{%
  \dps@IWVO{\string\end{Composing}}%
  \immediate\closeout \ds@composing@write}%
  \begin{document}
  \ComposingEnvMsg
  \end{document}
  \else\gdef\nextPuzzlePair{\write@@@ComposingEnv{#1}{#2}}\fi
  \nextPuzzlePair}%
\def\write@@@ComposingEnv#1#2{%
  \gdef\ds@currentArgi{#1}\gdef\ds@currentArgii{#2}%
  \ifx\ds@currentArgii\ds@myspace
    \def\ds@probNumNext{\write@@ComposingEnv}%
  \else
    \ifx\ds@currentArgii\ds@punc
      \def\ds@probNumNext{\write@@ComposingEnv}\else
      \def\ds@probNumNext{%
        \@ifundefined{dps@compQ#2}{% write to file
        \expandafter\gdef\csname dps@compQ#2\endcsname{found}%
        \dps@IWVO{\string\begin{cQ}{\noexpand#1}}%
        \dps@IWVO{\string\end{cQ}}%
        \dps@IWVO{\string\begin{cA}{\noexpand#1}}%
```

```
        \dps@IWVO{\string\end{cA}}%
        \dps@IWVO{}%
        }{}%
        \write@@ComposingEnv
      }%
    \fi
  \fi
  \ds@probNumNext
}
```

A standard `\verbatim` write used in exerquiz and other package in the AeB family.

```
\def\verbatimwrite{\@bsphack
  \let\do\@makeother\dospecials
  \catcode`\^^M\active \catcode`\^^I=12
  \def\verbatim@processline{%
    \immediate\write\verbatim@out
    {\the\verbatim@line}}%
  \verbatim@start}
\def\endverbatimwrite{\@esphack}
\def\dps@IWVO{\immediate\write\verbatim@out}
```

`\ds@populateList` is a utility command, its argument is a positive integer, `n`, and it generates a list of the form `\\{1}\\{2}...\\{n}`. This listing is later randomly permuted by `\ds@randomizeQuestionList` and `\ds@randomizeAnswerList`.

```
\def\ds@populateList#1{%
  \ds@listIn={}%
  \ds@nCnt\z@
  \@whilenum \ds@nCnt < #1\do {%
    \advance\ds@nCnt\@ne
    \edef\ds@listInHold{\the\ds@listIn\noexpand\\{\the\ds@nCnt}}%
    \ds@listIn = \expandafter{\ds@listInHold}%
  }%
}
```

Used in `\ds@randomizeList` to build the permuted list of numbers.

```
\def\ds@processi#1{\advance\ds@nCnt\@ne
  \ifnum\ds@nCnt=\ds@getRanNum\edef\ds@listOutHold{\the\ds@listOut}%
    \global\ds@listOut=\expandafter{\ds@listOutHold\\{#1}}%
  \else
    \edef\ds@listInHold{\the\ds@newListIn}%
    \ds@tmpToks = \expandafter{\ds@listInHold\\{#1}}%
    \ds@newListIn = \expandafter{\the\ds@tmpToks}%
  \fi
}
```

Used in `\displayRandomizedQuestions` to input the questions.

```
\def\ds@processii#1{\input{\jobname_q#1.cut}}
```

Used in `\displayRandomizedAnswers` to input the answers.

```
\def\ds@processiii#1{\input{\jobname_a#1.cut}}
```

Used in `\displayRandomizedAnswersLeftPanel` to input the top half of the permuted list.

```
\def\ds@processL#1{\advance\count@\@ne
  \ifnum\count@>\ds@aNumber\relax\else\input{\jobname_a#1.cut}\fi}
```

Used in `\displayRandomizedAnswersRightPanel` to input the bottom half of the permuted list.

```
\def\ds@processR#1{\advance\count@\@ne
  \ifnum\count@>\ds@aNumber\relax\input{\jobname_a#1.cut}\fi}
```

isplayRandomizedQuestions
\displayRandomizedAnswers
andomizedAnswersLeftPanel
ndomizedAnswersRightPanel

These are user commands that actually display the randomized questions and answers.

```
\def\displayRandomizedQuestions{\let\\=\ds@processii\the\ds@qlistOut
  \dps@AddToEmitAK{1}\dps@ckEmitAK}
\def\displayRandomizedAnswers{\set@typeset@protect
  \let\\=\ds@processiii\the\ds@alistOut
  \dps@AddToEmitAK{2}\dps@ckEmitAK} % dps
\def\displayRandomizedAnswersLeftPanel{{\set@typeset@protect
  \let\\=\ds@processL\count@\z@
  \divide\ds@aNumber\tw@ \xdef\lastOnLeft{\the\ds@aNumber}%
  \the\ds@alistOut}%
  \dps@AddToEmitAK{1}\dps@ckEmitAK}
\def\displayRandomizedAnswersRightPanel{{\set@typeset@protect
  \let\\=\ds@processR\count@\z@
  \divide\ds@aNumber\tw@ \the\ds@alistOut}%
  \dps@AddToEmitAK{1}\dps@ckEmitAK}
```

Develop a random permuted list for the questions.

```
\def\ds@randomizeQuestionList#1{%
  \global\ds@listIn={}\global\ds@newListIn={}\global\ds@listOut={}%
  \global\ds@tmpToks={}%\global\ds@qlistOut={}\global\ds@alistOut={}%
  \ds@nMax=#1\relax\ds@populateList{\the\ds@nMax}%
  \global\forquestionstrue
  \ifwerandomize
    \expandafter\ds@randomizeList
  \else
    \global\ds@qlistOut=\expandafter{\the\ds@listIn}
  \fi
}
```

Develop a random permuted list for the answers. Note that `\ifwerandomize` we randomize, else, the output list is the same as the input list.

```
\def\ds@randomizeAnswerList#1{%
  \global\ds@listIn={}\global\ds@newListIn={}\global\ds@listOut={}%
  \global\ds@tmpToks={}%\global\ds@qlistOut={}\global\ds@alistOut={}%
  \ds@nMax=#1\relax\ds@populateList{\the\ds@nMax}%
  \global\forquestionsfalse
  \ifwerandomize
    \expandafter\ds@randomizeList
  \else
```

```
    \global\ds@alistOut=\expandafter{\the\ds@listIn}%
  \fi
}
```

The loop that does all the work for randomizing.

```
\def\ds@randomizeList{\redefnextrandomAsNeeded
  \let\\=\ds@processi
  \setrannum{\ds@getRanNum}{1}{\ds@nMax}%
  \ds@nCnt\z@
%\typeout{LISTING: \the\ds@listIn}%
  \the\ds@listIn
  \ds@loopTest
}
```

The loop that does all the work for randomizing.

```
\def\ds@loopTest{\advance\ds@nMax\m@ne\relax
  \ifnum\ds@nMax>\z@
    \def\@next{%
      \ds@listIn=\expandafter{\the\ds@newListIn}%
      \ds@newListIn={}\ds@randomizeList}%
  \else
    \let\@next\relax
    \ifforquestions
      \global\ds@qlistOut=\expandafter{\the\ds@listOut}%
%\typeout{\ds@qlistOut = \the\ds@qlistOut}%
    \else
      \global\ds@alistOut=\expandafter{\the\ds@listOut}%
%\typeout{\ds@alistOut = \the\ds@alistOut}%
    \fi
  \fi
  \@next
}
```

`\placeMessageField` The command inserts the required message field. The optional first parameters enables the author to change the appearance of the field, the second two required arguments are the width and the height of the text field.

```
\newcommand{\placeMessageField}[3][]{\ifeqforpaper\else
  \textField[\Ff\FfReadOnly\BC{}#1
  \Ff\FfMultiline]{report}{#2}{#3}\fi}
```

7 Miscellaneous Settings

`\threshold` `\penaltypoints` `\passing` Threshold for number of incorrect answers for trying to answer one question. If the threshold is exceeded, `\dspenaltypoints` are added to the final points. Passing is missing no more than `\dspassing`.

```
\newcommand{\threshold}[1]{\def\dsthreshold{#1}}
\threshold{3}
\newcommand{\penaltypoints}[1]{\def\dspenaltypoints{#1}}
\penaltypoints{3}
```

```
\newcommand{\passing}[1]{\def\dspassing{#1}}
\passing{4}
```

⟨/package⟩

8 Advanced features

One problem when building an interactive puzzle is the lack of space for a rather long or complex question. All of the standard designs leave little space for the questions. Over the years, I've developed two methods to create more space for asking questions: (1) Place the questions in push button appearances; (2) Place the questions in layers (ocgs). In this section we provide define some basic commands to make if "easy" to ask longer question using either of these two methods.

8.1 Icon button appearances

In this case, in some central region we create a series of push buttons, all of which are initialy hidden. Push button can have an icon (a graphic) for its appearance. As the student works through the puzzle, the buttons are made visible to pose the question. The question is hidden again as the student moves on to the next question.

For this solution, two files are required: (1) the main puzzle file; and (2) an "icon" file. The next two subsections include commands and environments for each of these two files.

`usebtnappr` To use the icon button approach, the `usebtnappr` must be specified; in this case the package icon-appr in input by this package.

Workflow: Creating a finished puzzle is a three step process:

`wrtContent` 1. Compile the puzzle file with the option `wrtContent`

2. Compile the `icons.tex` file to create one or more icons files

`!wrtContent` 3. Compile the puzzle file with the option `!wrtConent`

8.1.1 Macros for the main puzzle document

⟨*btnadv⟩

`setContent{`⟨*name*⟩`}` (Where ⟨*name*⟩ is the second argument of the `\DeclarePuzzle` data structure.) An environment to set content of the question. The contents of the `setContent` is written verbatim to the file `\jobname-sc(\theenumi).cut`. This obviously assumes the list of questions is in an `enumerate` environment. These individual files are compiled together (in the `icons.tex` file to a PDF of all the questions. The formatting command `\quesNumTxt` is used to format the question header; the default is **Problem** ⟨*num*⟩. Additionally, `\quesNumTxTPost` that expands immediatelly after `\quesNumTxt`.

`\quesNumTxt`
`\quesNumTxTPost`

```
\newcommand{\quesNumTxt}[1]{\protect\textbf{Problem #1}}
\newcommand{\quesNumTxTPost}{\protect\newline}
\newenvironment{setContent}[1]{%
  \immediate\write\@auxout{\string\csarg
    \string\xdef{ltrpg#1}{\theenumi}}%
  \ifwrtContent
    \def\CommentCutFile{\jobname-sc(\theenumi).cut}%
    \immediate\openout\CommentStream=\CommentCutFile
    \begingroup
    \set@display@protect
    \let\verbatim@out\CommentStream
    \dps@IWVO{\quesNumTxt{\theenumi}\quesNumTxTPost}%
    \set@typeset@protect
    \expandafter\verbatimwrite
  \else
    \edef\x{\noexpand\pl@ceQues{\theenumi}}\x\expandafter
    \comment
  \fi
}{\ifwrtContent\expandafter
    \endverbatimwrite
    \endgroup
    \immediate\closeout\CommentStream
  \else
    \expandafter\endcomment
  \fi
}
```

`\ltrToNum{`⟨*num*⟩`}` The `setContent` environment above defines a series of commands that associate with each ⟨*name*⟩, the corresponding problem number (⟨*num*⟩).

```
\def\ltrToNum#1{\@nameuse{ltrpg#1}}
```

Embedding commands

`\dpsEmbedIcons` After creating the CUT files, as described in the previous environment, we need to compile the CUT files into one or more PDF(s). This is done in the other `icon.tex` file, described below. The `\dpsEmbedIcons` command goes within the `embedding` environment

```
\begin{embedding}
  \dpsEmbedIcons
\end{embedding}
```

The `embedding` environment is defined in the icon-appr package. `\dpsEmbedIcons` embeds icons images into the puzzle document.

```
\def\pglstWarningMsg{\PackageWarningNoLine{dps}
  {The file icons-pglst.sav not found.\MessageBreak
   Icons may not appear. Build the\MessageBreak icons.tex file}}
\ifwerandomize\else\let\pglstWarningMsg\@empty\fi
\def\dpsEmbedIcons{%
\InputIfFileExists{icons-pglst.sav}{\wrtContentfalse}
```

```
706   {\pglstWarningMsg\def\pagelist{}}%
707   \edef\TFOR{\noexpand\@tfor\noexpand\n:=\pagelist}%
708   \ifxetex
709     \TFOR\do{\embedIcon[name=Q\n]{icons-\n.pdf}}%
710   \else\ifpdf
711     \TFOR\do{\embedIcon[name=Q\n,hyopts={page=\n}]{icons.pdf}}%
712   \else % pdfmark
713     \TFOR\do{\embedIcon[name=Q\n,placement=btnQ.\n,%
714       page={\n-1}]{icons.pdf}}%
715   \fi\fi
716 }
```

Icon button fields

`\dpsQuesIcon[`⟨*opts*⟩`]{`⟨*num*⟩`}{`⟨*wd*⟩`}{`⟨*ht*⟩`}` The command that create a push button with an icon appearance. The ⟨*num*⟩ is the question number. The field name is `"btnQ.`⟨*num*⟩`"`; the value of the `\I` key is an indirect reference to the embedding of the image to be use, the reference is `Q`⟨*num*⟩.

```
717 \newcommand{\dpsQuesIcon}[4][]{%
718   \pushButton[\Ff{\FfReadOnly}\BG{}\S{S}#1\TP{1}\F{\FHidden}
719     \I{\csOf{Q#2}}\PA{.5 1}]{btnQ.#2}{#3}{#4}}
```

`\dpsOtherIcon[`⟨*opts*⟩`]{`⟨*fieldname*⟩`}{`⟨*wd*⟩`}{`⟨*ht*⟩`}` There is allowance for displaying additional button images. The

```
720 \newcommand{\dpsOtherIcon}[4][]{% \I{\csOf{name}} required
721   \pushButton[\Ff{\FfReadOnly}\BG{}\S{S}#1\TP{1}\F{\FHidden}
722     \PA{.5 1}]{#2}{#3}{#4}}
```

Placing the button icons. There are several ways of placing the image buttons; I have use both the textpos and the eso-pic packages, lately, I've preferred the latter package. Examples of both are contained in the examples.

`\placeQuesIcon{`⟨*place \dpsQuesIcon*⟩`}` We illustrate

```
\placeQuesIcon{\AddToShipoutPictureFG*{\AtTextCenter{\put(-72,0)
  {\dpsQuesIcon{#1}{2.25in}{9\baselineskip}}}}}
```

using the eso-pic package. The argument `#1` is eventually the problem number. `\placeQuesIcon` defines a macro `\pl@ceQues`, which appears in the `setContent` environment above.

`setContent`

```
723 \long\def\placeQuesIcon#1{\@ifundefined{textblock}
724   {\let\dps@mode\relax}{\let\dps@mode\par}%
725   \def\pl@ceQues##1{\dps@mode #1}}
```

`\placeOtherIcon{`⟨*place \dpsOtherIcon*⟩`}` Places an image other than a question.

```
\placeOtherIcon{\AddToShipoutPictureFG*{\AtTextCenter{\put(-72,0)
  {\dpsOtherIcon[\I{\csOf{Emoji}}]{btnEmoji}{2.25in}{9\baselineskip}}}}}
```

Both of the above examples are from `examples/advanced/stat_match1.tex`.

```
726 \long\def\placeOtherIcon#1{#1}
```

Define hooks into the question checkbox event. The two commands `\afterQhookA` and `\OnFocusQhookAA` are hooks onto the `\dpsQ` command. This allows us to post process the user's choice of a questions, and allows us to execute JS on focus.

```
\def\afterQhookA#1{%
  if(!event.target.isBoxChecked(0))dpsHideQFields();\r
  else\pdfSP if(PlayerSignIn())dpsShowQues("\ltrToNum{#1}");}
\def\OnFocusQhookAA#1{dpsHidePreviousQues("\ltrToNum{#1}")}
```

Support for a sideshow A sideshow is a tiled picture that is revealed as the player solves the puzzle.

Embedding sideshow graphics

`\dpsEmbedSideShow[`⟨*ext*⟩`]{`⟨*n-pics*⟩`}{`⟨*path*⟩`}` We take a graphic and explode it into rows and columns, ⟨*n-pics*⟩ is the total number of tiled pictures. We assume the tiles are created row-wise. We assume also a naming convention for the tiles if `mypic` is the base-name of the picture or graphic, then the tiles are named `mypic_01`, `mypic_02`, `mypic_03`, ... It is assumed a single digit index has a leading 0. Use the command `\sideshowPackaged` prior to `\dpsEmbedSideShow`.

```
\newcommand{\dpsEmbedSideShow}[3][]{\begingroup
%  \def\dps@NumSideShowPics{#2}%
  \gdef\dpsNumSideShowPics{#2}%
  \def\@Ext{#1}\ifx\@Ext\@empty\def\@Ext{.pdf}\else\def\@Ext{.#1}\fi
  \@tempcnta\z@
  \let\@embedList\@empty
  \let\DPSIndxList\@gobble
  \@whilenum \@tempcnta < \dpsNumSideShowPics \do{%
    \ds@nCnt\@tempcnta \advance\ds@nCnt\@ne
    \ifnum\ds@nCnt<10 \edef\x{0\the\ds@nCnt}\else
      \edef\x{\the\ds@nCnt}\fi
    \edef\z{\noexpand\g@addto@macro\noexpand\DPSIndxList{,"\x"}}\z
    \ifxetex\if@isPackaged
      \PackageWarning{dps}
      {There is no support for embedding packaged\MessageBreak
      PDFs with xelatex. Ignoring the \string\isPackaged\MessageBreak
      command}%
      \@isPackagedfalse
    \fi\fi
    \if@isPackaged
      \ifpdf
        \edef\y{\noexpand
          \embedIcon[name=pic\x,%
          hyopts={page=\x}]{#3_package.pdf}}%
      \else
        \edef\y{\noexpand
          \embedIcon[name=pic\x,placement=btnpic.\x,%
          page=\x-1]{#3_package.pdf}}%
      \fi
```

```
    \else
      \edef\y{\noexpand
        \embedIcon[name=pic\x,placement=btnpic.\x]{#3_\x\@Ext}}%
    \fi
    \expandafter\g@addto@macro\expandafter\@embedList\expandafter{\y}%
    \@tempcnta\ds@nCnt
  }% do
  \toks@=\expandafter{\@embedList}\the\toks@
  \endgroup
}
```

Inserting sideshow graphics

`\insertSideshow{`⟨*rows*⟩`}{`⟨*cols*⟩`}{`⟨*wd*⟩`}{`⟨*ht*⟩`}` Command for placing the tiles of a picture. We assume that the pictures are number consecutively across rows.

- ⟨*rows*⟩ the number of rows
- ⟨*cols*⟩ the number of columns
- ⟨*wd*⟩ the width of a tile
- ⟨*ht*⟩ the height of a tile

`\tileKVs`⟨*KV-pairs*⟩ A way to pass eform key-values to the optional argument of the underlying push button.

```
\def\tileKVs#1{\def\tile@KVs{#1}}
\tileKVs{}
\newcommand\insertSideshow[4]{\begingroup
  \offinterlineskip\@tempcnta\z@
  \multido{\iR=1+1}{#1}{\hbox{%
    \multido{\iC=1+1}{#2}{%
      \global\advance\@tempcnta\@ne
      \ifnum\@tempcnta<10\relax
        \edef\x{0\the\@tempcnta}\else
        \edef\x{0\the\@tempcnta}\fi
      \edef\iconPresets{\noexpand\I{\noexpand\csOf{pic\x}}}%
      \dpsOtherIcon[\BC{}\FB{true}\presets{\iconPresets}
        \presets{\tile@KVs}]{btnpic.\x}{#3}{#4}%
    }% inner multido
  }}% hbox, outer multido
  \endgroup
}
```

⟨/btnadv⟩

8.1.2 Macros for the icon document

icon-doc package

The icon document is separate from the puzzle document, so we need to create a special package (icon-doc) for it.

⟨∗icondoc⟩

```
\NeedsTeXFormat{LaTeX2e}[1997/12/01]
```

```
\ProvidesPackage{icon-doc}
   [2020/04/21 v1.0 icon-doc:
    Build Icon file and explode same (dps)]
\newif\ifdpsuseacrobat \dpsuseacrobatfalse
```

xelatex author
useacrobat
!useacrobat

The icon-doc package has one option (and one convenience option). Both options are targeted at users of xelatex. If the xelatex has Acrobat then use the `useacrobat` option; otherwise, use use the `!useacrobat` option. The default is `!useacrobat` so this option need not appear in the option list. More on the problems of xelatex in the description of `\@MultiQuesFiles` below.

```
\DeclareOption{useacrobat}{\dpsuseacrobattrue}
\DeclareOption{!useacrobat}{\dpsuseacrobatfalse}
\DeclareOption{twice}{\dpscomptwicetrue}
\newif\ifdpscomptwice \dpscomptwicefalse
\ProcessOptions\relax
\RequirePackage{ifxetex}
\RequirePackage{shellesc}
\RequirePackage{web}
\RequirePackage{eforms}
\execJSOn
\pagestyle{empty}
\parindent0pt \parskip0pt
\newwrite \wrtPkg
\newwrite\wrticonbody
\def\IWB#1{\immediate\write\wrticonbody{#1}}
\def\IWP#1{\immediate\write\wrtPkg{#1}}
```

Some standard code for writing verbatim to a file.

```
\def\verbatimwrite{\@bsphack
    \let\do\@makeother\dospecials
    \catcode`\^^M\active \catcode`\^^I=12
    \def\verbatim@processline{%
        \immediate\write\verbatim@out
        {\the\verbatim@line}}%
    \verbatim@start}
\def\endverbatimwrite{\@esphack}
```

icondoc
icons-template

Through the `icondoc` environment, you can set the LaTeX document into which the question content will be inserted. This environment is only used for the case of a xelatex user without Acrobat. The environment writes the is contents verbatim to the file `icons-template.tex`. This file is later input by `\@MultiQuesFiles` to build individual icon files.

```
\newenvironment{icondoc}
{%
   \immediate\openout \wrticonbody icons-template.tex
    \let\verbatim@out\wrticonbody
    \IWB{\string\RequirePackage{tmp}}%
    \verbatimwrite
}{%
    \endverbatimwrite
```

```
    \immediate\closeout \wrticonbody
}
```

The prototype, as well as the default definition of `icondoc`, based on `examples/advanced/stat_match1.tex`

```
\ifxetex\ifdpsuseacrobat\else
\begin{icondoc}
\documentclass{article}
\usepackage{web}
\margins{3pt}{3pt}{3pt}{3pt}
\screensize{9\baselineskip}{2.25in}
\parindent0pt
\begin{document}
\small
\dpsInputContent    % required, defined in \@MultiQuesFiles
\end{document}
\end{icondoc}
\fi\fi
```

During the course of compiling the icon document, we keep track of which questions have an icon. It may be that some questions are short enough to fit in the questions area. `\addToPageList` appearing in both `\@SnglQuesFile` and `\@MultiQuesFiles`. The resulting list `\pageList` is automatically written to the file `icons-pglst.sav` at the end of the icon document.

```
\let\pageList\@empty
\def\addToPageList#1{\edef\x{{#1}}\expandafter
  \g@addto@macro\expandafter
  \pageList\expandafter{\x}}
```

`\@SnglQuesFile` (An internal command) Under certain conditions, the command publicly known as `\createRequiredIcons`. This command is used for non-xelatex users, and by xelatex users who use Acrobat.

```
\def\@SnglQuesFile#1#2{%
  \@tempcnta#1\relax\advance\@tempcnta\@ne
  \edef\N{\the\@tempcnta}%
  \@tempcnta\@ne\relax
  \@whilenum\@tempcnta < \N \do{%
    \begingroup
```

Here is a key point. If the target file does not exist, that means that question does not use an icon appearance, so we create a blank PDF icon and do not register it with `\addToPageList`; otherwise, we do register it with `\addToPageList`.

```
      \InputIfFileExists{#2-sc(\the\@tempcnta).cut}
        {\addToPageList{\the\@tempcnta}}{\null}\par
    \endgroup
```

One icon per page.

```
    \newpage
    \advance\@tempcnta\@ne
  }%
}
```

\@MultiQuesFiles This command is publicly known as \createRequiredIcons when the author is a xelatex user who does not have Acrobat.

Problem with xelatex. When it comes to embedding a PDF in the document, xelatex does not recognize the page key; as a result, the questions must be wrapped in a separate icon file to be later imported into the puzzle document. In all other cases, we can conveniently place a questions in a single icon file, and import a particular page into the document corresponding to the question.

```
\def\@MultiQuesFiles#1#2{%
  \@tempcnta#1\relax\advance\@tempcnta\@ne
  \edef\N{\the\@tempcnta}%
  \@tempcnta\@ne\relax
  \@whilenum\@tempcnta < \N \do{%
    \IfFileExists{#2-sc(\the\@tempcnta).cut}
      {\addToPageList{\the\@tempcnta}}{}
```

The above lines are the same as in \@SnglQuesFile. Here's where we differ. We begin by creating a temporary package named tmp that contains the definition of \dpsInputContent.

```
    \immediate\openout\wrtPkg tmp.sty
      \IWP{\string\def\string\dpsInputContent{\string
        \InputIfFileExists{#2-sc(\the\@tempcnta).cut}%
        {}{\string\null}}}%
    \immediate\closeout\wrtPkg
```

\ShellEscape Key to the workflow for the author using xelatex without Acrobat is to use \ShellEscape. We compile (using xelatex) icons-template twice, for no apparent reason, followed by renaming the resultant PDF to icons-⟨*num*⟩.pdf, which is the name expected by the puzzle document.

```
    \ShellEscape{xelatex icons-template.tex}%
    \ifdpscomptwice\ShellEscape{xelatex icons-template.tex}\fi
    \ShellEscape{copy icons-template.pdf icons-\the\@tempcnta.pdf}%
    \advance\@tempcnta\@ne
  }\null % content for the icons.tex file
```

We finish up by deleting all of working files, including the temporary package tmp.

```
  \ShellEscape{del tmp.sty icons-template.*}%
}
```

\createRequiredIcons{⟨*n-ques*⟩}{⟨*puzzle-basename*⟩} This command creates either a single file containing ⟨*n-ques*⟩ pages of the required icons, or it creates ⟨*n-ques*⟩ icon files, each file containing one of the questions. For non-xelatex users, we use \@SnglQuesFile if the useacrobat option is taken, otherwise, we use \@MultiQuesFiles. In all other cases, \@SnglQuesFile is used.

```
\ifxetex
  \ifdpsuseacrobat
    \let\createRequiredIcons\@SnglQuesFile
  \else
    \let\createRequiredIcons\@MultiQuesFiles
  \fi
```

```
\else
  \let\createRequiredIcons\@SnglQuesFile
\fi
```

This JavaScript is used by xelatex users who have Acrobat. I should mention, this code assumes the document author has aeb_pro installed, including, most importantly, the correct installation of `aeb.js` and `aeb_pro.js`.

aeb.js & aeb_pro.js

```
\begin{defineJS}[\def\defineJSjsR{^^J}]{\execExplode}
/* Extract pages to folder */
// Regular expression used to acquire the base name of file
try {
for (var i = 0; i < this.numPages; i++)
aebTrustedFunctions(this,aebExtractPages,{
nStart: i,
cPath: "icons-" + (i+1) +".pdf"
});
} catch (e) { console.println("Aborted: " + e); }
\end{defineJS}
```

Write the file `icons-pglst.sav` at end of the document.

```
\def\wrtPageList{\newwrite\pagelist
  \immediate\openout \pagelist icons-pglst.sav
  \immediate\write\pagelist{\string\def\string\pagelist{\pageList}}
  \immediate\closeout\pagelist
}
```

One last case for xelatex users. If the author has Acrobat, we "explode" the single icon document into into its individual pages with the correct naming convention.

```
\ifxetex\ifdpsuseacrobat
\begin{execJS}{expl}
\execExplode
\end{execJS}
\fi\fi
```

Finally, we write `icons-pglst.sav` and the end of the document.

```
\AtEndDocument{\wrtPageList}
\let\WriteBookmarks\relax
```

⟨/icondoc⟩

8.2 OCG methods

Using OCG methods does not require an "`icons.tex`" file, the puzzle file is entirely self contained. The questions are typeset into their own layer, which we make visible or invisible, depending on the question selected. We require the aeb_pro package (with its `uselayers` option) to be properly installed with its JS files, as described in the manual of that package. In the preamble you can conveniently type,

aeb_pro required

```
\usepackage[%
```

```
    web={pro,tight},
    eforms,
    uselayers
  ]{aeb_pro}
```

910 ⟨∗ocgadv⟩

Define hooks into the question checkbox event. The two commands `\afterQhookA` and `\OnFocusQhookAA` are hooks onto the `\dpsQ` command. This allows us to post process the user's choice of a questions, and allows us to execute JS on focus.

```
911 \@ifpackageloaded{textpos}{\let\dps@mode\par}{\let\dps@mode\relax}
912 \def\afterQhookA#1{%
913   if(!event.target.isBoxChecked(0))dpsHideLayer("#1");\r
914   else\pdfSP if(PlayerSignIn())dpsShowLayer("#1");}
915 \def\OnFocusQhookAA#1{dpsHidePreviousLayer("#1")}
```

Parse the argument (⟨*name*⟩-⟨*num*⟩) to get the ⟨*name*⟩.

```
916 \def\dps@getOCGName#1-#2\@nil{\def\dps@OcgName{#1}}
```

`\fmtOCGQues{`⟨*various*⟩`}` Use this command to format the question layer. Within the argument of ⟨*various*⟩ you should place `\dpsQuesLayer{#1}`, which input the content file `\jobname-sc(#1).cut`. The following two examples use eso-pic and textpos packages respectively.

```
\fmtOCGQues{% eso-pic pkg
   \parbox[t][9\baselineskip][t]{2.25in}{\kern0pt\small\hfuzz11pt
   \psshadowbox[framesep=0pt]{\fcolorbox{red}{cornsilk}{%
   \parbox{\linewidth}{\dpsQuesLayer{#1}\vskip3pt}}}}}

\fmtOCGQues{% textpos pkg
  \parbox[t][9\baselineskip][t]{2.25in}{\kern0pt\small\hfuzz11pt
  \psshadowbox[framesep=0pt]{\fcolorbox{red}{cornsilk}{%
  \parbox{\linewidth}{\dpsQuesLayer{#1}\vskip3pt}}}}}
```

```
917 \def\fmtOCGQues#1{\def\fmtOCGQues@i##1{#1}}
```

`\dpsQuesLayer{(`⟨*name*⟩`-`⟨*num*⟩`)}` Inputs the appropriate content file. The command is placed in the `\fmtOCgQues` and its argument remains symbolic (`\dpsQuesLayer{#1}`), as seen in the examples above. The argument is never specified explicitly, but is only symbolically referenced with the `\placeQuesLayer` command; for example,

```
\placeQuesLayer{% eso-pic pkg
  \AddToShipoutPictureFG*{\AtTextCenter{\put(-72,72)
  {\insertQuesLayer{#1}}}}}

\placeQuesLayer{% textpos pkg
  \begin{textblock*}{2.25in}[0,0](2.5in+.725in,3in)
  \insertQuesLayer{#1}
  \end{textblock*}}
```

```
918 \def\dpsQuesLayer#1{\input{\jobname-sc(#1).cut}}
```

\placeQuesLayer {⟨*various*⟩} Use this command to place your question content on the page. The argument ⟨*various*⟩ depends on the package used (eso-pic or textpos. Examples are shown above under \dpsQuesLayer.

```
\long\def\placeQuesLayer#1{\@ifundefined{textblock}
  {\let\dps@mode\relax}{\let\dps@mode\par}%
  \def\pl@ceQuesL@yer##1{\dps@mode #1}}
```

\placeOtherLayer {⟨*various*⟩} A general purpose command for creating another layer, not associated with a question. The ⟨*various*⟩ argument is roll-your-own for inserting a graphic in a layer with a specific name. For example,

```
\placeOtherLayer{% eso-pic pkg
  \AddToShipoutPictureFG*{\AtTextCenter{\put(-72,36)
{\xBld{owclogo}\parbox{2.25in}
 {\includegraphics[width=\linewidth]{owc_self}}\eBld}}}}
```

The above creates a layer named owclogo consisting of a graphic. The placement is indicated as well.

```
\long\def\placeOtherLayer#1{#1}
```

\insertQuesLayer{⟨*name*⟩-⟨*num*⟩} Places the formatted question content (\fmtOCGQues@i) within a layer created by the \xBld/\eBld pair, the layer's name is ⟨*name*⟩ (parsed as \dps@OcgName).

```
\def\insertQuesLayer#1{\dps@getOCGName#1\@nil
  \edef\x{\noexpand\xBld{\dps@OcgName}}\x
  \fmtOCGQues@i{#1}\eBld}
```

\quesNumTxt
\quesNumTxTPost
\quesNumTxt and \quesNumTxTPost are the same, as described for button icon appearances.

```
\newcommand{\quesNumTxt}[1]{\protect\textbf{Problem #1}}
\newcommand{\quesNumTxTPost}{\protect\newline}
```

setContent{⟨*name*⟩} The setContent is the counterpart to the environment of the same name for button icon appearances. It performs a similar function, but yet is different. The setContent environment is placed within the cQ environment. The setContent follows the question prompt.

```
\newenvironment{setContent}[1]{%
     \gdef\scArg{#1}% save the argument for the end env
```

The \jobname-sc CUT file is indexed by ⟨*name*⟩-\theenumi. Thus, we have the ⟨*name*⟩ and problem number available to us.

```
  \def\CommentCutFile{\jobname-sc(#1-\theenumi).cut}%
  \immediate\openout\CommentStream=\CommentCutFile
  \begingroup
  \set@display@protect
  \let\verbatim@out\CommentStream
  \dps@IWVO{\quesNumTxt{\theenumi}\quesNumTxTPost}%
  \set@typeset@protect
  \verbatimwrite
}{%
  \endverbatimwrite
```

```
  \endgroup
  \immediate\closeout\CommentStream
  \edef\x{\noexpand\pl@ceQuesL@yer{\scArg-\theenumi}}\x
}
```

8.2.1 Support for a sideshow

`\insertSideshow[`⟨*ext*⟩`]{`⟨*rows*⟩`}{`⟨*cols*⟩`}[`⟨*hy-opts*⟩`]{`⟨*path*⟩`}` Command for placing the tiles of the picture. We assume that the pictures are numbered consecutively across rows.

⟨*ext*⟩ optional extension of the image

⟨*rows*⟩ number of rows

⟨*cols*⟩ number of columns

⟨*hy-opts*⟩ optional arguments for the `\includegraphics` command

⟨*path*⟩ base name of picture files (The files are index as follows: `mypic_01`, `mypic_02`, `mypic_03`, ... The basename is `mypic`, the underscore is added in by this command.

Usage:

```
\insertSideshow{3}{2}[width=.5\linewidth]{flowers2/rose}
```

```
\newcommand{\insertSideshow}[3][]{\begingroup
  \def\@Ext{#1}\ifx\@Ext\@empty\else\def\@Ext{.#1}\fi
  \def\@nrows{#2}\def\@ncols{#3}\insertSideshow@i
}
\newcommand\insertSideshow@i[2][]{\offinterlineskip
  \@tempcnta\z@
  \let\DPSIndxList\@gobble
  \multido{\iR=1+1}{\@nrows}{\hbox{%
    \multido{\iC=1+1}{\@ncols}{%
      \global\advance\@tempcnta\@ne
      \ifnum\@tempcnta<10\relax
        \edef\x{0\the\@tempcnta}\else\edef\x{\the\@tempcnta}\fi
        \edef\z{\noexpand\g@addto@macro\noexpand\DPSIndxList{,"\x"}}\z
        \xBld{pic\x}\includegraphics[#1]{#2_\x\@Ext}\eBld
    }% inner multido
  }}%  hbox, outer multido
```

Write the results of building the `\DPSIndxList` to the aux file. A typical result is `\gdef\DPSIndxList{"01","02","03","04","05","06"}`, which are the indices for the sideshow picture graphics.

```
  \immediate\write\@auxout{\string\gdef\string
    \DPSIndxList{\DPSIndxList}}%
  \endgroup
}
```

⟨/ocgadv⟩

9 Language Cutomizations

Below are the strings that are displayed in the message box. Wording may be changed to suite your needs.

(2020/04/21) Changed `( nMissed > n )` to `( nMissed > nPassing )` in the definition of `\congratFinished`.

```
⟨*english⟩
\def\chooseQ{"You must choose a question to answer before you answer!"}
\def\triedTooMuch{"You have tried this problem too many times,
  I'm adding "
  + \dspenaltypoints
  + " points, and resetting the penalty counter. Bad boy/girl!"}
\def\congratFinished{"Congratulations! You finished the puzzle"
  +((nMissed==0) ? " without missing a single problem, amazing!"
  : ", but you missed " + nMissed + " questions in the process!")}
\def\regretPleased{(( nMissed > nPassing )
  ? "I regret to report that you did not pass the test because
      you missed too many questions."
  : "I am pleased to report that you passed the test!")}
\def\reportPenaltyPoints{"The number of penalty points is "
  + nPenaltyPoints +"."}
\def\finalPenaltyScore{"Final penalty score is "
  + nTotalPenaltyPoints + ". "}
\def\aPenaltyScale{[-1,0], [0,4], [4, 10], [10,25], [25,5000]}
\def\aPenaltyMsgs
{%
  "Perfect!",
  "Very nice performance!",
  "This is not looking good. Perhaps a review is in order!",
  "Are you trying? No one could do so badly, you only need a
    seventh grade education!",
  "You're hopeless!"
}
\dlJSStr[noquotes]{\signInMsg}{%
  You must enter your name in the field at the top of
  the page to get credit for this assignment.}
⟨/english⟩
⟨*german⟩
\def\chooseQ{"Du musst erst eine Frage ausw\string\344hlen bevor
  Du antwortest!"}
\def\triedTooMuch{"Du hast es leider zu oft versucht, ich
  z\string\344hle  " + \dspenaltypoints
  + " Punkte dazu und setze den Z\string\344hler dann zur\string\374ck.
    Bitte streng Dich an!"}
\def\congratFinished{"Herzlichen Gl\string\374ckwunsch! Du hast das
  Puzzle beendet" + ((nMissed==0) ?
    " ohne auch nur einen Fehler zu machen, wunderbar!" :
    ", aber leider " + nMissed + " Mal falsch geantwortet!")}
\def\regretPleased{(( nMissed > nPassing )
```

```
  ? "Ich bedauere Dir mitteilen zu m\string\374ssen, dass Du den Test
    leider nicht bestanden hast, weil Du zu viele Fragen falsch
    beantwortet hast."
  : "Ich bin \string\344usserst erfreut Dir mitteilen
    zu d\string\374rfen, dass Du den Test bestanden hast!")}
\def\reportPenaltyPoints{"Die Anzahl der Strafpunkte ist  "
  + nPenaltyPoints +"."}
\def\finalPenaltyScore{"Die Gesamtanzahl der Strafpunkte ist damit "
  + nTotalPenaltyPoints + ". "}
\def\aPenaltyScale{[-1,0], [0,4], [4, 10], [10,25], [25,5000]}
\def\aPenaltyMsgs
{%
  "Perfekt!",
  "Sehr gute Vorstellung!",
  "Es sieht nicht sonderlich gut aus. Vielleicht w\string\344re
    eine Wiederholung gut!",
  "R\string\344tst Du eigentlich nur? Niemand kann wirklich so
    schlecht sein. Das ist Stoff aus der siebten Klasse!",
  "Du bist ein hoffnungsloser Fall!"
}
\dlJSStr[noquotes]{\signInMsg}{%
  You must enter your name in the field at the top of
  the page to get credit for this assignment.}
⟨/german⟩
⟨*package⟩
```

10 Form and document actions

`\printPDF[`⟨*opts*⟩`]{`⟨*wd*⟩`}{ht}` Opens the print dialog box, pre-populates some print parameters.

```
\newcommand{\printDPS}[3][]{%
  \pushButton[\CA{Print}\A{\JS{%
    var pp = this.getPrintParams();\r
    pp.firstPage=1;\r
    pp.lastPage=1;\r
    pp.pageHandling = pp.constants.handling.shrink;\r
    var fv = pp.constants.flagValues;\r
    pp.flags |= (fv.suppressCenter | fv.suppressRotate);\r
    this.print(pp);}}#1]{printDPS}{#2}{#3}%
}
```

`\resetPDF[`⟨*opts*⟩`]{`⟨*wd*⟩`}{ht}` Clears the puzzle board.

```
\newcommand{\resetDPS}[3][]{%
  \pushButton[\CA{Clear}\A{\JS{resetDPS();}}#1]{resetDPS}{#2}{#3}%
}
```

`\clearOnCloseOrSave` Try to prevent the student from saving the game (for some other student) we clear the game board if either student closes or save the document.

```
\def\dpsWCSWrnMsg{The file dps-wcs.def could not be found}
\newcommand{\clearOnCloseOrSave}{\InputIfFileExists{dps-wcs.def}{}
  {\PackageWarning{dps}{\dpsWCSWrnMsg}}}
```

⟨/package⟩
⟨∗willCloseSave⟩

This is the mechanism for preventing the student from saving the document and continuing at a later time. When the student tries to save or close the document, the entire puzzle board is cleared.

```
\begin{willClose}
resetDPS();
\end{willClose}
\begin{willSave}
resetDPS();
\end{willSave}
```

⟨/willCloseSave⟩

11 JavaScript Support

11.1 JavaScript common to all options

⟨∗package⟩

```
\def\lngthOfMsg{2000} % in milliseconds
```

`\dpsResetHook{`⟨*js-code*⟩`}` Can be used to add code lines to the `dpsReset()` function.

```
\def\dpsResetHook#1{\def\dpsresethook{#1}}
\dpsResetHook{;}
```

`\dpsFinishedEvent{`⟨*js-code*⟩`}` Can be used to add code lines to the `dpsFinishedHook()` function, a function that is called when the puzzle is complete.

```
\def\dpsFinishedEvent#1{\def\dpsfinishedevent{#1}}
\dpsFinishedEvent{;}
```

The main JavaScript segment for DPS

```
\begin{insDLJS}{match}{DPS: JavaScript support Das Puzzle Spiel}
var playerSignedIn = false;
var missesByQuestion = new Object();
var nPassing = \dspassing;
var nMissed = 0;
var nPenaltyPoints = 0;
var activeQuestion = "";
var f=this.getField("ckbxQ");
var g=f.getArray();
var QBC=g[0].strokeColor;
var pic = new Object();
var _dpsTO; // time out variable
var bRandPicMaps=\bRandPicMaps;
```

`PlayerSignIn()` Manages whether a player must sign into the `dpsSignInName` field. If this field does not exit, no sign-in is required.

```
function PlayerSignIn()
{
  if ( !playerSignedIn ) {
    var f = this.getField("dpsSignInName");
    if (f != null) {
      var nameField = f.value;
      if ( nameField.replace(/\s*/g,"") == "" ) {
        app.alert("\signInMsg");
        event.target.value="Off";
      } else
        playerSignedIn = true;
    } else playerSignedIn = true;
  }
  return playerSignedIn;
}
```

processChoice(⟨*name*⟩) This is the mouse up action of the answer check boxes.

```
function processChoice(name)
{
    // Get the question field that corresponds to this question,
    // see if checked.
    var f = this.getField("ckbxQ."+name);
    if ( (f != null) && (f.isBoxChecked(0)) ) { // right
        clearRedCrosses ();
        this.resetForm(["puzzle."+name]);
        event.target.textColor = ["RGB", 0, 0.6, 0];
        f.strokeColor = ["RGB", 0, 0.6, 0];
        f.readonly = true;
        event.target.readonly=true;
        try { afterCorrectChoiceHook() } catch(e) {};
        checkForFinished();
    } else { // wrong
        if ( activeQuestion != "" )
          var h = this.getField(activeQuestion);
        if ( (activeQuestion=="") || (h.readonly) ) {
            // active question already answered
            event.target.value = "Off";
            var g = this.getField("report");
            str = \chooseQ;
            g.value = str;
            var to = app.setTimeOut("clearMessages()", \lngthOfMsg);
        } else {
            event.target.style = style.cr;
            event.target.textColor = color.red;
            ++nMissed;
            if ( typeof missesByQuestion[activeQuestion] !="number" )
                missesByQuestion[activeQuestion] = 1;
            else
                missesByQuestion[activeQuestion] += 1;
            if (missesByQuestion[activeQuestion] > \dsthreshold) {
```

```
                var f = this.getField("report");
                str = \triedTooMuch;
                f.value = str;
                nPenaltyPoints += \dspenaltypoints;
                missesByQuestion[activeQuestion] = 0;
                clearRedCrosses ();
                var to = app.setTimeOut("clearMessages()", \lngthOfMsg);
            }
        }
    }
}
```

`clearMessages()` Clears the text field named `report`, which is created by `\placeMessageField`.

```
function clearMessages()
{
    var f = this.getField("report");
    f.value = "";
}
```

`clearRedCrosses()` Clears all answer check boxes what are marked with a cross, meaning wrong choice. Part of the mouse up action question check boxes.

```
function clearRedCrosses ()
{
    var g = this.getField("ckbxA");
    h = g.getArray();
    for ( i=0; i < h.length; i++) {
        if ( h[i].style == style.cr) h[i].checkThisBox(0,false);
        h[i].style = style.ch;
    }
 }
```

`checkForFinished()` Determines if all questions have been answered. Called from `processChoice()`.

```
function checkForFinished()
{
    var f = this.getField("puzzle");
    var g = f.getArray();
    var anyEmpty = false;
    for ( var i=0; i < g.length; i++) {
        if ( (g[i].name != "puzzle.space") && %
(g[i].value.replace(/\s/g,"") == "") ) {
            anyEmpty = true;
            break;
        }
    }
    var nTotalPenaltyPoints=nMissed + nPenaltyPoints;
    if ( !anyEmpty ) {
        try { dpsFinishedHook() } catch (e) {};
        var f = this.getField("report");
        str = \congratFinished
        + "\n" + \regretPleased
        + "\n" + \reportPenaltyPoints
        + "\n" + \finalPenaltyScore
```

```
1170        + " " + finalRating(nTotalPenaltyPoints);
1171        f.value = str;
1172    }
1173 }
```

`finalRating(`⟨*penalty-points*⟩`)` A function that returns a message string from the array (`\aPenaltyMsgs`) of penalty messages. Choice of messages is based on where the value ⟨*penalty-points*⟩ falls into the penalty scale array (`\aPenaltyScale`). This function is called from `checkForFinished()`.

```
1174 function finalRating(n) {
1175     var aPenaltyScale = new Array( \aPenaltyScale\space);
1176     var aPenaltyMsgs = new Array( \aPenaltyMsgs\space);
1177     for ( var i=0; i<aPenaltyScale.length; i++)
1178         if ( (n > aPenaltyScale[i][0]) && (n <= aPenaltyScale[i][1]) )
1179             return aPenaltyMsgs[i];
1180 }
```

`resetDPS()` A JavaScript function to reset the puzzle board. May be used as a push button action or within the `willClose` and `willSave` environments.

```
1181 function resetDPS() {
1182   this.delay=true;
1183   this.resetForm();
1184   var f=this.getField("puzzle");
1185   var g=f.getArray();
1186   for (var i=0; i<g.length; i++)g[i].value="";
1187   var f=this.getField("ckbxQ");
1188   f.strokeColor=QBC;
1189   var g=f.getArray()
1190   for (var i=0; i<g.length; i++)g[i].readonly=false;
1191   var f=this.getField("ckbxA");
1192   var g=f.getArray();
1193   for (var i=0; i<g.length; i++)g[i].readonly=false;
1194   this.dirty=false;
1195   if(typeof(xBlds)!="undefined"){
1196     for (var i=0; i<xBlds.length; i++) {
1197       var bName=xBlds[i].name.substring(3);
1198       toggleSetThisLayer(bName,false);
1199     }
1200   }
```

Hide any form fields with base name of `btnpic`.

```
1201   var f=this.getField("btnpic");
1202   if(f!=null)f.display=display.hidden;
1203   try {
```

`\dpsresethook` The hook (`\dpsresethook`) to add code lines during the reset.

```
1204     \dpsresethook
1205     if (typeof mixupDPS == "function") {
1206       mixupDPS();
1207       showDPS();
1208     }
1209   } catch(e){};
```

```
  playerSignedIn = false;
  missesByQuestion = new Object();
  nMissed = 0;
  nPenaltyPoints = 0;
  activeQuestion = "";
  aPicIndex=randomizePicList(aPicIndex,bRandPicMaps);
  if(bRandPicMaps)assignNamesToIndexes();
  this.delay=false;
}
```

11.1.1 JavaScript to support a sideshow

```
/*
    Create mapping from letters to pictures
*/
var aPicIndex=[\DPSIndxList];
var aNamesList=[\DPSNamesList];
var diff;
var PIlength=aPicIndex.length;
var NLlength=aNamesList.length;
if ( (diff=NLlength-PIlength) > 0 )
  for (var i=0; i<diff; i++)
    aPicIndex.push(null);
// now randomize aPicIndex, if requested
aPicIndex=randomizePicList(aPicIndex,bRandPicMaps);
// now make assignments
assignNamesToIndexes();
function assignNamesToIndexes() {
  for(var i=0; i< NLlength; i++)
    pic[aNamesList[i]]=aPicIndex[i];
}
function randomizePicList(array, bRandomize) {
  if (bRandomize) shuffle(array);
  return array;
}
```

The shuffle function was obtained from stackoverflow.[1]

```
function shuffle(array) {
  var currentIndex = array.length, temporaryValue, randomIndex;
  // While there remain elements to shuffle...
  while (0 !== currentIndex) {
    // Pick a remaining element...
    randomIndex = Math.floor(Math.random() * currentIndex);
    currentIndex -= 1;
    // And swap it with the current element.
    temporaryValue = array[currentIndex];
    array[currentIndex] = array[randomIndex];
    array[randomIndex] = temporaryValue;
  }
  return array;
```

[1] stackoverflow.com/questions/2450954/how-to-randomize-shuffle-a-javascript-array

```
}
\end{insDLJS}
```

11.2 JavaScript for the usebtnappr option

```
\ifusebtnappr
\begin{insDLJS}{dpsbtnappr}{DPS: Icon appearance support}
```

`dpsShowQues(`⟨*num*⟩`)` Part of the mouse-up action of the question check boxes. The function displays the hidden appearance icon for this question.

```
function dpsShowQues(n)
{
  if (PlayerSignIn()) {
    dpsShowFld("btnQ."+n);
%    var f=this.getField("btnQ."+n);
%    if (f!=null) f.display=display.visible;
  }
}
```

`dpsHideQFields()` Hides all icons fields with parent name of `btnQ`.

```
function dpsHideQFields() {
    dpsHideFld("btnQ");
%    var f=this.getField("btnQ");
%    f.display=display.hidden;
}
```

`dpsHideFinalField(`⟨*name*⟩`)` Hides the additional field that is displayed after player finishes. Typically this function is fired by the document author during the `dpsReset()` function.

`\dpsResetHook` Modification to this function are through the `\dpsResetHook` helper macro. For example,

```
\dpsResetHook{%
   dpsHideQFields();
   dpsHideFinalField("btnEmoji");
 }
```

Here, we hide all fields and hide the `"btnEmoji"` graphic.

```
function dpsHideFinalField(name){
  dpsHideFld(name);
%  var f=this.getField(name);
%  if (f!=null)f.display=display.hidden;
}
```

`dpsHidePreviousQues()` This function actually hides all fields, but only if there is an active question.

```
function dpsHidePreviousQues()
{
  if ( activeQuestion != "" ) {
    dpsHideQFields();
  }
}
function dpsShowFld(name) {
  var f=this.getField(name);
  if (f!=null)f.display=display.visible;
```

```
}
function dpsHideFld(name) {
  var f=this.getField(name);
  if (f!=null)f.display=display.hidden;
}

function displayPic(name){
%//  var name = activeQuestion.replace(/ckbxQ\./,"");
%//  var name = displayPic.name;
%  console.println("name: "+name);
%  console.println("pic[name]: "+pic[name]);
  if (pic[name] != null )dpsShowFld("btnpic."+pic[name]);
}
```

`afterCorrectChoiceHook()` After a correct answer, we hide all icon fields to get ready for the next question or final icon.

```
function afterCorrectChoiceHook()
{
  var name = activeQuestion.replace(/ckbxQ\./,"");
dpsHideQFields();
displayPic.name=name;
_dpsTO=app.setTimeOut("displayPic('"+name+"')",25);
}
```

`dpsFinishedHook()` This function fires when all questions have been answered. Called from `checkForFinished()`. This function executes a document authored function, `dpsfinishedevent`, if there is one defined.

`\dpsfinishedevent`

```
function dpsFinishedHook()
{
    dpsHideQFields();
    try {
      \dpsfinishedevent
      if(typeof sortoutDPS == "function") {
        ok2Continue = true;
        sortoutDPS();
      }
    }catch(e){}
}
\end{insDLJS}
```

Bubble sort the pics if `\sortPicMappings` is expanded in the preamble.

```
⟨/package⟩
⟨*sortjs⟩
\begin{insDLJS}{dpssort}{DPS: Bubble Sort}
// Global Data:
var hasBeenRandomized=false;
var btnbase="btnpic."; // btnpic.01, btnpic.02, etc
var iconbase="pic";    // pic01, pic02, etc.
var nTotalTiles=\dpsNumSideShowPics;
var randomDPS = new Array(nTotalTiles+1);
var ldps = randomDPS.length;
```

```
var timeout = 10;
var shutdown;
var debug = false; // memDebug;
var ok2Continue = true;
for (i=1; i<=nTotalTiles; i++) randomDPS[i]=i;
// Mixup DPS:
function mixupDPS()
{
    var i, rand, temp;
    for (i=1; i<= nTotalTiles; i++)
    {
        var rand = Math.random();
        rand *= ldps*ldps;
        rand = Math.ceil(rand);
        rand = rand \% ldps;
        if (rand == 0 ) rand = 1;
        temp = randomDPS[i];
        randomDPS[i]=randomDPS[rand];
        randomDPS[rand]=temp;
    }
}

// Show DPS:
function showDPS()
{
%    \sortCustomStartJS
    var I,J;
    for ( var i=1; i<=nTotalTiles; i++ )
    {
        I=((i<10)?"0":"")+i;
        J=((randomDPS[i]<10)?"0":"")+randomDPS[i];
        var oIcon = this.getIcon(iconbase+J);
%        var oIcon = oIconStreams[iconbase+J];
%        var oIcon = this.getIcon("pic."+randomDPS[i]);
        var f = this.getField(btnbase+I);
        f.buttonSetIcon(oIcon);
    }
}
// Sortout DPS:
function sortoutDPS()
{
    outerLoop(randomDPS.length-1);
}
function outerLoop(i)
{
     if ( ok2Continue && (i >= 0) ) %
shutdown = app.setTimeOut("app.clearTimeOut(shutdown); %
innerLoop("+i+",1);", timeout);
     else {
%        \sortCustomFinishJS
```

```
1377      }
1378 }
1379 function innerLoop(i,j)
1380 {
1381     var I, J;
1382     if ( j <= i )
1383     {
1384        if (randomDPS[j-1] > randomDPS[j])
1385        {
1386             var temp = randomDPS[j-1];
1387             randomDPS[j-1] = randomDPS[j];
1388             randomDPS[j] = temp;
1389             J=((randomDPS[j-1]<10)?"0":"")+randomDPS[j-1];
1390             I=((j-1 < 10)?"0":"")+(j-1);
1391             var oIcon = this.getIcon(iconbase+J);
1392             var f = this.getField(btnbase+I);
1393             f.buttonSetIcon(oIcon);
1394             J=((randomDPS[j]<10)?"0":"")+randomDPS[j];
1395             I=((j < 10)?"0":"")+j;
1396             var oIcon = this.getIcon(iconbase+J);
1397             var f = this.getField(btnbase+I);
1398             f.buttonSetIcon(oIcon);
1399         }
1400         j++
1401         if ( ok2Continue ) %
1402 shutdown = app.setTimeOut("app.clearTimeOut(shutdown); %
1403 innerLoop("+i+","+j+");", timeout);
1404         else {
1405 %            \sortCustomFinishJS
1406         }
1407     }
1408     else
1409     {
1410         i--;
1411         outerLoop(i);
1412     }
1413 }
1414 \end{insDLJS}
1415 ⟨/sortjs⟩
1416 ⟨*package⟩
1417 \fi
```

11.3 JavaScript for the `uselayers` option

We define similar functionality as described in Section 11.2.

```
1418 \ifuseocgappr
1419 \begin{insDLJS}{dpslayer}{DPS: Layer Support}
```

`dpsShowLayer(`⟨*name*⟩`)` Show the layer with a name of ⟨*name*⟩. The JavaScript function `toggleSetThisLayer` `toggleSetThisLayer()` is defined in aeb_pro with the `uselayers` option.

```
function dpsShowLayer(name)
{
    toggleSetThisLayer(name, true);
}
```

dpsHidePreviousLayer() Hides this previous layer, as recored by activeQuestion. The value of activeQuestion is of the form ckbxQ.⟨*name*⟩. We strip off the first 6 letters in this string.

```
function dpsHidePreviousLayer()
{
    if ( activeQuestion != "" ) {
%       var activename = activeQuestion.replace(/ckbxQ\./,"");
        var activename=activeQuestion.substring(6)
        dpsHideLayer(activename);
    }
}
```

dpsHideLayer(⟨*name*⟩) Hide the layer with a name of ⟨*name*⟩. The function toggleSetThisLayer()
toggleSetThisLayer is defined in aeb_pro with the uselayers option.

```
function dpsHideLayer(name)
{
    toggleSetThisLayer(name, false);
}
```

afterCorrectChoiceHook() A function that fires after a correct answer is registered.

```
function afterCorrectChoiceHook()
{
    var name = activeQuestion.replace(/ckbxQ\./,"");
    try {
```

Used for special effects.

```
      if (pic[name] != null )
      dpsShowLayer("pic"+pic[name]);
    } catch(e) {};
    dpsHideLayer(name);
}
```

dpsFinishedHook() This function fires when all questions have been answered. Called from checkForFinished(). This function executes a document authored function,
\dpsfinishedevent dpsfinishedevent, if there is one defined.

```
function dpsFinishedHook()
{
    var name = activeQuestion.replace(/ckbxQ\./,"");
    dpsHideLayer(name);
    try {
      \dpsfinishedevent
    }catch(e){}
}
\end{insDLJS}
\fi
\dps@restoreCats
```

```
\let\WriteBookmarks\relax
```
⟨/package⟩

12 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.

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

13 Change History