AcroTeX.Net

dps Package

Das Puzzle Spiel

D. P. Story

Abstract: Das Puzzle Spiel is a LaTeX package for creating a puzzle, a message actually, and a series of questions and answers. The document consumer matches the questions with the answers. With each match, another letter appears in the puzzle. Upon completion of all questions, the message hidden in the puzzle is revealed.

Distribution: Click on the link below, and save `dps.txt` to your hard drive.[a]

`dps.txt`

Rename `dps.txt` to `dps.zip`. Unzip `dps.zip` somewhere in your LaTeX search path. Unzipping will create a folder named `dps`. If using a TeX system requires it, refresh your file name database.

Brief descriptions of these files are given on page 3.

[a] If the link does not work for you, save `dps.txt` using the Adobe Reader user interface.

Last Revision Date: October 17, 2006

Table of Contents

1. Introduction

The work on this package was inspired by one of my son's worksheets in pre-algebra. The worksheet consisted of a series of simple algebraic expressions the student was to simplify. The simplified form was listed somewhere in the answers column. The answer had a letter associated with it which the student then placed in a puzzle. Upon completion of the worksheet, the puzzle (message) is completely filled in; if the message makes sense, the student can determine that he/she did the worksheet correctly.

I set out to duplicate this worksheet for electronic media, but also to have an option for paper as well.

2. Requirements

The following packages are required for dps beyond that of the standard LaTeX distribution:

- AeB: The Web and eForms components of the AcroTeX eDucation Bundle are used.
- `random.tex`: A TeX/LaTeX macro file to generate random numbers, the package was written by Donald Arseneau.
- xkeyval: For superior management of package options.

The package should work for users of distiller (dvips, dvipsone), pdftex and dvipdfm.

3. Acquiring dps.dtx and its demo files

The distribution is attached to this documentation.

- `dps_demo.tex`: A small demo file, that you can use to try the package with different options. Can build this for screen or for paper (`forpaper/forcolorpaper`). Two column format for questions, and half the answers on the left, and half on the right.
- `pmg_d1.tex`: The original file constructing during the development of this package. This puzzle has the famous u-umlaut. Can be compiled with the `forpaper/forcolorpaper` option of web. The layout is designated as "design 1" (d1): questions in center in two column format and answer on the left and right.
- `pmg_d1_p.tex`: Same as `pmg_d1.tex`, but set up `forpaper/forcolorpaper` in landscape mode.
- `pmg_d2_p.tex`: Same puzzle/questions as `pmg_d1.tex`, but uses the layout designated as "design 2" (d2): Questions and answer in column format on left and right, puzzle in the center. This leave a lot of white space in the middle, perhaps for a graphic.
- `pmg_d3_p.tex`: Same puzzle/questions as `pmg_d1.tex`, but uses the layout designated as "design 3" (d3): Puzzle and questions vertically aligned (questions in two column format), answer in single column.

4. Designing your Puzzle

There are several test files that you can use as basis of constructing your own puzzle.[1] The files themselves illustrate adequately the structure of the puzzle, but a few remarks are in order.

4.1. \DeclarePuzzle

The first step is to have a message, either funny, serious, or whatever. For the purpose of illustration, suppose your message is "Hello, Jürgen!" This message has all the elements that need to be discussed, letters, punctuation, spaces, capitalized letters, and accented letters.

`\DeclarePuzzle`: Use the argument for this command to declare the letters in the puzzle. The argument consists of a series of *paired parameters*, the first of each pair is a letter in the puzzle, a punctuation or a space; and the second in the pair is a name, which is used as a field name for the underlying form fields.

In the design of the puzzle there are three sets of form fields to manage: checkboxes for the questions, checkboxes for the answers, and text fields for the puzzle. The checkboxes and puzzle letter are all tied together by a common base field name, which is the second parameter in the parameter pairs.

In our puzzle, "Hello, Jürgen!", we place the following in preamble:

```
\DeclarePuzzle
{
   {H}{H}
   {e}{e}
   {l}{l}
   {l}{l}
   {o}{o}
   {,}{punc}
   {}{space}
   {J}{J}
   {\texorpdfstring{\"{u}}{\string\374}}{uml}
   {r}{r}
   {g}{g}
   {e}{e}
   {n}{n}
   {!}{punc}
}
```

For letters, the second parameter in the pair can be the same, as in `{H}{H}` and `{e}{e}`.

For punctuation, the first argument is the punctuation mark, the second is a special name `punc`; consequently, you see in the above `{,}{punc}` and `{,}{punc}`. Punctuation is not part of the puzzle (though it could be), the macro that ultimately processes this list of paired parameters looks for the name `punc`, and handles it appropriately.

[1] My apologies, this game is more properly described as a matching game with message, but my friend insisted that I call it a puzzle so that the package could be named Das Puzzle Spiel, or dps, which are my initials. dps

The space character also has a special second parameter `space`. The space will have a place in the puzzle, but has no question/answer corresponding to it.

Finally, there are the special latin-1 letters. In the puzzle above, I have an u-umlaut. The problem is that this umlaut appears in two contexts, a TEX and PDF context; the u- umlaut will appear as typeset content by TEX and will appear as a value of an Acrobat form field. The representation is different in each context. In this case, use the hyperref command `\texorpdfstring`. In above, the first argument of the u-umlaut is `\texorpdfstring{\"{u}}{\string\374}`. When typeset, the first argument of `\texorpdfstring` is used and in the PDF context, the second argument of `\texorpdfstring` is used.

Rule: As a general rule, and this rule will be repeated later, there should be one question for each distinct second argument, that is, for each distinct base name.

In the above example, letters like `e` and `l` appear more than once in the puzzle. Note that the second argument of each of these two e's is the same. There should be only one for this `e`, and when the question associated with `e` is correctly answered, both e's in the puzzle will appear. (If you want a question for each `e`, then you need to name the fields differently, `{e}{e1}` and `{e}{e2}`, for example.)

4.2. Begin composing the questions and answers

`Composing`: You compose the questions and answers within the `Composing` environment. At the top of the environment, certain counters are initialized. All the work is done at the end of the environment: The number of questions and answers are known, at which point, the order of the questions and answers are randomized.

`cQ` and `cA`: Within the `Composing` environment, you compose your questions and answers within the `cQ` and `cA` environments, respectively. Each question must be followed by its answer. You can have more answers than questions, but these answers *must* be listed last. (The answers will be randomized anyway.)

Each of these environments has one required argument, and `cA` has one optional argument. The required argument is the field name to which this answer corresponds. (The second argument of the paired arguments of `\DeclarePuzzle`.)

This optional argument of the `cA` environment only relevant when the document is compiled with the `showletters` option. The value of the argument is a letter to appear in the answers column. Normally, one of the first entries in the `\DeclarePuzzle` command 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 `Composing` environment, and the two environments `cQ` and `cA` that are contained within the `Composing` environment (can) go in the preamble just after the `\DeclarePuzzle` command.

When developing questions and answers, you must keep in mind the following rule:

One problem I discovered in the process of developing the demo files is setting up these environments so there are no errors.

Rule: As a general rule, and this rule will be repeated later, there should be one question for each distinct second argument, that is, for each distinct base name.

Because of human errors, sometimes we have questions/answer pairs that correspond to a duplicate field name—perhaps this letter occurs multiple times. No, no, that violates the red rule above. After having made this same error several times, I decided to let TEX do the work for me.

`\writeComposingEnv`: Just after the end of the argument of `\DeclarePuzzle`, place this command, like so:

```
\writeComposingEnv
\begin{document}
\end{document}
```

Remember, `\DeclarePuzzle` is in the preamble and the beginning of document has not been encountered, so lets just put the begin and end document just for the purpose of executing `\writeComposingEnv`. This command writes the file `\jobname_comp.def`. Once this file is created, copy and paste its contents into your source file. It should look like this:

```
\begin{Composing}

\begin{cQ}{M}
\end{cQ}
\begin{cA}{M}
\end{cA}

\begin{cQ}{a}
\end{cQ}
\begin{cA}{a}
\end{cA}
...
\end{Composing}
```

The correct number of environments should be there, with the correct argument inserted for each environment. Cool! Now, just compose your questions and answers.

After the file `\jobname_comp.def` has been created, *delete the* `\writeComposingEnv` *command* (and the begin/end document if you included it). This command is not part of the puzzle, but a helper component.

Another rule.

Rule: You must have one answer per question. Each answer is unique in the list of all answers (no two answers can be the same). You can have more answers then questions. You can have more answers than questions (distractions—answers "close" in appearance to the correct answer). Each of the fake questions must have a unique name.

5. Placing the Content

The content consists of five parts, plus whatever you wish to include in the document:

1. The title and instructions.
2. The puzzle
3. The questions
4. The answers
5. The message field

Each of these is discussed the the sections that follow.

5.1. The title and instructions

The title and instructions are your bailiwick, see package demo files for suggestions.

5.2. The puzzle

`\insertPuzzle{<nCols>}`: The puzzle, which consists of the first of the paired arguments declared in the `\DeclarePuzzle` command, is laid out in a tabular format. The one required argument of `\insertPuzzle` is `nCols`, the number of columns per row to be used.

If you look at the demo files, you'll see that `\insertPuzzle` is enclosed in a `minipage` environment (and in a `\fbox` as well). By placing `\insertPuzzle` you can control the width of the table, and it may help fit it with the other components of the game.

`\PuzzleAppearance`: Use this command to change the appearance of the Acrobat text field that comprise the interactive puzzle. The command takes one argument, this argument consists of one or more commands from the eForms package that change the appearance of a field. For example,

```
\PuzzleAppearance{\BC{1 0 0}}
```

changes the boundary color to red. See the eForms documentation.

`\rowsep`: By setting `\rowsep`, you can adjust the vertical space between tabular rows. The commands takes one argument, the amount of vertical skip, for example,

```
\rowsep{2ex}
```

5.3. The questions

`\displayRandomizedQuestions`: The questions are inserted by this command. This command must be placed in an `enumerate` environment. This will number the questions, so when, for example, the `showletters`, discussed later, is taken, there is a visual mapping between the questions, the answers and the puzzle.

If the number of questions is not great, you can list the questions in a single column; however, in the examples that I have done, I've determined that a two column format (using the `multicol` package) seems to me to be the best layout for the questions.

`\QuesAppearance`: Use this command to change the appearance of the Acrobat checkboxes for the questions that appear in the label margin. The command takes one argument, this argument consists of one or more commands from the eForms package that change the appearance of a field. For example,

```
\QuesAppearance{\BC{.5 .5 .5}}
```

changes the boundary color to a gray color. See the eForms documentation.

5.4. The answers

`\displayRandomizedAnswers`: As with the questions, the answer are displayed in by a similar command. Again, this command should be in a list environment, preferably the `itemize`. The list label is suppressed by placing `\item[]` before each answer. When the `showletters` option is taken, the letter this answer corresponds to will be the label.

One of the design layouts in the demo files has the questions in a two column format in the center with two columns of answers, half the answers to the left of the questions, and the other half to the right. The two commands

```
\displayRandomizedAnswersLeftPanel
\displayRandomizedAnswersRightPanel
```

are used for that purpose.[2] As with `\displayRandomizedAnswers`, each of these commands should be in an `itemize` environment.

`\AnsAppearance`: Use this command to change the appearance of the Acrobat checkboxes for the answers that appear in the label margin. The command takes one argument, this argument consists of one or more commands from the eForms package that change the appearance of a field. For example,

```
\AnsAppearance{\BC{.5 .5 .5}}
```

changes the boundary color to a gray color. See the eForms documentation.

[2] TEX doesn't know his left from his right, so you can actually place the left panel listing on the right. TEX will not object!

5.5. The message field

`\placeMessageField[<opts>]{<width>}{<height>}`: This Acrobat text field is used to write messages to the user. If the user tries to choose an answer before selecting an answer, s/he gets the message

```
"You must choose a question to answer before you answer!"
```

For the interactive version of this game[3], there is a language option for dps to change the messages from English, the default, to German, for example.

The parameters are `width` (the width of the text field), `height` (the height of the text field and `opt` (the optional argument for changing the appearance of the field, as described in the documentation of the eForms package.

The message field is automatically removed when the document is compiled in the `forpaper` option of the web package.

6. Let's have some Fun

In order to make answering the questions "fun", and in addition to the puzzle (or message), I implemented a point system. Each time the user checks an incorrect answer, that is recorded as a miss. After completion of the game, the JavaScript determines if the user has passed the test. To make it more interesting, a penalty point system is also used: If the user incorrectly answer the same questions multiple times (guessing!), penalty points are given.

The document author can set the various parameters of this aspect of the game.

`\threshold{<n>}`: The number, n, of times a person is allowed to miss the same question before being "awarded" penalty points. The command `\threshold` with its one argument defines the command `\dsthreshold` which expands to the argument, n, of `\threshold`. Set `\threshold` in the preamble, and use `\dsthreshold` as part of the instructions or description of the game. The default is `\threshold{3}`.

`penaltypoints{<n>}`: The number, n, of penalty points to be added into the final score. Penalty points are "awarded" for missing the same question more than the number specified by the argument of `\threshold`. The command `\penaltypoints` with its one argument defines the command `\dspenaltypoints` which expands to the argument, n, of `\penaltypoints`. Set `\penaltypoints` in the preamble, and use `\dspenaltypoints` as part of the instructions or description of the game. The default is `\penaltypoints{3}`.

`passing{<n>}`: The maximum number, n, of questions the user needs to miss and still pass the test. Passing or not does not depend on the number of penalty points. The command `\passing` with its one argument defines the command `\dspassing` which expands to the argument, n, of `\passing`. Set `\passing` in the preamble, and use `\dspassing` as part of the instructions or description of the game. The default is `\passing{4}`.

The number of incorrect answers and the total penalty points are combined. Based on the combined score, a final evaluation of the user's knowledge on the subject is displayed.

[3] Does that mean there is also a non-interactive version of this game, sir? Yes, yes there is. dps

7. Package Options

Here we list the options of package dps.

`nonrandomized`: The default behavior is to randomize the questions and to randomize the answers. With this option, the questions and answer are listed in the order that they appear in the `Composing` environment. This makes it easy for the document author to quickly solve the puzzle and to see if the check marks and letters appear as they should.

`viewmode`: Useful when using a dvi previewer. Here you can see the placement of the puzzle (with the letters to the puzzle filled in) and the boxes were the checkboxes go. Useful in the designing the layout of your game phase.

`showletters`: Sets up a visual relation between the questions, the answers and the puzzle elements. Useful in design phase, can be used with `viewmode`.

`showanswerkey`: Shows the answer key in the footer of the document. If the graphicx package is loaded, the answer key is rotated $180°$. This option is meant to be used when the `forpaper` option is taken in the web package.

`savedata`: Saves two pieces of data: the seed used by the file `random.tex` to randomly reorder the questions and answers, and the answer key. These two values are saved to the file `\jobname_data.sav`.

- The seed value can be used to reproduce the exact randomization at a later time. Open the file `\jobname_data.sav` in your editor, it might look like this:

```
Initial seed: \randomi=482053344
Solution Key: 1--e; 2--s; 3--i; 4--u; 5--l; 6--D; 7--a; 8--z; 9--p;
```

  Copy the TEX assignment `\randomi=482053344` and paste it in the preamble, just below the `\usepackage{dps}` line. This should reproduce the same randomizations.

- The second line in the above verbatim listing is the answer key, for the randomization initiated by the seed on the first line. You can copy and paste this second line into the TEX/LATEX document and publish the solution in a separate document, at a later time. This is useful when publishing for paper.

`lang=english|german|custom`: Currently, there are only two language options `english` and `german`, with the option of a user create customization. To use the `lang=custom` option, you must create the file `dps_str_cus.def`. Do this by taking `dps_str_us.def` (or `dps_str_de.def`, if you prefer) and copying and changing its name to `dps_str_cus.def`. Open `dps_str_cus.def` in your favorite editor and change the text strings to a language of your choice, or change the strings that tickle your funny bone more than the ones provided.[4] To represent accented characters, use the octal encoding defined in the file `pd1enc.def`, as distributed with the hyperref bundle. The u-umlaut, for example, should appear in the file as `\string\374` and not as `\"{u}`.

[4] Or tickle your funny bone *less*, if you are a crusty one.

▶ For your convenience, the 'Appendix' on page 13 contains a listing of the octal codes for accented letters.

Here is an important option of web package.

`forpaper/forcolorpaper`: These are options of the web package, not the dps package. Then this option is taken, the puzzle created is meant for paper publication. No Acrobat forms will be created, the `showletters` option of dps is automatically taken. If you originally designed the game for the screen, you may have to rework the sizing and design of the game to fit everything into the constraint of a piece of paper. Landscape is an option to think about, depending on your design and the number of questions and answers.

8. Checking for validity

When creating the game, human error can sneak in. The most critical part is the `\Declare-Puzzle` command and getting the names of your fields set up the way you want. Letters with the same field name (the second parameter of the pair) will only need one question, they will all "light up" when the question is answered.

As explained earlier, after you've decided on your puzzle and the field names, you can then create your `Composing` environment using the helper command, `\writeComposingEnv`. Review the discussion in 'Begin composing the questions and answers' on page 5.

To help you lay out your design, use the `viewmode` option, possibly along with the `showletter` option. This gives you a nice preview and you can see where everything is located. You may have to adjust the parameter for `\insertPuzzle` to fit the puzzle into the allotted space. If enclosing puzzle, questions and answer in frames, there may have to be some adjustment of the depth of the controlling `minipage` environments, and so on, etc., etc., etc., and, of course, etc.

Assuming you have successfully posed questions and answers, and designed a layout for your puzzle, questions, and answers, you are ready to test it. Using the `nonrandomized` option is nice for checking your puzzle; the questions and answers are listed in the same order.

If you have customized the text using the `lang=custom` option, you need to check that your text displays correctly; this is especially important if your new text contains accented characters, such as our old friend ü. To test your customized string, open your puzzle document in Acrobat (Reader will not do here) and execute these JavaScript lines in the console:[5]

```
this.resetForm()
nMissed = 0;
nPenaltyPoints = 0;
nPassing = 4;
checkForFinished(nPassing)
```

By changing the variables `nMissed`, `nPenaltyPoints` and `nPassing`, and executing, you get the different messages appearing in the message text field.

[5] To execute from the console, open the console window by pressing `Ctrl+J`, paste in the code, highlight all the lines and press the `Enter` key on the keyboard, or `Ctrl+Enter` on the keypad.

9. Using the Web Package

The web package has a many features that can be utilized as a part of your overall puzzle design.

Web has a powerful template management system for inserting background graphics into a document, and a system for painting the background a color other than the default white.

Use the `\margins` and `\screensize` commands to set the dimensions of your puzzle game page:

```
\margins{<left>}{<right>}{<top>}{<bottom>}
\screensize{<height>}{<width>}
```

Enter the title and author's name, as well as other metadata:

```
\title{<doc title>}
\author{<doc author>}
```

See the documentation, `webeqman.tex`, of the AeB for details.

10. Thanks

My thanks to Jürgen Gilg, of u-umlaut fame, for his help and kind suggestions during the development of this game.

That's all for now. Hope you enjoy *Das Puzzle Spiel* and find it a useful learning tool.

Now, I simply must get back to my retirement. 𝔇𝔖

11. Appendix

The following is a subset of the PDFDocEncoding character set for PDF, these are useful for creating your custom localization file `dps_str_cust.def`, as discussed in 'Package Options' on page 10.

11.1. German Umlaute (dieresis)

Here a little tabular how to substitute the German Umlaute (dieresis) in PD1.

Ä	\string\304	Ö	\string\326	Ü	\string\334
ä	\string\344	ö	\string\366	ü	\string\374
ß	\string\337				

11.2. Accents

Here a little tabular how to substitute accents in PD1.

À	\string\300	È	\string\310	Ì	\string\314
à	\string\340	è	\string\350	ì	\string\354
Á	\string\301	É	\string\311	Í	\string\315
á	\string\341	é	\string\351	í	\string\355
Â	\string\302	Ê	\string\312	Î	\string\316
â	\string\342	ê	\string\352	î	\string\356
Ò	\string\322	Ù	\string\331	ë	\string\353
ò	\string\362	ù	\string\371	Ç	\string\307
Ó	\string\323	Ú	\string\332	ç	\string\347
ó	\string\363	ú	\string\372		
Ô	\string\324	Û	\string\333		
ô	\string\364	û	\string\373		