

# The **dps** Package

D. P. Story

Email: dpstory@acrotex.net

processed June 9, 2020

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Options</b>	<b>2</b>
<b>3</b>	<b>Required Packages</b>	<b>4</b>
<b>4</b>	<b>Switches and things</b>	<b>7</b>
<b>5</b>	<b>Building the Puzzle</b>	<b>7</b>
5.1	Declaring the puzzle . . . . .	7
5.2	The underlying text fields of the puzzle . . . . .	8
5.3	Inserting the puzzle into the document . . . . .	10
5.4	The answer key . . . . .	12
5.5	Support for a sideshow . . . . .	13
<b>6</b>	<b>Questions and Answers</b>	<b>14</b>
<b>7</b>	<b>Miscellaneous Settings</b>	<b>22</b>
<b>8</b>	<b>Advanced features</b>	<b>23</b>
8.1	Icon button appearances . . . . .	23
8.1.1	Macros for the main puzzle document . . . . .	23
8.1.2	Macros for the icon document . . . . .	27
8.2	OCG methods . . . . .	31
8.2.1	Support for a sideshow . . . . .	34
<b>9</b>	<b>Language Cutomizations</b>	<b>35</b>
<b>10</b>	<b>Form and document actions</b>	<b>36</b>

<b>11 JavaScript Support</b>	<b>37</b>
11.1 JavaScript common to all options . . . . .	37
11.1.1 JavaScript to support a sideshow . . . . .	41
11.2 JavaScript for the <code>usebtnappr</code> option . . . . .	42
11.3 JavaScript for the <code>uselayers</code> option . . . . .	45
<b>12 Index</b>	<b>48</b>
1 <code>{*package}</code>	

## 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}
<code>nonrandomized</code>	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 <code>!nonrandomized</code> to ‘cancel’ <code>nonrandomized</code> .
	3 \DeclareOptionX{nonrandomized}{\werandomizefalse}
	4 \DeclareOptionX{!nonrandomized}{\werandomizetrue}
<code>viewmode</code>	Used for developing the puzzle. When <code>viewmode</code> is optioned, the puzzle and letters in the puzzle are seen. By adjusting the argument of <code>\insertPuzzle</code> you can get the distribution of the puzzle that you want. See also the comments preceding the definition of <code>\makeTextField</code> below. The <code>!viewmode</code> option cancels the <code>viewmode</code> option.
	5 \DeclareOptionX{viewmode}{\viewModetrue\previewtrue}
	6 \DeclareOptionX{!viewmode}{\viewModefalse\previewfalse}
<code>showletters</code>	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 <code>forpaper</code> mode, but can be used in “screen” mode. The convenience option of <code>!showletters</code> is also provided.
	7 \DeclareOptionX{showletters}{\showletterstrue}
	8 \DeclareOptionX{!showletters}{\showlettersfalse}

<code>showanswerlabels</code>	We give the user the option of showing the label for the answer, in the case of for screen presentation.
	<pre> 9 \DeclareOptionX{showanswerlabels} 10 {% 11     \ifeqforpaper\else\def\dpsAitemOptArg{} \fi 12 } 13 \DeclareOptionX{!showanswerlabels}{% 14     \def\dpsAitemOptArg{[]} 15 \def\dpsAitemOptArg{[]} </pre>
<code>showanswerkey</code>	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 <code>\AnswerKey</code> . Selecting <code>showanswerkey</code> also activates the <code>showletters</code> option. The convenience option <code>!showanswerkey</code> is also provided.
<code>!showanswerkey</code>	<pre> 16 \DeclareOptionX{showanswerkey}{\showsolutiontrue 17   \ExecuteOptionsX{showletters}} 18 \DeclareOptionX{!showanswerkey}{\showsolutionfalse} </pre>
<code>usebtnappr</code>	<b>Options for posing questions.</b> There are two methods of posing longer questions: (1) ( <code>usebtnappr</code> ) Place these questions in the appearance of a button; or (2) ( <code>uselayers</code> ) Place the questions in their own layer (OCG). Only one of these two options is allowed. Neither option is also permitted (short questions only).
<code>usebtnappr</code>	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.
<code>uselayers</code>	<pre> 19 \DeclareOptionX{usebtnappr}{\usebtnapprtrue 20   \ifuseocgappr 21     \PackageWarningNoLine{dps} 22       {Options usebtnappr and uselayers both used.\MessageBreak 23        Will use the uselayers option}% 24     \usebtnapprfalse 25     \let\dpsInputBtnAppr\relax 26   \else 27     \def\dpsInputBtnAppr{\InputIfFileExists{usebtnappr.def}{}{}} 28   \fi} 29 \let\dpsInputBtnAppr\relax </pre>
<code>uselayers</code>	A required option when you want to pose longer questions. Longer questions are placed in OCG (layers). This option brings in supporting commands.
<code>uselayers</code>	<pre> 30 \DeclareOptionX{uselayers}{\useocgapprtrue 31   \ifusebtnappr 32     \PackageWarningNoLine{dps} 33       {Options usebtnappr and uselayers both used.\MessageBreak 34        Will use the usebtnappr option}% 35     \useocgapprfalse 36     \let\dpsInputOcgAppr\relax 37   \else </pre>

```

38     \def\dpsInputOcgAppr{\InputIfFileExists{useocgappr.def}{}{}}
39   \fi}
40 \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.
41 \DeclareOptionX{savedata}{\savepuzzledatatrue}
42 \DeclareOptionX{!savedata}{\savepuzzledatafalse}
Various switches used by this package
43 \newif\ifwerandomize \werandomizetrue
44 \newif\ifviewMode \viewModefalse
45 \newif\ifshowletters\showlettersfalse
46 \newif\ifshowsolution\showsolutionfalse
47 \newif\ifsavepuzzledata\savepuzzledatafalse
48 \newif\ifusebtnappr \usebtnapprfalse
49 \newif\ifuseocgappr \useocgapprfalse
(20/06/03) New default is is \wrtContenttrue.
50 \newif\ifwrtContent\wrtContenttrue
If a paper option is taken, we show the letters.
51 \ifeqforpaper\showletterstrue\fi
lang The only language localizations are the clever commands that appear in the mes-
sage 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.
52 \define@choicekey*+{dps.sty}{lang}[\val\nr]{english,german,custom}
53 {%
54   \ifcase\nr\relax
55     \def\dps@lang@type{\input{dps_str_us.def}}\or
56     \def\dps@lang@type{\input{dps_str_de.def}}\or
57     \def\dps@lang@type{\input{dps_str_cus.def}}\else
58     \def\dps@lang@type{\input{dps_str_us.def}}\fi
59 }{\PackageWarning{dps}{Bad choice for lang, permissible values
60 are english, german and custom. Try again}}
61 \def\dps@lang@type{\input{dps_str_us.def}}
62 \AtEndOfPackage{\dps@lang@type}
63 \ProcessOptionsX
64 \edef\dps@restoreCats{%
65   \catcode`\noexpand\"=\the\catcode`\"
66   \catcode`\noexpand\,-\the\catcode`\,\relax
67   \catcode`\noexpand\_=\the\catcode`\_\relax
68 }
69 \makeother\" \makeother\, \makeother\_

```

### 3 Required Packages

```

70 \RequirePackage{web}
71 \RequirePackage{eforms}
72 \ifxetex\makeXasPDOff\fi
73 \RequirePackage{graphicx}

In addition to the web and eforms packages, the following are used in the macro
package.
74 \RequirePackage{verbatim}
75 \RequirePackage{calc}
76 \RequirePackage{multicol}
77 \RequirePackage{multido}
78 \hypersetup{pdfencoding=pdfdoc}

Input JavaScript for the usebtnappr option
79 \ifusebtnappr
80   \def\x{\AtEndOfPackage{\dpsInputBtnAppr}}%
81   \expandafter\x\fi

Input JavaScript for the uselayers option
82 \ifuseocgappr
83   \def\x{\AtEndOfPackage{\dpsInputOcgAppr}}%
84 \expandafter\x\fi

For usebtnappr, we require icon-appr to embed the graphics, and make them
appearances of buttons.
85 \ifusebtnappr
86   \def\dps@RP{\RequirePackage{icon-appr}[2020/06/05]}
87   \expandafter\dps@RP
88 \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.
89 \def\dps@ckForpglst{\IfFileExists{icons-pglst.sav}%
90   {\global\wrtContentfalse}{}%
91   \ifusebtnappr
92     \ifwrtContent
93       \global\savepuzzledatatrue
94     \else
95       \global\savepuzzledatafalse
96     \fi
97   \fi
98 }
99 \ifusebtnappr
100 \def\dps@emitEOP{\AtEndOfPackage{\dps@ckForpglst}}
101 \expandafter\dps@emitEOP\fi

```

**Input random.tex.** Input and make modifications.

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

We modify `\nextrandom` to save the startup seed.

```

103 \def\dps@nextrandom{%
104   \def\nextrandom{\begingroup
105     \ifnum\randomi<\@ne % then initialize with time
106       \global\randomi\time
107       \global\multiply\randomi388 \global\advance\randomi\year
108       \global\multiply\randomi31 \global\advance\randomi\day
109       \global\multiply\randomi97 \global\advance\randomi\month
110       \message{Randomizer initialized to \the\randomi.}%
111       \nextrandom \nextrandom \nextrandom
112       \xdef\ds@saveRandomSeed{\the\randomi}%
113     \fi
114     \count@ii\randomi
115     \divide\count@ii 127773 % modulus = multiplier * 127773 + 2836
116     \count@\count@ii
117     \multiply\count@ii 127773
118     \global\advance\randomi-\count@ii % random mod 127773
119     \global\multiply\randomi 16807
120     \multiply\count@ 2836
121     \global\advance\randomi-\count@
122     \ifnum\randomi<\z@ \global\advance\randomi 2147483647\relax\fi
123   \endgroup
124 }
125 }
126 \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 it is first used in the preamble. The command `\redefnextrandomAsNeeded` appears in the `\ds@randomizeList`, which is where all randomization occurs.

```

127 \def\redefnextrandomAsNeeded{\ifnextrandomredefd\else
128   \global\let\nextrandom\dps@nextrandom
129   \global\nextrandomredefdtrue\fi}

```

`\useRandomSeed{<pos-num>}` Use the number `<pos-num>` as the initial seed.

```

130 \def\useRandomSeed#1{\randomi=#1
131   \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.

```

132 \def\inputRandomSeed{\ifwerandomize
133   \InputIfFileExists{\jobname_data.sav}{}{}%
134   \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 exist, the seed used is based on the date and time.

```

135 \def\dpsLastSeed#1{\def\dps@LastSeed{#1}}
136 \def\useLastSeed{\ifwrandomize
137   \InputIfFileExists{\jobname_data.sav}{}{}%
138   @ifundefined{dps@LastSeed}{}{%
139     {\randomi=\dps@LastSeed\relax}%
140   \edef\ds@saveRandomSeed{\the\randomi}\fi
141 }

```

## 4 Switches and things

```

142 \newif\ifforquestions \forquestionstrue
143 \newcount\ds@nCnt
144 \newcount\ds@nMax
145 \newcount\ds@qNumber\ds@qNumber=0
146 \newcount\ds@aNumber\ds@aNumber=0
147 \newcount\ds@probCnt \ds@probCnt=0
148 \newcount\ds@nCntCols \ds@nCntCols=0
149 \newcount\ds@getRanNum
150 \newtoks\ds@listIn \ds@listIn={}
151 \newtoks\ds@ newListIn \ds@ newListIn={}
152 \newtoks\ds@listOut \ds@listOut={}
153 \newtoks\ds@tmpToks \ds@tmpToks={}
154 \newtoks\ds@qlistOut \ds@qlistOut={}
155 \newtoks\ds@alistOut \ds@alistOut={}
156 \newtoks\ds@PuzzleAppearancetoks \ds@PuzzleAppearancetoks={}
157 \newtoks\ds@QuesAppearancetoks \ds@QuesAppearancetoks={}
158 \newtoks\ds@AnsAppearancetoks \ds@AnsAppearancetoks={}
159 \newwrite\ds@question@write
160 \setlength{\multicolssep}{\topsep}
161 \def\csarg#1#2{\expandafter#1\csname #2\endcsname}
162 \let\dps@One=1 \let\dps@Zero=0

```

## 5 Building the Puzzle

### 5.1 Declaring the puzzle

\DeclarePuzzle{\langle puzzle-arg \rangle}, where \langle puzzle-arg \rangle is a series of *pairs of arguments*.

```

\DeclarePuzzle{%
  {\langle letter_1 \rangle}{\langle name_1 \rangle}
  {\langle letter_2 \rangle}{\langle name_2 \rangle}
  ...
  ...
  {\langle letter_n \rangle}{\langle name_n \rangle}
}

```

The argument \langle letter \rangle represents a letter in the puzzle; \langle letter \rangle 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.

```
163 \def\DeclarePuzzle#1{%
164   \gdef\puzzleParameters{#1}%
165   \let\DPSPNamesList\@gobble
166   \dps@getNames#1\relax\relax
167 }
168 \def\dps@getNames{\begingroup\dps@getNames@i}
169 \def\dps@getNames@i#1#2{%
170   \ifx#2\relax\let\getNextN@me\endgroup
171   \else\let\getNextN@me\dps@getNames@i
```

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

```
172   \def@rgii{#2}\ifx@\rgii\ds@myspace\else
173     \ifx@\rgii\ds@punc\else
174       \ifx@\rgii\ds@cr\else
175         \@ifundefined{ds@name@#2}{\g@addto@macro\DPSPNamesList{,#2}%
176           \csarg\let{ds@name@#2}\@empty}{}%
177         \fi\fi\fi
178     \fi
179   \getNextN@me
180 }
```

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

```
181 \def\nPuzzleCols#1{\def\nCols{#1}}
182 \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,  $\langle KV-pairs \rangle$  is a set of eforms key-value pairs.

```

183 \def\PuzzleAppearance#1{\ds@PuzzleAppearence{#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
\wdPuzzleFields on the options. Control the width of the fields with \wdPuzzleFields{\width}
\htPuzzleFields and the height with \htPuzzleFields{\height}.
184 \def\wdPuzzleFields#1{\bgroup\setlength{\tempdima{#1}}%
185   \xdef\wd@fPF{\the\tempdima}\egroup}
186 \def\wd@fPF{1.6em}
187 \def\htPuzzleFields#1{\bgroup\setlength{\tempdima{#1}}%
188   \xdef\ht@fPF{\the\tempdima}\egroup}
189 \def\ht@fPF{11bp}
190 \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.
191 \ifeqforpaper
192   \ifviewMode
193     \def\ds@makeTextField#1#2{\def\argii{#2}\ifx\argii
194       \ds@mymspace\let\ds@ul\relax\else\let\ds@ul\underbar\fi
195       \ifshowletters\raisebox{-10pt}{%
196         \makebox[0pt][l]{\makebox[\wd@fPF][c]{\footnotesize
197           \ds@getProbNumber{#1}{#2}}}}\fi
198       \ds@ul{\makebox[\wd@fPF][c]{\dps@strut
199         \Hy@pdfstringfalse{#1}}}
200   \else
201     \def\ds@makeTextField#1#2{\def\argii{#2}\ifx\argii\ds@mymspace
202       \let\ds@ul\relax\else\let\ds@ul\underbar\fi
203       \ifshowletters\raisebox{-10pt}{%
204         \makebox[0pt][l]{\makebox[\wd@fPF][c]{\footnotesize
205           \ds@getProbNumber{#1}{#2}}}}\fi
206       \ds@ul{\makebox[\wd@fPF][c]{\dps@strut\hfil}}}
207   \fi
208 \else
209   \ifviewMode
210     \def\ds@makeTextField#1#2{\ifshowletters\raisebox{-10pt}{%
211       \makebox[0pt][l]{\makebox[\wd@fPF][c]{\footnotesize
212         \ds@getProbNumber{#1}{#2}}}}\fi
213     \underbar{\makebox[\wd@fPF][c]{\dps@strut
214       \Hy@pdfstringfalse{#1}}}
215   \else
216     \def\ds@makeTextField#1#2{\ifshowletters
217       \raisebox{-10pt}{\makebox[0pt][l]{\makebox[\wd@fPF][c]{\footnotesize
218         \ds@getProbNumber{#1}{#2}}}}\fi
219     \edef\textfieldTmp{\noexpand\textField[\noexpand\Q{1}%
220       \noexpand\S{U}\noexpand\Ff\noexpand\FFReadOnly
221       \the\ds@PuzzleAppearence{#1}]\%
222       {puzzle.#2}\{\wd@fPF\}\{\ht@fPF\}\Hy@pdfstringtrue\textfieldTmp}
```

```
223 \fi
224 \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.

```
225 \def\dps@emitAK{0}
226 \def\dps@AddToEmitAK#1{\bgroup
227   @_tempcnta=\dps@emitAK\relax
228   \advance @_tempcnta by#1\relax
229   \xdef\dps@emitAK{\the @_tempcnta}\egroup
230 }
231 \def\dps@ckEmitAK{\ifnum\dps@emitAK>\thr@@\expandafter
232   \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.

```
233 \def\insertPuzzle#1{\begingroup\def @_rgi{#1}%
234   \ifx @_rgi\empty
235     \ifx \nCols\empty
236       \PackageWarning{\dps}{\string\insertPuzzle\space
237         needs an argument, use\MessageBreak
238         either \string\insertPuzzle{<nCols>} or\MessageBreak
239         declare \string\nPuzzleCols{<nCols>} in preamble.\MessageBreak
240         Setting <nCols> to 10 for now}\gdef\nCols{10}%
241     \fi
242   \else
243     \xdef\nCols{#1}%
244   \fi
245   \ifviewMode\Hy@pdfstringfalse\else\Hy@pdfstringtrue\fi
246   \let\tops\texorpdfstring\expandafter\dps@Puzzle
247   \expandafter{\puzzleParameters}\endgroup
248   \dps@AddToEmitAK{1}\dps@ckEmitAK
249 }
```

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

```
250 \def\rowsep#1{\gdef @_rowsep{[#1]}\gdef @_rowskip{#1}}
251 \rowsep{\rowsep@default}
```

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

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

```
253 \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`

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

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

```
256 \def\dps@@Puzzle{\begin{tabular}%
257 {@{}*{\nCols}{l@{}{}}}\@dpsPuzzlei%
258 }
```

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

```
259 \def\@dpsPuzzlei#1#2{\ifx#2\relax
260 \gdef\nextPuzzleChar{\@dpsPuzzleDone}\else
261 \gdef\nextPuzzleChar{\@dpsPuzzlei{#1}{#2}}\fi
262 \nextPuzzleChar
263 }
```

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`).

```
264 \def\@dpsPuzzlei#1#2{\def\argii{#2}%
265 \ifx\argii\ds@punc
266 % so something with punc
267 \def\@puzzNext{\#1\@takeaPeek}@\setSep@dpsPuzzlei}%
268 \else\ifx\argii\ds@cr
269 % do something with cr
270 \def\@puzzNext{\global\ds@nCntCols\z@
271 \eq@tabEnd\@dpsPuzzlei}%
272 \else
273 % ok to make a field
274 \def\@puzzNext{\ds@makeTextField{#1}{#2}\@takeaPeek}%
275 \fi\fi
276 \@puzzNext
277 }
278 \def\ds@cr{cr}
279 \def\@takeaPeek#1#2{\def\argii{#2}%
280 \ifx\argii\ds@punc
```

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

```
281 \def\@puzzNext{\#1\@takeaPeek}%
282 \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.

```

283     \ifx\argii\ds@cr
284         \def\@puzzNext{\global\ds@nCntCols\z@
285             \eq@tabEnd@\dpsPuzzlei}%
286     \else
287         \def\@puzzNext{\@setSep@dpsPuzzleii{#1}{#2}}%
288     \fi
289 \fi
290 \@puzzNext
291 }
292 \def\@setSep@dpsPuzzlei{ \global\advance\ds@nCntCols\@ne
293   \ifnum\ds@nCntCols=\nCols\relax
294     \expandafter\eq@tabEnd
295     \global\ds@nCntCols\z@\else
296     \expandafter\eq@tabSep\fi
297   \dpsPuzzlei}
298 \def\@dpsPuzzleDone{\end{tabular}\ifnum\ds@nCntCols=0\relax
299   \ifshowletters\vskip\@rowskip\relax
300   \else\vskip3pt\fi
301   \vskip-2\baselineskip
302 \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`.

```

303 \def\AnswerKey{The answer key is not available.\PackageWarning{dps}
304   {The showletters option is required to generate\MessageBreak
305   an answer key,}}
306 \def\ds@buildAnswerKey
307 {%
308   {%
309     \Hy@pdfstringfalse
310     \let\tops\texorpdfstring
311     \let\protect\unexpandable@protect
312     \count@\z@\toks@={}%
313     \loop
314       \advance\count@\@ne
315       \edef\y{\@nameuse{dps@probLetterii}\the\count@}%
316       \edef\x{\@nameuse{dps@probLetterKey}\y}%
317       \expandafter\ifx\x\relax\else
318         \csarg\xdef{dps@probLetter}\the\count@{\expandafter\noexpand

```

```

319      \csname dps@probLetterAlt\y\endcsname}\fi
320      \xdef\AnswerKey{\the\toks@
321      \the\count@--\c@nameuse{dps@probLetter}{\the\count@}; }%
322      \toks@=\expandafter{\AnswerKey}%
323      \ifnum\count@ < \ds@qNumber\repeat
324  }%
325 \ifshowsolution
326   \cfooter{\let\tops\texorpdfstring
327   \footnotesize\@ifundefined{rotatebox}{\AnswerKey}%
328   {\rotatebox{180}{\AnswerKey}}}\fi
329 }
330 \def\ds@writePuzzleData{\ifsavepuzzledata
331   \newwrite \ds@savedata
332   \begingroup
333   \immediate\openout \ds@savedata \jobname_data.sav
334   \def\msg{i}{Initial seed}
335   \def\msgii{Answer key:}
336   \let\verbatim@out\ds@savedata
337   \uccode`c='%\uppercase{%
338     \ifwerandomize
339       \dps@IWW0{\string\randomi=\ds@saveRandomSeed\space
340         c \msg}\%
341       \dps@IWW0{\string\dpsLastSeed{\the\randomi}}\fi
342     \ifshowletters\let\tops\texorpdfstring
343       \set@display@protect
344       \dps@IWW0{c \msgii\space\AnswerKey}\fi
345   }
346   \immediate\closeout\ds@savedata
347   \endgroup
348 \fi}
349 \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`.

```

350 \def\setdpsfootskip#1{\bgroup
351   \setlength\@tempdima{#1}\ifeqforpaper\else
352   \xdef\web@footskip{\the\@tempdima}\fi
353   \egroup\InitLayout % a web command
354 }
355 \setdpsfootskip{.25in}

```

## 5.5 Support for a sideshow

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.

`\randomizePicMappings`

```

356 \let\DPSSIndxList\@empty
357 \let\DPSSNamesList\@empty
358 \def\randomizePicMappings{\def\bRandPicMaps{true}}
359 \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.
360 \ifusebtnappr
361   \def\sortPicMappings{%
362     \InputIfFileExists{sortjs.def}{}{%
363       \OpenAction{\JS{try{if(!hasBeenRandomized)
364         {hasBeenRandomized=true;mixupDPS();showDPS();}
365       }catch(e){}}}}%
366   }
367 \else
368   \let\sortPicMappings\relax
369 \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).
370 \newif\if@isPackaged \@isPackagedfalse
371 \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.

```

372 \newenvironment{Composing}
373 {\global\ds@qNumber=0 \global\ds@aNumber=0}
374 {\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.

```

375 \newenvironment{cQ}[1]
376 {%
377   \global\advance\ds@qNumber\@ne
378   \immediate\openout \ds@question@write \jobname_q\the\ds@qNumber.cut
379   \let\verbatim@out\ds@question@write\set@display@protect
380   \dps@IWV0{\protect\dpsQ{#1}}%

```

```

381     \set@typeset@protect
382     \verbatimwrite
383 }{%
384     \endverbatimwrite
385     \immediate\closeout \ds@question@write
386 }

```

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.

```

387 \newenvironment{cA}[2] []
388 {%
389     \def\argi{\global\advance\ds@aNumber@ne
390     \immediate\openout \ds@question@write \jobname_a\the\ds@aNumber.cut
391     \let\verbatim@out\ds@question@write
392     \set@display@protect
393     \dps@IWV0{\protect\dpsA\ifshowletters
394         \ifx\argi\empty\else[\#1]\fi\fi{#2}}%
395     \set@typeset@protect
396     \verbatimwrite
397 }%
398     \endverbatimwrite
399     \immediate\closeout \ds@question@write
400 }

```

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

```

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

```

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

\afterQhookA      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.

```
403 \def\afterQhookA#1{if(PlayerSignIn());}
404 %\let\afterQhookA\@gobble
405 \let\OnFocusQhookAA\@gobble
```

\widestFmtQNum{\text} Sets the width of the underlying checkbox for the question.

```
406 \def\widestFmtQNum#1{\bgroup
407   \settowidth{\tempdima}{#1}%
408   \ifxetex\advance\tempdima2.5bp\else
409     \advance\tempdima.5pt\fi
410   \xdef\Qwidth{\the\tempdima}\egroup
411 \widestFmtQNum{00.}
412 \def\htOfQ#1{\setlength{\tempdima}{#1}\ifxetex
413   \advance\tempdima2bp\relax\fi\edef\Qht{\the\tempdima}}
414 \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.

```
415 \newcommand{\dpsQ}[1]{\item\relax\ifeqforpaper\else
416   \edef\checkboxTmp{\noexpand\checkBox[\the\ds@QuesAppearence]toks
417   \noexpand\textSize{0}\noexpand
418   \noexpand\JS{activeQuestion = event.target.name;\noexpand
419   \r clearRedCrosses();\noexpand\r\noexpand\afterQhookA{#1}}\%
420   \noexpand\AA{\noexpand\AAOnFocus{\noexpand\JS{%
421   this.resetForm(["ckbxQ"]); \noexpand\r\noexpand
422   \OnFocusQhookAA{#1}}}}\{ckbxQ.#1\{\Qwidth\{\Qht\}{Yes}\}\%
423   \makebox[0pt][r]{\strut
424     \smash{\checkboxTmp}\efKern{-.5pt}{-1.5pt}\enspace}\fi
425   \ignorespaces
426 }
```

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.

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

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

```

429 \def\widestFmtALtr#1{\bgroup
430   \settowidth{\@tempdima}{#1}\ifxetex
431     \addtolength{\@tempdima}{2bp+8pt}\else
432     \addtolength{\@tempdima}{8pt}\fi
433   \xdef\Awidth{\the\@tempdima}\egroup}
434 \widestFmtALtr{w}
435 \def\htOfA#1{\setlength{\@tempdima}{#1}\ifxetex
436   \advance\@tempdima2bp\relax\fi\edef\Aht{\the\@tempdima}}
437 \htOfA{13bp}

\dpSA Checkbox for the answer.
438 \newcommand{\dpSA}[2][]{\expandafter
439   \item\dpSAitemOptArg\relax
440   \ifeqforpaper\else
441     \edef\checkboxTmp{\noexpand
442       \checkBox[\the\ds@AnsAppearenceTok
443         \noexpand\textSize{0}\noexpand
444           \A{\noexpand\JS{processChoice("#2");}}]%
445           \cboxA.#2]{\Awidth}{\Aht}{Yes}}%
446     \makebox[0pt][r]{\strut
447       \smash{\checkboxTmp}\enspace}%
448   \fi
449   \def\argi{#1}%
450   \bgroup % dps
451     \let\tops\texorpdfstring
452     \Hy@pdfstringfalse
453     \ifx\argi\empty
454       \global\csarg\let\dpSAprobLetterKey#2\relax
455     \else
456       \csarg\gdef\dpSAprobLetterKey#2{#2}%
457       \csarg\gdef\dpSAprobLetterAlt#2{#1}%
458     \fi
459   \egroup
460   \ifshowletters
461     \ifx\argi\empty
462       \let\getLetterNext\relax
463       \let\ds@foundLetter\dpSAzero\def\ds@currFN{#2}%
464       \def\getLetterNext{\ds@typesetPuzzleLetter}%
465       \expandafter\getLetterNext
466     \else
467       \makebox[0pt][r]{\let\tops\texorpdfstring
468         \Hy@pdfstringfalse\@ltrFmtA{#1}\enspace\kern4bp}%
469     \fi
470   \fi
471   \ignorespaces
472 }

473 \def\ds@typesetPuzzleLetter{\expandafter
474   \typeset@PuzzleLetter\expandafter{\puzzleParameters}}
475 \def\typeset@PuzzleLetter#1{\typeset@@PuzzleLetter#1\relax\relax}%

```

```

476 \def\typeset@@PuzzleLetter#1#2{%
477   \ifx#1\relax
478     \gdef\nextPuzzleLetter{\relax}\else
479     \gdef\nextPuzzleLetter{\typeset@PuzzleLetteri{#1}{#2}}\fi
480   \nextPuzzleLetter
481 }
482 \def\typeset@PuzzleLetteri#1#2{\def\argii{#2}\ifx\argii\ds@currFN
483   \ifx\ds@foundLetter\dps@Zero
484     \makebox[0pt][r]{\let\tops\texorpdfstring
485     \Hy@pdfstringfalse@{\ltrFmtA{#1}\enspace\kern4bp}}%
486     \let\ds@foundLetter\dps@One\fi
487     \expandafter\typeset@@PuzzleLetter
488   \else
489     \expandafter\typeset@@PuzzleLetter
490   \fi
491 }

```

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.

```

492 \def\ds@publishRandomLists{%
493   \ds@randomizeQuestionList{\the\ds@qNumber}%
494   \ds@randomizeAnswerList{\the\ds@aNumber}%
495 }
496 \def\ds@myspace{space}
497 \def\ds@getProbNumber#1#2{%
498   \gdef\ds@currentArgi{#1}\gdef\ds@currentArgii{#2}%
499   \ifx\ds@currentArgi\ds@myspace\def\ds@probNumNext{\relax}\else
500     \def\ds@probNumNext{%
501       \ifundefined{dps@probNum#2}{\global\advance\ds@probCnt\@ne
502         \ds@getProbNumber{\the\ds@probCnt}%
503       }{\csname dps@probNum#2\endcsname}%
504     }%
505   \fi
506   \ds@probNumNext
507 }
508 \def\ds@getProbNumber#1{%
509   \count@z@ \let\\=\ds@getNthOne\the\ds@qlistOut}%
510   \csname dps@probNum\ds@currentArgi\endcsname
511 }
512 \def\ds@getNthOne#1{\advance\count@\@ne
513   \ifnum\ds@probCnt=#1\relax
514     \csarg\xdef{dps@probNum\ds@currentArgii}{\the\count@}%
515     \toks@=\expandafter{\ds@currentArgi}%
516     \csarg\xdef{dps@probLetter\the\count@}{\the\toks@}%
517     \csarg\xdef{dps@probLetterii\the\count@}{%
518       \ds@currentArgii}%
519   \fi

```

520 }

\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.

```
521 \def\writeComposingEnv{%
522   \newwrite \ds@composing@write
523   \immediate\openout \ds@composing@write \jobname_comp.def
524   \let\verbatim@out\ds@composing@write
525   \dps@IWW0{\string\begin{Composing}}%
526   \dps@IWW0{}%
527   \expandafter\write@ComposingEnv\expandafter{\puzzleParameters}%
528 }
529 \def\write@ComposingEnv#1{\write@@ComposingEnv#1\relax\relax}%
530 \newcommand{\ComposingEnvMsg}{\begin{quote}An outline of your
531   \texttt{Composing} environment is written to
532   \texttt{\jobname\_comp.def}, based on data in the
533   argument of your \texttt{\string\DeclarePuzzle} command. Copy and
534   paste the contents of this file into your puzzle document following
535   \texttt{\string\DeclarePuzzle} then fill in your questions and
536   answers. Good luck.\end{quote}}
537 \def\write@@ComposingEnv#1#2{\ifx#1\relax
538   \gdef\nextPuzzlePair{%
539     \dps@IWW0{\string\end{Composing}}%
540     \immediate\closeout \ds@composing@write}%
541   \begin{document}
542   \ComposingEnvMsg
543   \end{document}
544   \else\gdef\nextPuzzlePair{\write@@ComposingEnv{#1}{#2}}\fi
545   \nextPuzzlePair}%
546 \def\write@@@ComposingEnv#1#2{%
547   \gdef\ds@currentArgi{#1}\gdef\ds@currentArgii{#2}%
548   \ifx\ds@currentArgii\ds@myspace
549     \def\ds@probNumNext{\write@@ComposingEnv}%
550   \else
551     \ifx\ds@currentArgii\ds@punc
552       \def\ds@probNumNext{\write@@ComposingEnv}\else
553       \def\ds@probNumNext{%
554         \@ifundefined{dps@compQ#2}{% write to file
555           \expandafter\gdef\csname dps@compQ#2\endcsname{found}%
556           \dps@IWW0{\string\begin{cQ}{\noexpand#1}}%
557           \dps@IWW0{\string\end{cQ}}%
558           \dps@IWW0{\string\begin{cA}{\noexpand#1}}%}
```

```

559      \dps@IWV0{\string\end{cA}}%
560      \dps@IWV0{}%
561      }{}%
562      \write@@ComposingEnv
563      }%
564      \fi
565      \fi
566      \ds@probNumNext
567 }

```

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

```

568 \def\verbatimwrite{@bsphack
569   \let\do\@makeother\dospecials
570   \catcode`\^^M\active \catcode`\^I=12
571   \def\verbatim@processline{%
572     \immediate\write\verbatim@out
573     {\the\verbatim@line}}%
574   \verbatim@start}
575 \def\endverbatimwrite{@esphack}
576 \def\dps@IWV0{\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`.

```

577 \def\ds@populateList#1{%
578   \ds@listIn={}
579   \ds@nCnt=0%
580   @whilenum \ds@nCnt < #1\do {%
581     \advance\ds@nCnt@ne
582     \edef\ds@listInHold{\the\ds@listIn\noexpand\\{\the\ds@nCnt}}%
583     \ds@listIn = \expandafter{\ds@listInHold}%
584   }%
585 }

```

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

```

586 \def\ds@processi#1{\advance\ds@nCnt@ne
587   \ifnum\ds@nCnt=\ds@getRanNum\edef\ds@listOutHold{\the\ds@listOut}%
588   \global\ds@listOut=\expandafter{\ds@listOutHold\\{\#1}}%
589 \else
590   \edef\ds@listInHold{\the\ds@newListIn}%
591   \ds@tmpToks = \expandafter{\ds@listInHold\\{\#1}}%
592   \ds@newListIn = \expandafter{\the\ds@tmpToks}%
593 \fi
594 }

```

Used in `\displayRandomizedQuestions` to input the questions.

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

Used in `\displayRandomizedAnswers` to input the answers.

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

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

```
597 \def\ds@processL#1{\advance\count@0ne
598   \ifnum\count@0>\ds@aNumber\relax\else\input{\jobname_a#1.cut}\fi}
```

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

```
599 \def\ds@processR#1{\advance\count@0ne
600   \ifnum\count@0>\ds@aNumber\relax\input{\jobname_a#1.cut}\fi}
```

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

`andomizedAnswersLeftPanel` 601 `\def\displayRandomizedQuestions{\let\\=\ds@processii\the\ds@qlistOut`

`ndomizedAnswersRightPanel` 602 `\dps@AddToEmitAK{1}\dps@ckEmitAK}`

```
603 \def\displayRandomizedAnswers{\set@typeset@protect
```

```
604   \let\\=\ds@processiii\the\ds@alistOut
```

```
605   \dps@AddToEmitAK{2}\dps@ckEmitAK} % dps
```

```
606 \def\displayRandomizedAnswersLeftPanel{\set@typeset@protect
```

```
607   \let\\=\ds@processL\count@0z@
```

```
608   \divide\ds@aNumber\tw@ \xdef\lastOnLeft{\the\ds@aNumber}%
```

```
609   \the\ds@alistOut}%

```

```
610   \dps@AddToEmitAK{1}\dps@ckEmitAK}
```

```
611 \def\displayRandomizedAnswersRightPanel{\set@typeset@protect
```

```
612   \let\\=\ds@processR\count@0z@
```

```
613   \divide\ds@aNumber\tw@ \the\ds@alistOut}%

```

```
614   \dps@AddToEmitAK{1}\dps@ckEmitAK}
```

Develop a random permuted list for the questions.

```
615 \def\ds@randomizeQuestionList#1{%
616   \global\ds@listIn={} \global\ds@ newListIn={} \global\ds@listOut={}
617   \global\ds@tmpToks={} \global\ds@qlistOut={} \global\ds@alistOut={}
618   \ds@nMax=#1\relax\ds@populateList{\the\ds@nMax}%
619   \global\forquestionstrue
620   \ifwerandomize
621     \expandafter\ds@randomizeList
622   \else
623     \global\ds@qlistOut=\expandafter{\the\ds@listIn}
624   \fi
625 }
```

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

```
626 \def\ds@randomizeAnswerList#1{%
627   \global\ds@listIn={} \global\ds@ newListIn={} \global\ds@listOut={}
628   \global\ds@tmpToks={} \global\ds@qlistOut={} \global\ds@alistOut={}
629   \ds@nMax=#1\relax\ds@populateList{\the\ds@nMax}%
630   \global\forquestionsfalse
631   \ifwerandomize
632     \expandafter\ds@randomizeList
633   \else
```

```

634     \global\ds@alistOut=\expandafter{\the\ds@listIn}%
635   \fi
636 }

```

The loop that does all the work for randomizing.

```

637 \def\ds@randomizeList{\redefnrandomAsNeeded
638   \let\\=\ds@processi
639   \setrnum{\ds@getRanNum}{1}{\ds@nMax}%
640   \ds@nCnt\z@
641 \%typeout{LISTING: \the\ds@listIn}%
642   \the\ds@listIn
643   \ds@loopTest
644 }

```

The loop that does all the work for randomizing.

```

645 \def\ds@loopTest{\advance\ds@nMax\m@ne\relax
646   \ifnum\ds@nMax>\z@
647     \def\@next{%
648       \ds@listIn=\expandafter{\the\ds@newListIn}%
649       \ds@newListIn={}\ds@randomizeList}%
650   \else
651     \let\@next\relax
652     \ifforquestions
653       \global\ds@qlistOut=\expandafter{\the\ds@listOut}%
654 \%typeout{\ds@qlistOut = \the\ds@qlistOut}%
655   \else
656     \global\ds@alistOut=\expandafter{\the\ds@listOut}%
657 \%typeout{\ds@alistOut = \the\ds@alistOut}%
658   \fi
659   \fi
660   \@next
661 }

```

**\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.

```

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

```

## 7 Miscellaneous Settings

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

```

665 \newcommand{\threshold}[1]{\def\dsthreshold{#1}}
666 \threshold{3}
667 \newcommand{\penaltyPoints}[1]{\def\dspenaltyPoints{#1}}
668 \penaltyPoints{3}

```

```

669 \newcommand{\passing}[1]{\def\dpssassing{#1}}
670 \passing{4}
671 </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 it "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 initially 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:

- |                          |  |
|--------------------------|--|
| <code>wrtContent</code>  | <ol style="list-style-type: none"> <li>1. Compile the puzzle file with the option <code>wrtContent</code></li> <li>2. Compile the <code>icons.tex</code> file to create one or more icons files</li> </ol> |
| <code>!wrtContent</code> | <ol style="list-style-type: none"> <li>3. Compile the puzzle file with the option <code>!wrtContent</code></li> </ol>  |

#### 8.1.1 Macros for the main puzzle document

```

672 <*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 immediately after `\quesNumTxt`.

```

673 \newcommand{\quesNumTxt}[1]{\protect\textrm{bf}{Problem #1}}
674 \newcommand{\quesNumTxTPost}{\protect\nline}
675 \newenvironment{setContent}[1]{%
676   \immediate\write\@auxout{\string\csarg
677     \string\xdef{ltrpg#1}{\theenumi}}%
678   \ifwrtContent
679     \def\CommentCutFile{\jobname-sc(\theenumi).cut}%
680     \immediate\openout\CommentStream=\CommentCutFile
681     \begingroup
682     \set@display@protect
683     \let\verbatim@out\CommentStream
684     \dps@IWWO{\quesNumTxt{\theenumi}\quesNumTxTPost}%
685     \set@typeset@protect
686     \expandafter\verbatim@write
687   \else
688     \edef\x{\noexpand\pl@ceQues{\theenumi}}\x\expandafter
689     \comment
690   \fi
691 }{\ifwrtContent\expandafter
692   \endverbatim@write
693   \endgroup
694   \immediate\closeout\CommentStream
695 \else
696   \expandafter\endcomment
697 \fi
698 }

```

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

```
699 \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.

```

700 \def\pglstWarningMsg{\PackageWarningNoLine{dps}
701   {The file icons-pglst.sav not found.\MessageBreak
702   Icons may not appear. Build the\MessageBreak icons.tex file}}
703 \ifwerandomize\else\let\pglstWarningMsg\empty\fi
704 \def\dpsEmbedIcons{%
705 \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{\csO{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][]{%
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. `setContent` `\placeQuesIcon` defines a macro `\pl@ceQues`, which appears in the `setContent` environment above.

```

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{\csO{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.

```
727 \def\afterQhookA#1{%
728   if(!event.target.isBoxChecked(0))dpsHideQFields();\r
729   else\pdfSP if(PlayerSignIn())dpsShowQues("\ltrToNum{#1}");}
730 \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`.

```
731 \newcommand{\dpsEmbedSideShow}[3] [] {\begingroup
732 % \def\dps@NumSideShowPics{#2}%
733 \gdef\dpsNumSideShowPics{#2}%
734 \def\@Ext{#1}\ifx\@Ext\@empty\def\@Ext{.pdf}\else\def\@Ext{.#1}\fi
735 \@tempcnta\z@
736 \let\@embedList\@empty
737 \let\@DPSIndxList\@gobble
738 \whilenum \@tempcnta < \dpsNumSideShowPics \do{%
739   \ds@nCnt\@tempcnta \advance\ds@nCnt\@ne
740   \ifnum\ds@nCnt<10 \edef\x{0\the\ds@nCnt}\else
741     \edef\x{\the\ds@nCnt}\fi
742   \edef\z{\noexpand\g@addto@macro\noexpand\@DPSIndxList{,\\"x"}\z
743   \ifxetex\if@isPackaged
744     \PackageWarning{dps}
745     {There is no support for embedding packaged\MessageBreak
746      PDFs with xelatex. Ignoring the \string\isPackaged\MessageBreak
747      command}%
748   \fi\@isPackagedfalse
749 \fi\fi
750 \if@isPackaged
751   \ifpdf
752     \edef\y{\noexpand
753       \embedIcon[name=pic\x,%
754       hyopts={page=\x}{#3_package.pdf}}%
755   \else
756     \edef\y{\noexpand
757       \embedIcon[name=pic\x,placement=btnpic.\x,%
758       page=\x-1]{#3_package.pdf}}%
759   \fi
```

```

760     \else
761         \edef\y{\noexpand
762             \embedIcon[name=pic\x,placement=btncic.\x]{#3_\x@Ext}}%
763     \fi
764     \expandafter\g@addto@macro\expandafter\@embedList\expandafter{\y}%
765     \tempcnta\ds@nCnt
766 }% do
767 \toks@=\expandafter{\@embedList}\the\toks@
768 \endgroup
769 }

```

### 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.

```

770 \def\tileKVs#1{\def\tile@KVs{#1}}
771 \tileKVs{}
772 \newcommand\insertSideshow[4]{\begingroup
773   \offinterlineskip\tempcnta\z@%
774   \multido{\iR=1+1}{#1}{\hbox{%
775     \multido{\iC=1+1}{#2}{%
776       \global\advance\tempcnta\one
777       \ifnum\tempcnta<10\relax
778         \edef\x{0\the\tempcnta}\else
779         \edef\x{0\the\tempcnta}\fi
780       \edef\iconPresets{\noexpand\I{\noexpand\cs@f{pic\x}}}%}
781       \dps@otherIcon[\BC{}\FB{true}]\presets{\iconPresets}
782       \presets{\tile@KVs}{btncic.\x}{#3}{#4}}%
783     }% inner multido
784   }}% hbox, outer multido
785 \endgroup
786 }
787 </btadv>

```

#### 8.1.2 Macros for the icon document

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

```

788 <*icondoc>
789 \NeedsTeXFormat{LaTeX2e}[1997/12/01]

```

```

790 \ProvidesPackage{icon-doc}
791   [2020/04/21 v1.0 icon-doc:
792     Build Icon file and explode same (dps)]
793 \newif\ifdpsuseacrobat \dpsuseacrobatfalse

xelatex author      The icon-doc package has one option (and one convenience option). Both options
useacrobat          are targeted at users of xelatex. If the xelatex has Acrobat then use the useacrobat
!useacrobat         option; otherwise, 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.

794 \DeclareOption{useacrobat}{\dpsuseacrobattrue}
795 \DeclareOption{!useacrobat}{\dpsuseacbatfalse}
796 \DeclareOption{twice}{\dpscomptwicetrue}
797 \newif\ifdpscomptwice \dpscomptwicefalse
798 \ProcessOptions\relax
799 \RequirePackage{ifxetex}
800 \RequirePackage{shellesc}
801 \RequirePackage{web}
802 \RequirePackage{eforms}
803 \execJSON
804 \pagestyle{empty}
805 \parindent0pt \parskip0pt
806 \newwrite \wrtPkg
807 \newwrite\wrticonbody
808 \def\IWB#1{\immediate\write\wrticonbody{#1}}
809 \def\IWP#1{\immediate\write\wrtPkg{#1}}


Some standard code for writing verbatim to a file.

810 \def\verbatimwrite{@bsphack
811   \let\do\@makeother\dospecials
812   \catcode`\^M\active \catcode`\^I=12
813   \def\verbatim@processline{%
814     \immediate\write\verbatim@out
815     {\the\verbatim@line}}%
816   \verbatim@start}
817 \def\endverbatimwrite{@esphack}

icondoc  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
icons-template    a xelatex user without Acrobat. The environment writes the contents verbatim
                   to the file icons-template.tex. This file is later input by \MultiQuesFiles to
                   build individual icon files.

818 \newenvironment{icondoc}
819 {%
820   \immediate\openout \wrticonbody icons-template.tex
821   \let\verbatim@out\wrticonbody
822   \IWB{\string\RequirePackage{tmp}}%
823   \verbatimwrite
824 }{%
825   \endverbatimwrite

```

```
826     \immediate\closeout \wrticonbody
827 }
```

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

```
828 \ifxetex\ifdpsuseacrobat\else
829 \begin{icondoc}
830 \documentclass{article}
831 \usepackage{web}
832 \margins{3pt}{3pt}{3pt}{3pt}
833 \screensize{9}\baselineskip{2.25in}
834 \parindent0pt
835 \begin{document}
836 \small
837 \dpsInputContent    % required, defined in \@MultiQuesFiles
838 \end{document}
839 \end{icondoc}
840 \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. `\addToList` 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.

```
841 \let\pageList@\emptyset
842 \def\addToList#1{\edef\x{\{#1}\}}\expandafter
843   \g@addto@macro\expandafter
844   \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.

```
845 \def\@SnglQuesFile#1#2{%
846   \tempcnta#1\relax\advance\tempcnta\@ne
847   \edef\N{\the\tempcnta}%
848   \tempcnta\@ne\relax
849   \whilenum\tempcnta < \N \do{%
850     \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 `\addToList`; otherwise, we do register it with `\addToList`.

```
851   \InputIfFileExists{\the\tempcnta}.cut}
852   {\addToList{\the\tempcnta}\{\null\}\par
853 }
```

One icon per page.

```
854   \newpage
855   \advance\tempcnta\@ne
856 }%
857 }
```

\@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.

```

858 \def\@MultiQuesFiles#1#2{%
859   \tempcnta#1\relax\advance\tempcnta\@ne
860   \edef\N{\the\tempcnta}%
861   \tempcnta\@ne\relax
862   \whilenum\tempcnta < \N \do{%
863     \IfFileExists{#2-sc(\the\tempcnta).cut}{%
864       \addToPageList{\the\tempcnta}}{}}
865   \immediate\openout\wrtPkg tmp.sty
866   \IWP{\string\def\string\dpsInputContent{\string
867     \InputIfFileExists{#2-sc(\the\tempcnta).cut}{%
868       {}{\string\null}}}{}}
869   \immediate\closeout\wrtPkg

```

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.

```

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

```

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

```

875 \ShellEscape{del tmp.sty icons-template.*}%
876 }

```

\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.

```

877 \ifxetex
878   \ifdpsuseacrobat
879     \let\createRequiredIcons\@SnglQuesFile
880   \else
881     \let\createRequiredIcons\@MultiQuesFiles
882   \fi

```

```

883 \else
884   \let\createRequiredIcons\SnglQuesFile
885 \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*.

```

886 \begin{defineJS}[\def\defineJSjsR{{}^J}]{\execExplode}
887 /* Extract pages to folder */
888 // Regular expression used to acquire the base name of file
889 try {
890 for (var i = 0; i < this.numPages; i++)
891 aebTrustedFunctions(this,aebExtractPages,{
892 nStart: i,
893 cPath: "icons-" + (i+1) +".pdf"
894 });
895 } catch (e) { console.println("Aborted: " + e); }
896 \end{defineJS}

```

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

```

897 \def\wrtPageList{\newwrite\pagelist
898   \immediate\openout \pagelist icons-pglst.sav
899   \immediate\write\pagelist{\string\def\string\pagelist{\pageList}}
900   \immediate\closeout\pagelist
901 }

```

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

```

902 \ifxetex\ifdpsuseacrobat
903 \begin{execJS}{expl}
904 \execExplode
905 \end{execJS}
906 \fi\fi

```

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

```

907 \AtEndDocument{\wrtPageList}
908 \let\WriteBookmarks\relax
909 
```

*icons-pglst*

## 8.2 OCG methods

*aeb\_pro* required

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,

```
\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.

```
919 \long\def\placeQuesLayer#1{\ifundefined{textblock}
920   {\let\dps@mode\relax{\let\dps@mode\par}%
921    \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{%
  \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.

```
922 \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).

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

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

```
926 \newcommand{\quesNumTxt}[1]{\protect\textrbf{Problem #1}}
927 \newcommand{\quesNumTxTPost}{\protect\nothing}
```

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.

```
928 \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.

```
930 \def\CommentCutFile{\jobname-sc(\#1-\theenumi).cut}%
931 \immediate\openout\CommentStream=\CommentCutFile
932 \begingroup
933 \set@display@protect
934 \let\verbatim@out\CommentStream
935 \dps@IWW0{\quesNumTxt{\theenumi}\quesNumTxTPost}%
936 \set@typeset@protect
937 \verbatimwrite
938 }{%
939 \endverbatimwrite
```

```

940   \endgroup
941   \immediate\closeout\CommentStream
942   \edef\x{\noexpand\pl@ceQuesL@yer{\scArg-\theenumi}}\x
943 }

```

### 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}

944 \newcommand{\insertSideshow}[3][]{\begingroup
945   \def\@Ext{\#1}\ifx\@Ext\empty\else\def\@Ext{\#1}\fi
946   \def\nrows{\#2}\def\ncols{\#3}\insertSideshow@i
947 }
948 \newcommand\insertSideshow@i[2][]{\offinterlineskip
949   \tempcnta\z@%
950   \let\DPSSIndxList\gobble
951   \multido{\iR=1+1}{\nrows}{\hbox{%
952     \multido{\iC=1+1}{\ncols}{%
953       \global\advance\tempcnta\@ne
954       \ifnum\tempcnta<10\relax
955         \edef\x{0\the\tempcnta}\else\edef\x{\the\tempcnta}\fi
956         \edef\z{\noexpand\g@addto@macro\noexpand\DPSSIndxList{"\x"}\z
957         \x\Bld{pic\x}\includegraphics[\#1]{\#2_\x\@Ext}\eBld
958     }% inner multido
959   }}% hbox, outer multido

```

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

```

960   \immediate\write\auxout{\string\gdef\string
961     \DPSSIndxList{\DPSSIndxList}}%
962   \endgroup
963 }

964 </ocgadrv>

```

## 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.

```

965 {*english}
966 \def\chooseQ{"You must choose a question to answer before you answer!"}
967 \def\triedTooMuch{"You have tried this problem too many times,
968   I'm adding "
969   + \dpenaltyPoints
970   + " points, and resetting the penalty counter. Bad boy/girl!"}
971 \def\congratFinished{"Congratulations! You finished the puzzle"
972   +((nMissed==0) ? " without missing a single problem, amazing!"
973   : ", but you missed " + nMissed + " questions in the process!")
974 \def\regretPleased{(( nMissed > nPassing )
975   ? "I regret to report that you did not pass the test because
976       you missed too many questions."
977   : "I am pleased to report that you passed the test!"}
978 \def\reportPenaltyPoints{"The number of penalty points is "
979   + nPenaltyPoints +"."}
980 \def\finalPenaltyScore{"Final penalty score is "
981   + nTotalPenaltyPoints +"."}
982 \def\apenaltyScale{[-1,0], [0,4], [4, 10], [10,25], [25,5000]}
983 \def\apenaltyMsgs
984 {%
985   "Perfect!",
986   "Very nice performance!",
987   "This is not looking good. Perhaps a review is in order!",
988   "Are you trying? No one could do so badly, you only need a
989       seventh grade education!",
990   "You're hopeless!"
991 }
992 \dJSStr[noquotes]{\signInMsg}{%
993   You must enter your name in the field at the top of
994   the page to get credit for this assignment.}
995 {/english}
996 {*german}
997 \def\chooseQ{"Du musst erst eine Frage ausw\string\344hlen bevor
998 Du antwortest!"}
999 \def\triedTooMuch{"Du hast es leider zu oft versucht, ich
1000 z\string\344le " + \dpenaltyPoints
1001   + " Punkte dazu und setze den Z\string\344hler dann zur\string\374ck.
1002   Bitte streng Dich an!"}
1003 \def\congratFinished{"Herzlichen Gl\string\374ckwunsch! Du hast das
1004 Puzzle beendet" + ((nMissed==0) ?
1005   " ohne auch nur einen Fehler zu machen, wunderbar! " :
1006   ", aber leider " + nMissed + " Mal falsch geantwortet!"}
1007 \def\regretPleased{(( nMissed > nPassing )

```

```

1008 ? "Ich bedauere Dir mitteilen zu m\string\"374ssen, dass Du den Test
1009 leider nicht bestanden hast, weil Du zu viele Fragen falsch
1010 beantwortet hast."
1011 : "Ich bin \string\"344usserst erfreut Dir mitteilen
1012 zu d\string\"374rfen, dass Du den Test bestanden hast!"}
1013 \def\reportPenaltyPoints{"Die Anzahl der Strafpunkte ist "
1014 + nPenaltyPoints +"."}
1015 \def\finalPenaltyScore{"Die Gesamtanzahl der Strafpunkte ist damit "
1016 + nTotalPenaltyPoints + ". "}
1017 \def\aPenaltyScale{[-1,0], [0,4], [4, 10], [10,25], [25,5000]}
1018 \def\aPenaltyMsgs
1019 {%
1020 "Perfekt!",
1021 "Sehr gute Vorstellung!",
1022 "Es sieht nicht sonderlich gut aus. Vielleicht w\string\"344re
1023 eine Wiederholung gut!",
1024 "R\string\"344tst Du eigentlich nur? Niemand kann wirklich so
1025 schlecht sein. Das ist Stoff aus der siebten Klasse!",
1026 "Du bist ein hoffnungsloser Fall!"
1027 }
1028 \d1JSStr[noquotes]{\signInMsg}{%
1029 You must enter your name in the field at the top of
1030 the page to get credit for this assignment.}
1031 </german>
1032 <*package>

```

## 10 Form and document actions

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

```

1033 \newcommand{\printDPS}[3] [] {%
1034   \pushButton[\CA{Print}\A{\JS{%
1035     var pp = this.getPrintParams();\r
1036     pp.firstPage=1;\r
1037     pp.lastPage=1;\r
1038     pp.pageHandling = pp.constants.handling.shrink;\r
1039     var fv = pp.constants.flagValues;\r
1040     pp.flags |= (fv.suppressCenter | fv.suppressRotate);\r
1041     this.print(pp);}}#1}{printDPS}{#2}{#3}%
1042 }

```

\resetPDF[*<opts>*]{*<wd>*}{*<ht>*} Clears the puzzle board.

```

1043 \newcommand{\resetDPS}[3] [] {%
1044   \pushButton[\CA{Clear}\A{\JS{resetDPS();}}#1]{resetDPS}{#2}{#3}%
1045 }

```

\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.

```

1046 \def\dpsWCSWrnMsg{The file dps-wcs.def could not be found}
1047 \newcommand{\clearOnCloseOrSave}{\InputIfFileExists{dps-wcs.def}{}}
1048   {\PackageWarning{dps}{\dpsWCSWrnMsg}}}

1049 </package>
1050 <*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.

```

1051 \begin{willClose}
1052 resetDPS();
1053 \end{willClose}
1054 \begin{willSave}
1055 resetDPS();
1056 \end{willSave}
1057 </willCloseSave>

```

## 11 JavaScript Support

### 11.1 JavaScript common to all options

```

1058 <*package>
1059 \def\lengthOfMsg{2000} % in milliseconds

```

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

```

1060 \def\dpsResetHook#1{\def\dpsresethook{\#1}}
1061 \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.

```

1062 \def\dpsFinishedEvent#1{\def\dpsfinishedevent{\#1}}
1063 \dpsFinishedEvent{;}

```

The main JavaScript segment for DPS

```

1064 \begin{insDLJS}{match}{DPS: JavaScript support Das Puzzle Spiel}
1065 var playerSignedIn = false;
1066 var missesByQuestion = new Object();
1067 var nPassing = \dpspassing;
1068 var nMissed = 0;
1069 var nPenaltyPoints = 0;
1070 var activeQuestion = "";
1071 var f=this.getField("ckbxQ");
1072 var g=f.getArray();
1073 var QBC=g[0].strokeColor;
1074 var pic = new Object();
1075 var _dpsT0; // time out variable
1076 var bRandPicMaps=\bRandPicMaps;

```

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

```

1077 function PlayerSignIn()
1078 {
1079   if ( !playerSignedIn ) {
1080     var f = this.getField("dpsSignInName");
1081     if (f != null) {
1082       var nameField = f.value;
1083       if ( nameField.replace(/\s*/g,"") == "" ) {
1084         app.alert("\signInMsg");
1085         event.target.value="Off";
1086       } else
1087         playerSignedIn = true;
1088     } else playerSignedIn = true;
1089   }
1090   return playerSignedIn;
1091 }

processChoice(<name>)) This is the mouse up action of the answer check boxes.
1092 function processChoice(name)
1093 {
1094   // Get the question field that corresponds to this question,
1095   // see if checked.
1096   var f = this.getField("ckbxQ."+name);
1097   if ( (f != null) && (f.isBoxChecked(0)) ) { // right
1098     clearRedCrosses ();
1099     this.resetForm(["puzzle."+name]);
1100     event.target.textColor = ["RGB", 0, 0.6, 0];
1101     f.strokeColor = ["RGB", 0, 0.6, 0];
1102     f.readonly = true;
1103     event.target.readonly=true;
1104     try { afterCorrectChoiceHook() } catch(e) {};
1105     checkForFinished();
1106   } else { // wrong
1107     if ( activeQuestion != "" )
1108       var h = this.getField(activeQuestion);
1109     if ( (activeQuestion=="") || (h.readonly) ) {
1110       // active question already answered
1111       event.target.value = "Off";
1112       var g = this.getField("report");
1113       str = \chooseQ;
1114       g.value = str;
1115       var to = app.setTimeOut("clearMessages()", \lengthOfMsg);
1116     } else {
1117       event.target.style = style.cr;
1118       event.target.textColor = color.red;
1119       ++nMissed;
1120       if ( typeof missesByQuestion[activeQuestion] != "number" )
1121         missesByQuestion[activeQuestion] = 1;
1122       else
1123         missesByQuestion[activeQuestion] += 1;
1124       if (missesByQuestion[activeQuestion] > \dsthreshold) {

```

```

1125         var f = this.getField("report");
1126         str = \triedTooMuch;
1127         f.value = str;
1128         nPenaltyPoints += \dspenaltypoints;
1129         missesByQuestion[activeQuestion] = 0;
1130         clearRedCrosses ();
1131         var to = app.setTimeOut("clearMessages()", \lengthOfMsg);
1132     }
1133 }
1134 }
1135 }

clearMessages() Clears the text field named report, which is created by \placeMessageField.
\placeMessageField 1136 function clearMessages()
1137 {
1138     var f = this.getField("report");
1139     f.value = "";
1140 }

clearRedCrosses() Clears all answer check boxes what are marked with a cross, meaning wrong
choice. Part of the mouse up action question check boxes.
1141 function clearRedCrosses ()
1142 {
1143     var g = this.getField("ckbxA");
1144     h = g.getArray();
1145     for ( i=0; i < h.length; i++) {
1146         if ( h[i].style == style.cr) h[i].checkThisBox(0,false);
1147         h[i].style = style.ch;
1148     }
1149 }

checkForFinished() Determines if all questions have been answered. Called from processChoice().
1150 function checkForFinished()
1151 {
1152     var f = this.getField("puzzle");
1153     var g = f.getArray();
1154     var anyEmpty = false;
1155     for ( var i=0; i < g.length; i++) {
1156         if ( (g[i].name != "puzzle.space") && %
1157 (g[i].value.replace(/\s/g,"") == "") ) {
1158             anyEmpty = true;
1159             break;
1160         }
1161     }
1162     var nTotalPenaltyPoints=nMissed + nPenaltyPoints;
1163     if ( !anyEmpty ) {
1164         try { dpsFinishedHook() } catch (e) {};
1165         var f = this.getField("report");
1166         str = \congratFinished
1167         + "\n" + \regretPleased
1168         + "\n" + \reportPenaltyPoints
1169         + "\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){};

```

```

1210     playerSignedIn = false;
1211     missesByQuestion = new Object();
1212     nMissed = 0;
1213     nPenaltyPoints = 0;
1214     activeQuestion = "";
1215     aPicIndex=randomizePicList(aPicIndex,bRandPicMaps);
1216     if(bRandPicMaps)assignNamesToIndexes();
1217     this.delay=false;
1218 }

```

### 11.1.1 JavaScript to support a sideshow

```

1219 /*
1220     Create mapping from letters to pictures
1221 */
1222 var aPicIndex=[\DPSIndexList];
1223 var aNamesList=[\DPSNamesList];
1224 var diff;
1225 var PIlength=aPicIndex.length;
1226 var NLlength=aNamesList.length;
1227 if ( (diff=NLlength-PIlength) > 0 )
1228     for (var i=0; i<diff; i++)
1229         aPicIndex.push(null);
1230 // now randomize aPicIndex, if requested
1231 aPicIndex=randomizePicList(aPicIndex,bRandPicMaps);
1232 // now make assignments
1233 assignNamesToIndexes();
1234 function assignNamesToIndexes() {
1235     for(var i=0; i< NLlength; i++)
1236         pic[aNamesList[i]]=aPicIndex[i];
1237 }
1238 function randomizePicList(array, bRandomize) {
1239     if (bRandomize) shuffle(array);
1240     return array;
1241 }

```

The shuffle function was obtained from stackoverflow.<sup>1</sup>

```

1242 function shuffle(array) {
1243     var currentIndex = array.length, temporaryValue, randomIndex;
1244     // While there remain elements to shuffle...
1245     while (0 !== currentIndex) {
1246         // Pick a remaining element...
1247         randomIndex = Math.floor(Math.random() * currentIndex);
1248         currentIndex -= 1;
1249         // And swap it with the current element.
1250         temporaryValue = array[currentIndex];
1251         array[currentIndex] = array[randomIndex];
1252         array[randomIndex] = temporaryValue;
1253     }
1254     return array;

```

---

<sup>1</sup> [stack overflow .com/questions/2450954/how-to-randomize-shuffle-a-javascript-array](https://stackoverflow.com/questions/2450954/how-to-randomize-shuffle-a-javascript-array)

```
1255 }
1256 \end{insDLJS}
```

## 11.2 JavaScript for the usebtnappr option

```
1257 \ifusebtnappr
1258 \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.

```
1259 function dpsShowQues(n)
1260 {
1261   if (PlayerSignIn()) {
1262     dpsShowFld("btnQ."+n);
1263 %   var f=this.getField("btnQ."+n);
1264 %   if (f!=null) f.display=display.visible;
1265 }
1266 }
```

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

```
1267 function dpsHideQFields() {
1268   dpsHideFld("btnQ");
1269 %   var f=this.getField("btnQ");
1270 %   f.display=display.hidden;
1271 }
```

**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.

```
1272 function dpsHideFinalField(name){
1273   dpsHideFld(name);
1274 %   var f=this.getField(name);
1275 %   if (f!=null)f.display=display.hidden;
1276 }
```

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

```
1277 function dpsHidePreviousQues()
1278 {
1279   if ( activeQuestion != "" ) {
1280     dpsHideQFields();
1281   }
1282 }
1283 function dpsShowFld(name) {
1284   var f=this.getField(name);
1285   if (f!=null)f.display=display.visible;
```

```

1286 }
1287 function dpsHideFld(name) {
1288   var f=this.getField(name);
1289   if (f!=null)f.display=display.hidden;
1290 }

1291 function displayPic(name){
1292 %// var name = activeQuestion.replace(/ckbxQ\./,"");
1293 %// var name = displayPic.name;
1294 % console.println("name: "+name);
1295 % console.println("pic[name]: "+pic[name]);
1296 if (pic[name] != null )dpsShowFld("btnclick."+pic[name]);
1297 }

```

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

```

1298 function afterCorrectChoiceHook()
1299 {
1300   var name = activeQuestion.replace(/ckbxQ\./,"");
1301   dpsHideQFields();
1302   displayPic.name=name;
1303   _dpsTO=app.setTimeOut("displayPic('"+name+"')",25);
1304 }

```

`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.

```

1305 function dpsFinishedHook()
1306 {
1307   dpsHideQFields();
1308   try {
1309     \dpsfinishedevent
1310     if(typeof sortoutDPS == "function") {
1311       ok2Continue = true;
1312       sortoutDPS();
1313     }
1314   }catch(e){}
1315 }
1316 \end{insDLJS}

```

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

```

1317 </package>
1318 <*sortjs>
1319 \begin{insDLJS}{dpssort}{DPS: Bubble Sort}
1320 // Global Data:
1321 var hasBeenRandomized=false;
1322 var btnbase="btnclick."; // btnclick.01, btnclick.02, etc
1323 var iconbase="pic";    // pic01, pic02, etc.
1324 var nTotalTiles=\dpsNumSideShowPics;
1325 var randomDPS = new Array(nTotalTiles+1);
1326 var ldps = randomDPS.length;

```

```

1327 var timeout = 10;
1328 var shutdown;
1329 var debug = false; // memDebug;
1330 var ok2Continue = true;
1331 for (i=1; i<=nTotalTiles; i++) randomDPS[i]=i;
1332 // Mixup DPS:
1333 function mixupDPS()
1334 {
1335     var i, rand, temp;
1336     for (i=1; i<= nTotalTiles; i++)
1337     {
1338         var rand = Math.random();
1339         rand *= ldps*ldps;
1340         rand = Math.ceil(rand);
1341         rand = rand \% ldps;
1342         if (rand == 0 ) rand = 1;
1343         temp = randomDPS[i];
1344         randomDPS[i]=randomDPS[rand];
1345         randomDPS[rand]=temp;
1346     }
1347 }
1348
1349 // Show DPS:
1350 function showDPS()
1351 {
1352 %    \sortCustomStartJS
1353     var I,J;
1354     for ( var i=1; i<=nTotalTiles; i++ )
1355     {
1356         I=((i<10)?"0":"")+i;
1357         J=((randomDPS[i]<10)?"0":"")+randomDPS[i];
1358         var oIcon = this.getIcon(iconbase+J);
1359 %        var oIcon = oIconStreams[iconbase+J];
1360 %        var oIcon = this.getIcon("pic."+randomDPS[i]);
1361         var f = this.getField(btnbase+I);
1362         f.buttonSetIcon(oIcon);
1363     }
1364 }
1365 // Sortout DPS:
1366 function sortoutDPS()
1367 {
1368     outerLoop(randomDPS.length-1);
1369 }
1370 function outerLoop(i)
1371 {
1372     if ( ok2Continue && (i >= 0) ) %
1373     shutdown = app.setTimeOut("app.clearTimeOut(shutdown); %"
1374     innerLoop("+i+",1);", timeout);
1375     else {
1376 %        \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 
```

### 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.

```

1420 function dpsShowLayer(name)
1421 {
1422     toggleSetThisLayer(name, true);
1423 }

```

**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.

```

1424 function dpsHidePreviousLayer()
1425 {
1426     if ( activeQuestion != "" ) {
1427 %         var activename = activeQuestion.replace(/ckbxQ\./,"");
1428         var activename=activeQuestion.substring(6)
1429         dpsHideLayer(activename);
1430     }
1431 }

```

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

```

1432 function dpsHideLayer(name)
1433 {
1434     toggleSetThisLayer(name, false);
1435 }

```

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

```

1436 function afterCorrectChoiceHook()
1437 {
1438     var name = activeQuestion.replace(/ckbxQ\./,"");
1439     try {
        Used for special effects.
1440         if (pic[name] != null )
1441             dpsShowLayer("pic"+pic[name]);
1442     } catch(e) {};
1443     dpsHideLayer(name);
1444 }

```

**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.

```

1445 function dpsFinishedHook()
1446 {
1447     var name = activeQuestion.replace(/ckbxQ\./,"");
1448     dpsHideLayer(name);
1449     try {
1450         \dpsfinishedevent
1451     }catch(e){}
1452 }
1453 \end{insDLJS}
1454 \fi
1455 \dps@restoreCats

```

```
1456 \let\WriteBookmarks\relax  
1457 </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.

Symbols	
\%	337, 1341
\@Ext	734, 762, 945, 957
\@MultiQuesFiles	30, 837, 858, 881
\@SnglQuesFile	29, 845, 879, 884
\@auxout	676, 960
\@bsphack	568, 810
\@dpsPuzzleDone	260, 298
\@dpsPuzzleI	257, 259, 271, 285, 297
\@dpsPuzzleII	261, 264
\@embedList	736, 764, 767
\@isPackagedfalse	370, 748
\@isPackagedtrue	371
\@ltrFmtA	427, 468, 485
\@makeother	69, 569, 811
\@ncols	946, 952
\@next	647, 651, 660
\@nrows	946, 951
\@puzzNext	267, 270, 274, 276, 281, 284, 287, 290
\@rgi	233, 234
\@rgii	172–174
\@rowsep	250, 254
\@rowskip	250, 299
\@setSep@\dpsPuzzleII	267, 287, 292
\@takeaPeek	267, 274, 279, 281
\@unexpandable@protect	311
!nonrandomized (option)	2
!savedata (option)	4
!showanswerkey (option)	3
!showletters (option)	2
!useacrobat (option)	28
!viewmode (option)	2
!wrtContent (option)	23
\^	570, 812
\_	67, 69, 532
<b>A</b>	
\A	418, 444, 1034, 1044
\AA	420
\AAOnFocus	420
\active	570, 812
\addToPageList	842, 852, 864
afterCorrectChoiceHook (JS function)	43, 46
\aftergroup	374
\afterQhookA	16, 403, 404, 419, 727, 912
<b>B</b>	
\Aht	436, 445
\AnsAppearance	401
\AnswerKey	303
\aPenaltyMsgs	983, 1018, 1176
\aPenaltyScale	982, 1017, 1175
\argi	389, 394, 449, 453, 461
\argii	193, 201, 264, 265, 268, 279, 280, 283, 482
\AtEndDocument	349, 907
\AtEndOfPackage	62, 80, 83, 100
\Awidth	433, 445
<b>C</b>	
\CA	1034, 1044
cA (environment)	<u>387</u>
\cfooter	326
\checkbox	416, 442
\checkboxTmp	416, 424, 441, 447
checkForFinished (JS function)	39
\chooseQ	966, 997, 1113
clearMessages (JS function)	39
\clearOnCloseOrSave	<u>1046</u>
clearRedCrosses (JS function)	39
\comment	689
\CommentCutFile	679, 680, 930, 931
\CommentStream	680, 683, 694, 931, 934, 941
Composing (environment)	<u>372</u>
\ComposingEnvMsg	530, 542
\congratFinished	971, 1003, 1166
cQ (environment)	<u>375</u>
\createRequiredIcons	<u>877</u>
\csarg	161, 176, 318, 454, 456, 457, 514, 516, 517, 676
\csOf	719, 720, 780
<b>D</b>	
\day	108
\DeclareOption	794–796
\DeclareOptionX	3–9, 13, 16, 18, 19, 30, 41, 42
\DeclarePuzzle	<u>163</u> , 533, 535
\define@choicekey	52

\defineJSjsR ..... 886 \dpsLastSeed ..... 135, 341  
 \displayRandomizedAnswers ..... 601 \DPSNamesList ..... 165, 175, 357, 1223  
 \displayRandomizedAnswersLeftPanel ..... 601 \dpsNumSideShowPics ..... 733, 738, 1324  
 \displayRandomizedAnswersRightPanel ..... 601 \dpsOtherIcon ..... 25, 720, 781  
 \displayRandomizedQuestions ..... 601 \dpsQ ..... 380, 415  
 \divide ..... 115, 608, 613 \dpsQuesIcon ..... 25, 717  
 \dlJSStr ..... 992, 1028 \dpsQuesLayer ..... 32, 918  
 \documentclass ..... 830 \dpsResetHook ..... 37, 42, 1060, 1061  
 \dospecials ..... 569, 811 \dpsresethook ..... 40, 1060, 1204  
 \dps@Puzzle ..... 255, 256 \dpsShowLayer (JS function) ..... 45  
 \dps@AddToEmitAK ..... 226, 248, 602, 605, 610, 614 \dpsShowQues (JS function) ..... 42  
 \dps@ckEmitAK ..... 231, 248, 602, 605, 610, 614 \dpsuseacrobatfalse ..... 793, 795  
 \dps@ckForpglst ..... 89, 100 \dpsuseacrobattrue ..... 794  
 \dps@emitAK ..... 225, 227, 229, 231 \dpsWCSWrnMsg ..... 1046, 1048  
 \dps@emitEOP ..... 100, 101 \ds@getProbNumber ..... 502, 508  
 \dps@getNames ..... 166, 168 \ds@listOut ..... 155, 604, 609, 613, 617, 628, 634, 656, 657  
 \dps@getNames@i ..... 168, 169, 171 \ds@AnsAppearenceTok ..... 158, 402, 442  
 \dps@getOCGName ..... 916, 923 \ds@aNumber ..... 146, 373, 389, 390, 494, 598, 600, 608, 613  
 \dps@IWVO ..... 339, 341, 344, \ds@buildAnswerKey ..... 232, 306  
 380, 393, 525, 526, 539, 556–560, 576, 684, 935 \ds@composing@write ..... 522–524, 540  
 \dps@lang@type ..... 55–58, 61, 62 \ds@cr ..... 174, 268, 278, 283  
 \dps@LastSeed ..... 135, 139 \ds@currentArgi ..... 498, 515, 547  
 \dps@mode ..... 724, 725, 911, 920, 921 \ds@currentArgii ..... 498, 499, 510, 514, 518, 547, 548, 551  
 \dps@nextrandom ..... 103, 128 \ds@currFN ..... 463, 482  
 \dps@NumSideShowPics ..... 732 \ds@foundLetter ..... 463, 483, 486  
 \dps@OcgName ..... 916, 924 \ds@getNthOne ..... 509, 512  
 \dps@One ..... 162, 486 \ds@getProbNumber ..... 197, 205, 212, 218, 497  
 \dps@Puzzle ..... 246, 254 \ds@getRanNum ..... 149, 587, 639  
 \dps@restoreCats ..... 64, 1455 \ds@listIn ..... 150,  
 \dps@RP ..... 86, 87 578, 582, 583, 616, 623, 627, 634, 641, 642, 648  
 \dps@strut ..... 190, 198, 206, 213 \ds@listInHold ..... 582, 583, 590, 591  
 \dps@Zero ..... 162, 463, 483 \ds@listOut ..... 152, 587, 588, 616, 627, 653, 656  
 \dpsA ..... 393, 438 \ds@listOutHold ..... 587, 588  
 \dpsAitemOptArg ..... 11, 14, 15, 439 \ds@loopTest ..... 643, 645  
 \dpscomptwicefalse ..... 797 \ds@makeTextField ..... 9, 193, 201, 210, 216, 274  
 \dpscomptwicetrue ..... 796 \ds@myspace ..... 172, 194, 201, 496, 499, 548  
 \dpsEmbedIcons ..... 24, 704 \ds@nCnt ..... 143, 579–582, 586, 587, 640, 739–741, 765  
 \dpsEmbedSideShow ..... 26, 731 \ds@nCntCols ..... 148, 270, 284, 292, 293, 295, 298  
 \dpsFinishedEvent ..... 37, 1062, 1063 \ds@newListIn ..... 151, 590, 592, 616, 627, 648, 649  
 \dpsfinishedevent ..... 43, 46, 1062, 1309, 1450 \ds@nMax ..... 144, 618, 629, 639, 645, 646  
 \dpsFinishedHook (JS function) ..... 43, 46 \ds@populateList ..... 577, 618, 629  
 \dpsHideFinalField (JS function) ..... 42 \ds@probCnt ..... 147, 501, 502, 513  
 \dpsHideLayer (JS function) ..... 46 \ds@probNumNext ..... 499, 500, 506, 549, 552, 553, 566  
 \dpsHidePreviousLayer (JS function) ..... 46 \ds@processi ..... 586, 638  
 \dpsHidePreviousQues (JS function) ..... 42 \ds@processii ..... 595, 601  
 \dpsHideQFields (JS function) ..... 42 \ds@processiii ..... 596, 604  
 \DPSIndxList ..... 356, 737, 742, 950, 956, 961, 1222 \ds@processL ..... 597, 607  
 \dpsInputBtnAppr ..... 25, 27, 29, 80 \ds@processR ..... 599, 612  
 \dpsInputContent ..... 837, 866 \ds@publishRandomLists ..... 374, 492  
 \dpsInputOcgAppr ..... 36, 38, 40, 83 \ds@punc ..... 173, 253, 265, 280, 551

\ds@PuzzleAppearancetoks . . . . .	156, 183, 221
\ds@qlistOut . . . . .	154, 509, 601, 617, 623, 628, 653, 654
\ds@qNumber . . . . .	145, 323, 373, 377, 378, 493
\ds@QuesAppearancetoks . . . . .	157, 401, 416
\ds@question@write . . . . .	159, 378, 379, 385, 390, 391, 399
\ds@randomizeAnswerList . . . . .	494, 626
\ds@randomizeList . . . . .	621, 632, 637, 649
\ds@randomizeQuestionList . . . . .	493, 615
\ds@savedata . . . . .	331, 333, 336, 346
\ds@saveRandomSeed . . . . .	112, 131, 134, 140, 339
\ds@tmpToks . . . . .	153, 591, 592, 617, 628
\ds@typesetPuzzleLetter . . . . .	464, 473
\ds@ul . . . . .	194, 198, 202, 206
\ds@writePuzzleData . . . . .	330, 349
\dpassing . . . . .	669, 1067
\dspenaltypoints . . . . .	667, 969, 1000, 1128
\dsthreshold . . . . .	665, 1124
\DV . . . . .	221
<b>E</b>	
\eBld . . . . .	925, 957
\efKern . . . . .	424
\egroup . . . . .	185, 188, 229, 353, 410, 433, 459
\embedIcon . . . . .	709, 711, 713, 753, 757, 762
\endcomment . . . . .	696
\endverbatim@write . . . . .	384, 398, 575, 692, 817, 825, 939
\enspace . . . . .	424, 447, 468, 485
environments:	
cA . . . . .	387
Composing . . . . .	372
cQ . . . . .	375
icondoc . . . . .	818
setContent . . . . .	25, 33, 673
\eq@tabEnd . . . . .	254, 271, 285, 294
\eq@tabSep . . . . .	253, 296
\execExplode . . . . .	886, 904
\execJSON . . . . .	803
\ExecuteOptionsX . . . . .	17
<b>F</b>	
\F . . . . .	718, 721
\FB . . . . .	781
\Ff . . . . .	220, 663, 664, 718, 721
\FfMultiline . . . . .	664
\FfReadOnly . . . . .	220, 663, 718, 721
\FHidden . . . . .	718, 721
\finalPenaltyScore . . . . .	980, 1015, 1169
finalRating (JS function) . . . . .	40
\fmtOCGQues . . . . .	32, 917
\fmtOCGQues@i . . . . .	917, 925
\footnotesize . . . . .	196, 204, 212, 218, 327
\forquestionsfalse . . . . .	630
\forquestionstrue . . . . .	142, 619
<b>G</b>	
\g@addto@macro . . . . .	175, 742, 764, 843, 956
\getLetterNext . . . . .	462, 464, 465
\getNextN@me . . . . .	170, 171, 179
<b>H</b>	
\ht@fPF . . . . .	188–190, 222
\htOfA . . . . .	435, 437
\htOfQ . . . . .	412, 414
\htPuzzleFields . . . . .	9, 187
\Hy@pdfstringfalse . . . . .	199, 214, 245, 309, 452, 468, 485
\Hy@pdfstringtrue . . . . .	222, 245
\hypersetup . . . . .	78
<b>I</b>	
\I . . . . .	719, 720, 780
\iC . . . . .	775, 952
icondoc (environment) . . . . .	818
\iconPresets . . . . .	780, 781
\if@isPackaged . . . . .	370, 743, 750
\ifdpsscptwice . . . . .	797, 871
\ifdpuseacrobat . . . . .	793, 828, 878, 902
\ifeqforpaper . . . . .	11, 51, 191, 351, 415, 440, 662
\IfFileExists . . . . .	89, 863
\ifforquestions . . . . .	142, 652
\ifnextrandomredefd . . . . .	126, 127
\ifpdf . . . . .	710, 751
\ifsavepuzzledata . . . . .	47, 330
\ifshowletters . . . . .	
. . . . .	45, 195, 203, 210, 216, 232, 299, 342, 393, 460
\ifshowsolution . . . . .	46, 325
\ifusebttnappr . . . . .	31, 48, 79, 85, 91, 99, 360, 1257
\ifuseocgappr . . . . .	20, 49, 82, 1418
\ifviewMode . . . . .	44, 192, 209, 245
\ifwerandomize . . . . .	43, 132, 136, 338, 620, 631, 703
\ifwrtContent . . . . .	50, 92, 678, 691
\ifxetex . . . . .	72, 408, 412, 430, 435, 708, 743, 828, 877, 902
\includegraphics . . . . .	957
\InitLayout . . . . .	353
\input . . . . .	55–58, 61, 102, 595, 596, 598, 600, 918
\InputIfExists . . . . .	
. . . . .	27, 38, 133, 137, 362, 705, 851, 867, 1047
\inputRandomSeed . . . . .	6, 132
\insertPuzzle . . . . .	233
\insertQuesLayer . . . . .	33, 923
\insertSideshow . . . . .	27, 34, 772, 944

\insertSideshow@i . . . . .	946, 948	N				
\iR . . . . .	774, 951	\N . . . . .	847, 849, 860, 862			
\isPackaged . . . . .	746	\n . . . . .	707, 709, 711, 713, 714, 1167–1169			
\item . . . . .	415, 439	\nCols . . . . .	181, 182, 235, 240, 243, 257, 293			
\IWB . . . . .	808, 822	\NeedsTeXFormat . . . . .	789			
\IWP . . . . .	809, 866	\newline . . . . .	674, 927			
<b>J</b>						
\JS . . . . .	363, 418, 420, 444, 1034, 1044	\newwrite . . . . .	159, 331, 522, 806, 807, 897			
JS functions:		\nextPuzzleChar . . . . .	260–262			
afterCorrectChoiceHook . . . . .	43, 46	\nextPuzzleLetter . . . . .	478–480			
checkForFinished . . . . .	39	\nextPuzzlePair . . . . .	538, 544, 545			
clearMessages . . . . .	39	\extrandom . . . . .	104, 111, 128			
clearRedCrosses . . . . .	39	\extrandomredefdfalse . . . . .	126			
dpsFinishedHook . . . . .	43, 46	\extrandomredefdtrue . . . . .	129			
dpsHideFinalField . . . . .	42	nonrandomized (option) . . . . .	2			
dpsHideLayer . . . . .	46	\nPuzzleCols . . . . .	8, 181, 239			
dpsHidePreviousLayer . . . . .	46	\null . . . . .	852, 868, 874			
dpsHidePreviousQues . . . . .	42					
dpsHideQFields . . . . .	42	<b>O</b>				
dpsShowLayer . . . . .	45	\offinterlineskip . . . . .	773, 948			
dpsShowQues . . . . .	42	\OnFocusQhookAA . . . . .	16, 405, 422, 730, 915			
finalRating . . . . .	40	\OpenAction . . . . .	363			
PlayerSignIn . . . . .	37	\openout . . . . .	333, 378, 390, 523, 680, 820, 865, 898, 931			
processChoice . . . . .	38	options:				
resetDPS . . . . .	40	!nonrandomized . . . . .	2			
toggleSetThisLayer . . . . .	45, 46	!savedata . . . . .	4			
<b>K</b>						
\kern . . . . .	302, 468, 485	!showanswerkey . . . . .	3			
<b>L</b>						
lang (option) . . . . .	4	!showletters . . . . .	2			
\lastOnLeft . . . . .	608	!useacrobat . . . . .	28			
\lengthOfMsg . . . . .	1059, 1115, 1131	!viewmode . . . . .	2			
\loop . . . . .	313	!wrtContent . . . . .	23			
\ltrFmtA . . . . .	16, 427, 428	lang . . . . .	4			
\ltrToNum . . . . .	24, 699, 729, 730	nonrandomized . . . . .	2			
<b>M</b>						
\m@ne . . . . .	645	savedata . . . . .	4			
\makeXasPDOff . . . . .	72	showanswerkey . . . . .	3			
\margins . . . . .	832	showanswerlabels . . . . .	3			
\month . . . . .	109	showletters . . . . .	2			
\msgi . . . . .	334, 340	useacrobat . . . . .	28			
\msgii . . . . .	335, 344	usebttnappr . . . . .	3, 23			
\multicolsep . . . . .	160	uselayers . . . . .	3			
\multido . . . . .	774, 775, 951, 952	viewmode . . . . .	2			
\multiply . . . . .	107–109, 117, 119, 120	wrtContent . . . . .	23			
<b>P</b>						
\PA . . . . .	719, 722					
\PackageWarning . . . . .	59, 236, 303, 744, 1048					
\PackageWarningNoLine . . . . .	21, 32, 700					
\pageList . . . . .	841, 844, 899					
\pagelist . . . . .	706, 707, 897–900					
\pagestyle . . . . .	804					
\parindent . . . . .	805, 834					

\parskip .....	805	\rowsep@default .....	251, 252
\passing .....	665	\rule .....	190
\pdfSP .....	729, 914		
\penaltytypoints .....	665		
\pglstWarningMsg .....	700, 703, 706		
\pl@ceQues .....	688, 725		
\pl@ceQuesL@yer .....	921, 942		
\placeMessageField .....	39, 662		
\placeOtherIcon .....	25, 726		
\placeOtherLayer .....	33, 922		
\placeQuesIcon .....	25, 723		
\placeQuesLayer .....	33, 919		
PlayerSignIn (JS function) .....	37		
\presets .....	781, 782		
\previewfalse .....	6		
\previewtrue .....	5		
\printDPS .....	1033		
\printPDF .....	1033		
processChoice (JS function) .....	38		
\ProcessOptions .....	798		
\ProcessOptionsX .....	63		
\ProvidesPackage .....	790		
\pushButton .....	718, 721, 1034, 1044		
\PuzzleAppearance .....	8, 183		
\puzzleParameters .....	164, 247, 474, 527		
<b>Q</b>			
\Q .....	219		
\Qht .....	413, 422		
\QuesAppearance .....	401		
\quesNumTxt .....	23, 33, 673, 684, 926, 935		
\quesNumTxTPost .....	23, 33, 674, 684, 927, 935		
\Qwidth .....	410, 422		
<b>R</b>			
\r .....	419, 421, 728, 913, 1035–1040		
\raisebox .....	195, 203, 210, 217		
\randomi .....	105–110, 112, 114, 118, 119, 121, 122, 130, 134, 139, 140, 339, 341		
\randomizePicMappings .....	13, 358		
\redefnextrandomAsNeeded .....	127, 637		
\regretPleased .....	974, 1007, 1167		
\repeat .....	323		
\reportPenaltyPoints .....	978, 1013, 1168		
\RequirePackage .....	2, 70, 71, 73–77, 86, 799–802, 822		
\resetDPS .....	1043		
resetDPS (JS function) .....	40		
\resetPDF .....	1043		
\rotatebox .....	328		
\rowsep .....	250		
<b>S</b>			
\S .....	220, 718, 721		
\s .....	1083, 1157		
savedata (option) .....	4		
\savepuzzledatafalse .....	42, 47, 95		
\savepuzzledatatrue .....	41, 93		
\scArg .....	929, 942		
\screensize .....	833		
\set@display@protect .....	343, 379, 392, 682, 933		
\set@typeset@protect .....	381, 395, 603, 606, 611, 685, 936		
setContent (environment) .....	25, 33, 673		
\setdpsfootskip .....	13, 350, 355		
\setrnum .....	639		
\ShellEscape .....	870–872, 875		
showanswerkey (option) .....	3		
showanswerlabels (option) .....	3		
showletters (option) .....	2		
\showlettersfalse .....	8, 45		
\showletterstrue .....	7, 51		
\showsolutionfalse .....	18, 46		
\showsolutiontrue .....	16		
\sideshowPackaged .....	14, 26, 371		
\signInMsg .....	992, 1028, 1084		
\small .....	836		
\smash .....	424, 447		
\sortCustomFinishJS .....	1376, 1405		
\sortCustomStartJS .....	1352		
\sortPicMappings .....	14, 361, 368		
\strut .....	423, 446		
<b>T</b>			
\texorpdfstring .....	246, 310, 326, 342, 451, 467, 484		
\textbf .....	673, 926		
\textField .....	219, 663		
\textfieldTmp .....	219, 222		
\textSize .....	417, 443		
\texttt .....	531–533, 535		
\TFOR .....	707, 709, 711, 713		
\theenumi .....	677, 679, 684, 688, 930, 935, 942		
\thr@O .....	231		
\threshold .....	665		
\tile@KVs .....	770, 782		
\tileKVs .....	27, 770, 771		
\time .....	106		
toggleSetThisLayer (JS function) .....	45, 46		
\tops .....	246, 310, 326, 342, 451, 467, 484		
\topsep .....	160		

\TP	718, 721
\triedTooMuch	967, 999, 1126
\typeset@PuzzleLetter	475, 476, 487, 489
\typeset@PuzzleLetter	474, 475
\typeset@PuzzleLetteri	479, 482
 U	
\uccode	337
\underbar	194, 202, 213
\uppercase	337
useacrobat (option)	28
usebttnappr (option)	3, 23
\usebttnapprfalse	24, 48
\usebttnapprtrue	19
\useLastSeed	6, 136
uselayers (option)	3
\useocgapprfalse	35, 49
\useocgapprtrue	30
\usepackage	831
\useRandomSeed	6, 130
 V	
\verbatim@line	573, 815
\verbatim@out	336, 379, 391, 524, 572, 576, 683, 814, 821, 934
\verbatim@processline	571, 813
\verbatim@start	574, 816
\verbatimwrite	382, 396, 568, 686, 810, 823, 937
viewmode (option)	2
\viewModefalse	6, 44
\viewModetrue	5
 W	
\wd@fPF	185, 186, 196, 198, 204, 206, 211, 213, 217, 222
\wdPuzzleFields	9, 184
\web@footskip	352
\werandomizefalse	3
\werandomizetrue	4, 43
\widestFmtALtr	16, 429, 434
\widestFmtDQNum	16, 406, 411
\write	572, 576, 676, 808, 809, 814, 899, 960
\write@@ComposingEnv	544, 546
\write@@ComposingEnv	529, 537, 549, 552, 562
\write@ComposingEnv	527, 529
\WriteBookmarks	908, 1456
\writeComposingEnv	521
\wrtContent (option)	23
\wrtContentfalse	90, 705
\wrtContenttrue	50
\wrtIconbody	807, 808, 820, 821, 826
\wrtPageList	897, 907
\wrtPkg	806, 809, 865, 869
 X	
\x	80, 81, 83, 84, 316, 317, 688, 740–742, 753, 754, 757, 758, 762, 778–780, 782, 842, 844, 924, 942, 955–957
\xBld	924, 957
 Y	
\y	315, 316, 319, 752, 756, 761, 764
\year	107
 Z	
\z	742, 956

## 13 Change History

v1.0 (2006/10/31)	
General: Added the <code>showanswerlabels</code> option	3
\DeclarePuzzle: Added a token list to hold any user changes in the appearance of the puzzle, as as suggested by Robert Marik.	9
v1.1 (2020/04/21)	
General: Added <code>\setdpsfootskip</code>	13
Added <code>\widestFmtDNum</code> to set width of the checkbox for questions	16
Added <code>resetDPS()</code> JavaScript function.	40
Replace <code>tabular*</code> with <code>tabular</code> in <code>\insertPuzzle</code>	11
cA: Modified cA, using <code>\set@display@protect</code>	
and <code>\set@typeset@protect</code>	15
\DeclarePuzzle: Added <code>\wdPuzzleFields</code> and <code>\htPuzzleFields</code>	9
\dpsQ: Added <code>\ltrFmtA</code> to format the answer letter	16
cQ: Modified cQ, using <code>\set@display@protect</code> and <code>\set@typeset@protect</code>	14
\writeComposingEnv: <code>\writeComposingEnv</code> no longer needs to be followed by begin and end document	19
v1.5 (2020/08/06)	
General: Support for packaged sideshow graphics	26

v1.6 (2020/05/31)

General: Improved parsing for punc and cr . . . . . 11

v1.7 (20/06/03)

General: Changing from \wrtContentfalse as

the default to \wrtContenttrue as the default.	4
Detect if icons-pglst.sav, if yes set \wrtContentfalse	5