exams.sty Manual

Version [2014/09/19]

Laurence R. Taylor

September 2014

Contents

1 Introduction

This is the manual for the [2014/09/19] version of my test macros. This is a maintenance release of the [2013/03/01] version.

This package allows you to construct exams, quizzes, worksheets, etc. and then print several versions with permuted answers and/or permuted problems. It is designed to separate, as much as possible, the task of making up problems with the task of assembling your problems into one or more exams.

Feel free to change any default behavior you find objectionable but please change the `\ProvidesClass` so your `exams.sty` are not confused with mine.

This release contains this manual, the `exams.sty` file, a few default cover pages and a file `exam_short.tex`. The `exam_short.tex` can be used to write the exam even if the person using it does not have a copy of `exams.sty`. You can copy their problems directly into the exam you are assembling without retyping.

1.1 Getting started

Here is a typical file at the start of making up an exam.

```
\documentclass[12pt]{amsart}
\usepackage{exams}

\begin{document}
\begin{exam}

\end{exam}
\end{document}
```

You may have additional packages and macros as with any LaTeX file, but the above six lines of code are all you need to get started.

If other people are helping you write the exam and they do not have and do not want a copy of `exams.sty`, included in the usual distribution for this package is a file `exam_short.tex`. They can write their problems using a copy this file and you can copy and paste their code into one exam file.

2 Writing the problems

The first step in writing an exam is to produce the problems, and if there are multiple choice problems, the answers. The `exams.sty` code has been written so as to make this step as simple as possible.

Each problem is contained in a `problem` environment,

```
\begin{problem} ... \end{problem}.
```

Just write the TEX code for the problem. When you TEX the file the problems will appear numbered in the same order in which they appear in the file. If the problem is a partial credit problem (the student is expected to write out a solution), this is all you need to do.

However, inside the `problem`-environment you may write a solution:

```
\begin{solution} ... \end{solution}.
```

It is highly recommended that you write solutions but it is not required. You may also write solutions to some problems but not others.

If the problem is a multiple-choice problem, you will need a list of answers inside the `problem` environment. The syntax is:

```
\begin{answers} ... \end{answers}
```

End the code for each answer with a `\\`. You may have as many or as few answers as you like and the number of answers for each problem can be different. Two answers seems like the minimum for a real multiple choice problem but if you just want to write the correct answer, continue writing problems and come back at the end to put in wrong answers, you can do this. Answers will be permuted later (see §4).

The correct answer must always be the first one in the list.

2.1 Additional problem related environments

There are two other environments available but seldom used. They are

```
\begin{before problem} ... \end{before problem}
\begin{after problem} ... \end{after problem}
```

and as the names imply they put material immediately before a problem or immediately after. They are needed because anything between `\begin{exam}` and `\end{exam}` which is not inside an environment ends up on the first page by itself and this is never what you want.

The `after problem` environment is used if it is necessary to have something after the answers is a multiple choice problem.

The `after problem` environment is never necessary if there is no `answers` environment since its contents can be appended to the `problem` environment to achieve the same output. You can if you wish still use it.

Here is a display of all the environments associated to a single problem.

```
\begin{problem}
\begin{before problem}
...
\end{before problem}

PROBLEM TeX CODE
\begin{answers}
...\\
...\\
.
.
.
...\\
\end{answers}
\begin{solution}
...
\end{solution}
\begin{after problem}
...
\end{after problem}
\end{problem}
```

All environments are optional except for the outer `problem` one. The order of the pieces is irrelevant. In principal you can put `problem` TEX code anywhere in the `problem` environment outside of the other environments. There seems to be little reason not to put it before any `answer` or `solution` environments. If you prefer, the `solution` environment can come before the `answers`.

2.2 Before and after in the pdf file

There are two environments related to `before problem` and `after problem`.

`\begin{before pdf problem}{#1} ... \end{before pdf problem}`

`\begin{after pdf problem}{#1} ... \end{after pdf problem}`

These also put material before and after a problem but the #1 is the problem number in the pdf file. If you permute problems as explained in 4.3.2 this material stays with the number in the pdf file whereas the `before` and `after problem` material is permuted with the problem. The problem number #1 is required and this material can be put anywhere in the `exam` environment. If it turns out that there is no problem number #1 in the pdf file, the corresponding material is ignored.

One common use for `before pdf problem` comes up in quizzes, worksheets or handouts where a full blown cover page is unnecessary but a line or two giving the student a line on which to write their name and/or a brief title or a date can be produced by using a
`\begin{before pdf problem}{1} SOME TeX STUFF \end{before pdf problem}`
It will put `SOME TeX STUFF` before the first problem in the pdf file.

3 What happens when the file is TEXed

Here is a quick overview of what happens when you TEX the file. All the code before the `\begin{exam}` is processed. Then the optional variable for `\begin{exam}` is collected and the appropriate information is set. If the optional variable is not present some minimal default information is used. Otherwise the information is collected from the appropriate version file (as explained in 4).

Then the problems are read and put in `\vbox`'s for later. If there is an `answers`-environment, the answers are collected and `\vbox`'ed. Ditto for any `solution`, `before problem`, `after problem`, `before pdf problem` or `after pdf problem` environments. Only when TEX arrives at the `\end{exam}` are the boxes processed and written to the `pdf` file.[1] Since the environments are `\vbox`'ed, the TEX code in them is expanded when they are read which is long before they are shipped out. Details of how the various boxes are arranged in the final output are discussed beginning in the next section.

4 Versions of the exam

Once you have enough problems it is time to assemble them into an exam. The `exam.sty` package is designed to make it easy to produce multiple versions from the same collection of problems. Suppose for purposes of discussion that the exam file you have created is named `foo.tex`.

When you were creating problems and TEXing the file you had no control over the formatting but by creating additional versions you can gain control over the output. Additional

[1] *Except for anything not inside one of the above environments. This is written as it is processed and so ends up on the first page by itself.*

versions are numbered by positive integers. The actual `pdf` output file is controlled by the optional argument to the `exam` environment and by an associated version file.

First you need to make a folder named **`versions`**
in the same directory as **`foo.tex`**.

Initially the output file, `foo.pdf`, has little spacing between the problems, always has the first answer as the correct one and solutions are printed if you wrote them. You may want to give the students solutions after the exam but this is not so desirable during the exam itself. This default behavior, with a few exceptions, is difficult to change. To produce better output, it is time to create versions.

You may use any positive integer you like for a version but to keep things simple, let us start by producing version `1`. Chance `\begin{exam}` to `\begin{exam}[1]`. When you TEX `foo.tex`, the look of the output will be rather different. Each problem is on its own page and answers are evenly spaced, three on the first line and two on the second. Problems and answers appear in the same order in which they occurred in the `foo.tex` file. Solutions are gone.

You certainly want to permute the multiple choice answers since otherwise all the correct answers will be (a). You may also wish to permute the order of the problems or even not use some of the problems in version `1`. The spacing of the answers may need to be cleaned up since it is entirely possible that answers will overlap as currently typeset. You may wish to have more problems on a page.

All of this is fixed in the version file and by additional changes to the optional variable in the `exam` environment.

4.1 The optional argument to the exam environment

The `exam` environment takes an optional argument, `\begin{exam}[v.c.s]`,

4.1.1 The `v` piece

The value of `v` is the version number. It can be any non-negative integer and versions are discussed in more detail in §4.2.

4.1.2 The `c` piece

The value of `c` is explained in the cover page section (§5.2) and can be ignored for now.

4.1.3 The s piece

The value of `s` controls what is done with the solutions. If `s`=1 the solution is written immediately after the problem. If `s`=2, all the solutions are written at the end of the file. Any other value results in the solutions being ignored.

4.1.4 Default behavior for the optional argument

The default value for the optional `exam` argument is `0.0.1`: `v`=0 for version 0 and `s`=1 which writes the solution immediately after the problem. The value of `c` is irrelevant in version 0. If you do not want the solutions printed, `\begin{exam}[0.0.0]` removes them.

Missing entries in the `exam` argument are filled in with default values.

- If `v` is not missing, then missing `c` and `v` entries are set to 0.
  For example, `\begin{exam}[0.0.0]` and `\begin{exam}[0]` are equivalent.
- If the `v` entry is missing then `c` is always set to 0 and if `s` is missing, it is set to 1.
  Hence `\begin{exam}[...]` , `\begin{exam}[..]`, `\begin{exam}[.]`, `\begin{exam}[]` and `\begin{exam}[0.0.1]` are all equivalent.

4.2 The version file

If you look in the `versions` directory after you TEX `foo.tex` with `v`=1, there is now a file, `1.ver` in the `versions` directory. (You can change this behavior as described below, but for now let's assume you have made a `versions` directory.) This is the version file for version 1. When you change `\begin{exam}[1]` to `\begin{exam}[2]` and reTEX, there will be a new file, `2.ver`.

This is not the most efficient way to proceed. It is easier to fix up `1.ver` *some and then just copy and change the* `1` *to other numbers to get other version files.*

There are two steps I like to do to version 1 before I copy to create the other version file(s). To explain these, open `1.ver` in the word processor you use to create `foo.tex`. Strictly speaking you can use any word processor that will save the file as a plain text file but the word processor you use to create `foo.tex` has this property.

The `1.ver` file is readable but a bit strange. First there are lines pertaining to the typesetting of answers.

4.2.1 AnswerData fields

If the are no multiple choice problems there will be no lines of the sort and you may skip to 4.2.2.

4.2.1.1 AnswerData.Perm lines

First there will be some lines that look like

```
\namedArgs{#_tex.AnswerData}{{Perm}->{{1}{2}{3}{4}{5}}}
```

There is one of these lines for each multiple choice problem and `#_tex` is a number which is the number of the problem in the `foo.tex` file. As we shall see, this may not be the number of the problem in the `foo.pdf` file but `#_tex` always refers to the number of the problem in the `foo.tex` file. For now you should ignore these lines except to note that in this example, problem number `#_tex` has 5 answers. In general you should see a list of consecutive integers from 1 to the number of answers for the problem.

4.2.1.2 AnswerData.perLine lines

. Next come lines

```
\namedArgs{#_tex.AnswerData}{{perLine}->{3}}
```

These are the lines which set the answer spacing to `3` on a line. Change these numbers as desired to get better spacing of the answers. More control is available if needed. Instead of one number you can have a comma-separated list of numbers. *It is vital that there be no comma after the last entry.* As a first step, no matter what is in this variable, it is strung out with commas so '3' becomes '3,3,3,3,. . . 3' and a '2,1' becomes '2,1,2,1,2,1,. . . '.

The first number is peeled off and is used to compute the size of each answer. Then the next number is peeled off and that number of answers is put on the first line; then the next number is peeled off and that number of answers is put on the second line; etc.

If there are 5 answers, '3' results in 3 evenly spaced answers on the first line and the other 2 on the second line with the same spacing. The '2,1' puts one answer on the first line and 2 answers on the next 2 lines with the available space split evenly between the two answers.

4.2.1.3 AnswerData.Marks & skip lines

The next sort of lines look like

```
\namedArgs{#_tex.AnswerData}{{Marks}->{{a}{b}{c}{d}{e}}{skip}->{\vskip 10pt}}
```

The `Marks`-field are how the answers are to be labeled and you may change these to suit yourself. Since there are 5 answers in our example, you need 5 marks although if there are more the additional ones will just be ignored.

Notice in the argument for `Marks` only the a, b, c, etc. appear. The formatting for how these marks appear in the exam and on the cover page is handled by three macros described below in 5.5.

The `skip`-field is the amount of additional space between two lines of answers. Again you may adjust this to suit yourself.

Then there is one type of line which occurs for each problem in the `foo.tex` file.

4.2.2 ProblemData field

There is one line of this sort for each problem.

```
\namedArgs{#_tex.ProblemData}{{pts}->{}}
```

Each problem in the `foo.pdf` file is labeled with a number and the contents of this field. It is called `pts` because I use it to print the point value of the problem, but it can contain any legal TEX code that can be evaluated when the problem is read.

There are some additional lines which you should ignore for now. When each problem in version 1 looks good when TEX'ed, copy the version 1 file and rename it for as many versions as you need.

4.3 Fine tuning the version file

Each version file now has to be handled separately.

4.3.1 Permuting answers

The first step is to permute the answers. Assume by way of example that problem `#_tex` has 5 answers. The default value of the `AnswerData.Perm` line is

```
\namedArgs{#_tex.AnswerData}{{Perm}->{{1}{2}{3}{4}{5}}}
```

Changing this line to

```
\namedArgs{#_tex.AnswerData}{{Perm}->{{3}{5}{4}{1}{2}}}
```

has the following effect. The first answer in the `foo.tex` file for problem `#_tex` is the third answer in the `foo.pdf` file. The second answer in the `foo.tex` file is the fifth answer in the `foo.pdf` file. The third answer in the `foo.tex` file is the fourth answer in the `foo.pdf` file. The fourth answer in the `foo.tex` file is the first answer in the `foo.pdf` file. The fifth answer in the `foo.tex` file is the second answer in the `foo.pdf` file.

It is not difficult to write a script that will produce these lines with random permutations.

You should now reTEX the file to insure that the spacing for the answers is still good. If all the answers are roughly the same size there will be no problem but if the sizes are rather different, altering the printed order may spoil the spacing. If needed you can either change the spacing using

```
\namedArgs{#_tex.AnswerData}{{perLine}->{?}}
```

change the permutation by hand or use one of the more advanced `perLine` options discussed in 4.2.1.2.

4.3.2 Choosing, spacing and permuting problems

Once your answers are permuted satisfactorily it is time to consider the

```
\namedArgs{#_pdf.PermutedProblemData}{{fileProblem}->{#_tex}{spaceAfter}->{\vfill \eject }}
```

lines. By default `#_pdf` = `#_tex` but `#_pdf` is the number of the problem in the `foo.pdf` file. Hence the `PermutedProblemData` lines should start with `#_pdf` = 1, followed by `#_pdf` = 2, `#_pdf` = 3, and so on until you have the number of problems you want in the `foo.pdf` file.

You may use any subset of the problems in the `foo.tex` *file you want and you may change them with each version.* If you leave the `fileProblem`-fields with their default values all the problems will be printed in the same order as they occur in the `foo.tex` file.

You need one `pdf.PermutedProblemData`*-line*
for each problem you want on the printed exam,
numbered consecutively starting with 1.

You do not have to use all the problems in the `foo.tex` *file.*

The `spaceAfter` field adds a spacing command after the problem. The `\vfill\eject` produces a new page. A command of just `\vfill` will put the next problem on the same page and divide the vertical space evenly. You could use `\vskip 80pt` to force an 80 pt space after the problem and let TEX do the rest.

5 Cover pages

Cover pages can be added to the front of the `foo.pdf` file if desired.

First you need to make a folder named `front_page`
in the same directory as `foo.tex`.

If there is a `front_page/cover.v.page` file in this folder and if the version is `v`, then a cover page is added for this version. If there is a `front_page/cover.page` file in this folder but not a `front_page/cover.v.page` file then this file is used to create the cover page(s).

Precisely, when running version `v`, the program looks first for a `front_page/cover.v.page` file and if it is present, it is used for the cover page. If there is no such file, then, if the file `front_page/cover.page` is present, it is used. If neither file is present, then no cover pages are added, even if requested.

There are two kinds of cover pages. One can be used as a cover page for the actual exam. A second sort looks like the first unless there are multiple choice questions. In this case the correct answer for each multiple choice question is marked so it can be used to prepare templates for grading the exam.

The cover page is responsible for printing itself.

All the `foo.tex` file does is load the requested cover page file which is then responsible for producing the desired output. With a bit of work you can produce any sort of cover page you may need.

There are some values and commands available to the cover page file:

- Values: (as macros not counters)
  - `\finalNumberOfPages` (the number of pages in the exam)
  - `\finalNumberOfProblems` (the number of problems in the `foo.pdf` file)
  - `\numberOfProblemsInFile` (the number of problems in the `foo.tex` file)
- Commands:
  - `\ifmarkedAnswerSheet`
  - `\isPartialCredit#1`, `\PartialCredittrue` and `\PartialCreditfalse`
  - `\multipleChoiceLine#1#2`, `\partialCreditItem#1` and `\markedPartialCreditItem#1`
  - `\partialCreditItem#1` and
  - `\markedPartialCreditItem#1`

5.1 Values

There is a small caveat when using `\finalNumberOfPages`. It is possible that different versions may have different numbers of pages but by the time the exam is TEX'ed the cover page is long gone. This can also happen if you TEX once with solutions and then without them (or vice versa).

To solve this problem, the number for the previous run is stored in the `foo.aux` file and since the cover page is written before the exam proper, the number passed to the cover page code is the number from the previous run. (If the current version was not TEX'ed last, the value is a blank.) If you use `\finalNumberOfPages` in your cover page code you must TEX the file twice whenever you change anything that affects the number of pages to be sure this number is set correctly.

5.2 `\ifmarkedAnswerSheet`

`\ifmarkedAnswerSheet` is a `TEX-if` which is true if the cover sheet should be marked and false otherwise. To set it you use the `c` piece from the argument to `\begin{exam}[v.c.s]`. If `c`=1 an unmarked sheet is produced but if `c`=2 a marked sheet is produced: `c`=2 is useful if you want to hand out a copy of the exam with the correct answers marked on the cover page.

More useful is c=21 which produces two pages at the start of foo.pdf. The first is marked and the second is unmarked. It is expected that the marked version will be removed and used to construct templates for grading the exam and the exam itself will consist of the unmarked cover sheet and the problems.

The \finalNumberOfPages in this version is the number of pages in the foo.pdf file if c is not 21, but if c = 21 it is the number of pages after the marked answer page is removed. In other words, it is the number of pages in the exam as passed out to the students.

5.3 \isPartialCredit#1, \PartialCredittrue and \PartialCreditfalse

The command \isPartialCredit#1 behaves like a TEX-if. The argument is the number of a problem in the foo.pdf file. Usage is

\isPartialCredit{#_pdf} PC \else MC \fi

If the problem number #_pdf is a partial credit problem, the PC code is executed but if it is a multiple choice problem the MC code is executed. The PartialCredit flag is set to \PartialCredittrue or \PartialCreditfalse as appropriate. This value is retained until the next \isPartialCredit#1 is executed.

5.4 \multipleChoiceLine#1#2

#1 is the number of the problem in the foo.pdf file and #2 is a width. What is returned is the array of Marks for that problem, evenly spaced across the width and formatted as described next.

5.5 \partialCreditItem#1 and \markedPartialCreditItem#1

#1 is the mark to be formatted. The default is (#1) unless we are doing a marked answer sheet and this is the correct answer. Then the default is (•). You are free to redefine either of these commands to produce a different looking cover page.

The formatting of the mark next to the answer in the actual exam is handled by \lineMark#1. The variable #1 contains the mark from 4.2.1.3. The default is (#1).

5.6 Additional cover page information

If you need the version number in your TEX code for the cover page, you can get it with \value{VERSION}.

For example \ifmarkedAnswerSheet{Version \#\number\value{VERSION}}{} will print the version number on the marked answer sheet.

You can alter the appearance of the cover page for different versions either by using \value{VERSION} and TEX macros to get different output for different versions or make a cover.v.page for each version.

5.7 Cover page caveats

Because the cover page code is loaded after the preamble any packages needed by it must be loaded in the preamble for `foo.tex`. Since it may be loaded twice (if `c`=21) it should not do any `\new`'s either. Any `\newcommand`'s or `\newcounter`'s or such need to be in the `foo.tex` file. `\def`'s are allowed since LaTeX does not object to a `\def` which has already been defined.

Another point to remember is that the cover page is responsible for ending itself so except in rare cases it needs to end with a `\newpage`.

Any headers/footers you want in the exam need to be defined after the `\newpage` for the cover page. If you try to put then in the `foo.tex` file they will also be used for the cover page. You could set the headers/footers in the cover page code, as the supplied examples do, but then these values will be used in the rest of the file. The supplied cover page examples use `fancyhdr` for both the cover pages and to set the headers/footers for the exam. Hence to use the supplied examples of cover pages you will need to have a `\usepackage{fancyhdr}` in the preamble of the `foo.tex` file.

6 Fine tuning the appearance of the marks.

Any problem which has answers has an associated list of marks from the
`\namedArgs{#_tex.AnswerData}{{Marks}->{{a}{b}{c}{d}{e}}{skip}->{?}}`
line in the version file. When the answers for this problem are written in the `foo.pdf`-file, each answer is preceded by one of these marks, in order. The first answer on the page uses the first mark in the list and so on. You are free to use any labels you like.

A default line is prepared when you first do a run with a new version number. It just takes its values from `\defaultMarks` defined by `\gdef\defaultMarks{abcdefghi}`. If you routinely use a different collection of marks you can change the `\defaultMarks` in the `exam.sty` file.

Recall (here) that the mark next to the problem is formatted by `\linkMark#1`.

7 Additional commands

7.1 For loops

For-loops can be useful in writing cover pages and since they are used in the `exam.sty` code they have been made available to you. The syntax is

```
\forloop[5]
```

so there is one optional argument and 4 non-optional ones. The optional argument is the step counter for the loop and is `1` by default. The second argument (which is the first non-option one) is the start value for the counter. The next argument is the end value and the next is the name of the counter to be used for the loop. The last argument is the code for the loop.

Since LaTeX counters are global, you must be careful that none of the code for the loop does things to your counter in unexpected ways.

By way of example, here is a bit of code which prints the multiple choice lines to be marked on the cover page.

```
\forloop{1}{\number\finalNumberOfProblems}{tmpLRTC}{%
\isPartialCredit{\value{tmpCounter}}%
\else%
\multipleChoiceLine{\value{tmpLRTC}}{6in}%
\par%
\fi%
}
```

You can find additional examples in the included sample cover pages.

7.2 Writing additional versions in one file

After you have gotten all your versions ready to be duplicated it is sometimes convenient to have a single `pdf`-file with all the versions, one after the other.

To achieve this, after the `\end{exam}` you can put

```
\WriteNewVersion{?}
```

where the question mark is a `v.c.s`.

For example,

```
\begin{exam}[1.21]
.
.
.
\end{exam}
\WriteNewVersion{2.21}
\WriteNewVersion{3.21}
\end{document}
```

writes version 1, followed by version 2 followed by version 3.

Another example:

```
\begin{exam}[1.21]
.
.
.
\end{exam}
\WriteNewVersion{1.0.1}
\end{document}
```

writes one version with cover sheets for the actual exam followed by the same exam with solutions which can be passed out after the exam.

7.3 Writing additional versions caveats

If you TEX multiple versions at once, the `\finalNumberOfPages` is stored for each version and the code retrieves the correct value for that version. Hence if you TEX twice, each version will have `\finalNumberOfPages` set correctly for that version

EXCEPT

When you do something like the last example above. The `\finalNumberOfPages` in the `\WriteNewVersion{1.0.1}` will not be set correctly and the second time you TEX the file `\finalNumberOfPages` will not be set right for the first copy of the exam. More generally, anytime you put together a file which does two prints of the same version, the `\finalNumberOfPages` for that version is set to the number of pages for the last version you TEX'ed.

One solution in such a case is to take an unused version number, say 5, copy `ver.1` to `ver.5` and replace `\WriteNewVersion{1.0.1}` with `\WriteNewVersion{5.0.1}`. TEX twice and all will be well. Because `ver.1` and `ver.5` have the same information, they will print the same exam except version 1 has no solutions and version 5 has solutions. The number of pages for version 1 is the number for the exam and the number of pages for version 5 is the number for the exam with solutions. In fact, there is no need to copy 1.ver. Just make a 5.ver file with one line of code: `\input versions/1.ver`.

The code

```
\begin{exam}[1.21]
    .
    .
    .
\end{exam}
\WriteNewVersion{2.21}
\WriteNewVersion{3.21}
\end{document}
```

works correctly because the `\finalNumberOfPages` is stored for each version the first time the file is TEX'ed and the second time the correct number of pages for that version is retrieved.

There is one final potential issue. When a problem is read, the first line is indented a distance to allow for the `\namedArgs{#_tex.ProblemData}{{pts}->{CODE}}` to be written after the problem number and before the problem proper. This is set by the first version TEX'ed so if a second version is TEX'ed the space for the first version is used. This should

not be a problem unless you use wildly different CODE for the same problem in different versions.

A Dependencies

`exams.sty` requires the following packages:

- `etex.sty`
- `ifthen.sty`
- `geometry.sty`

A Google search will find a location from which you may download any package you may need. `geometry.sty` could be removed with a little recoding but `ifthen.sty` is pretty deeply embedded. `etex.sty` is needed because the code makes profligate use of boxes and counters and so even 15 multiple choice questions will overwhelm TEX's default allotments.

All these packages are included in most standard TEX distributions so probably `exams.sty` will work "out of the box".