writing-exams-in-orgmode

John Kitchin

October 23, 2013

Contents

1 Writing exams in org-mode

There are a few aspects of writing exams that are tedious the way I normally do it. Typically I write exams in Word because I can easily see the layout, how much whitespace there is to write answers in, etc... I like to put a gradesheet on the final page which lists each problem, the points it is worth, and a blank to put the grade for that problem into. It is tedious to go back through 10 pages of questions to look up all the points, enter them in, add them up, etc... And if I decide to renumber the questions or move them, it must be repeated.

Construction of the grade sheet ought to be automated. That is not going to happen in my hands in MS Word. I have seen this done in LaTeX in the Acrotex educational package, but I don't see myself hacking LaTeX any time soon either. Enter org-mode. I could see writing the exam in org-mode, and writing some code to construct the grade table.

The idea is that we would store the points in a PROPERTY of an org-heading. Then, to create the table, we just loop through the headlines,

gather the levels and points, and then construct a table to put in the exported LaTeX. The next section contains some exam questions that will form the basis of this post.

1.1 The exam section

1.1.1 Do you get it?

What is the answer to the universe?

1.1.2 Multipart question

1. Circle the best answer What is 1 + 1? a) 2 b) 3 c) 4
2. Describe a cat.

1.1.3 Essay question

Expound on the meaning of life.

1.2 Back to the grade table construction

We will parse the buffer to get the headlines, and then extract some properties from each headline. For each headline that has points we will print the title and points it is worth, and finally the total number of points.

```
(setq total-points 0)    ; counter for the total points

```

```
;; now loop over headlines
(org-element-map
    (org-element-parse-buffer 'headline) 'headline
  ;; function to print headline, level and points
  (lambda (headline)
    (let ((points (org-element-property :POINTS headline))
          (title  (org-element-property :title headline)))
      (if points (progn
                   (setq total-points (+ total-points (string-to-number points)))
                   (princ (format "title=%s\nPOINTS=%s\n\n" title points)))))))

(princ (format "Total points = %s" total-points))
```

```
title=Do you get it?
POINTS=5

title=Circle the best answer
POINTS=10

title=Describe a cat.
POINTS=4

title=Essay question
POINTS=25

Total points = 44
```

That is the foundation for the grade table. What I would like is the grade table to be structured like this:

problem	Points	grade
ref to heading	#points	

Where the `ref to heading` is the same number as the actual heading. During the export, labels are generated for each section, which we could refer to. To take advantage of this we need to figure out what the labels are, so we can refer to them.

After quite a bit of hackery, I figured out how to access this information [1]. It is not readily apparent this is how to do it, but it works!

```
(let* ((info (org-export-collect-tree-properties (org-element-parse-buffer 'headline) '()))
       (headline-numbering (plist-get info :headline-numbering)))
```

[1] This code does not generate correct labels for headlines with TODO in them, or for footnotes.

```
(org-element-map (plist-get info :parse-tree) 'headline
  (lambda (headline)
    (format "\\ref{sec-%s}" (mapconcat
                      (lambda (x) (format "%s" x)) (cdr (assoc headline headline-numbering)) "-")))))
```

```
("\\ref{sec-1}" "\\ref{sec-2}" "\\ref{sec-2-1}" "\\ref{sec-2-2}" "\\ref{sec-2-2-1}" "
```

So, let us try putting this all together [2].

```
(setq total-points 0)    ; counter for the total points
(princ "#+caption: The grade table\n")
(princ "#+attr_latex: :align |c|c|c|\n")
(princ "|-\n")
(princ "|Problem|Possible Points|Points earned|\n|-\n")
(let* ((info (org-export-collect-tree-properties (org-element-parse-buffer 'headline) '()))
       (headline-numbering (plist-get info :headline-numbering)))
  (org-element-map (plist-get info :parse-tree) 'headline
    (lambda (headline)
      (let ((ref (format "\\ref{sec-%s}" (mapconcat
                                          (lambda (x) (format "%s" x)) (cdr (assoc headline headline-numbering)) "-
            (points (org-element-property :POINTS headline)))
        (if points (progn
                     (setq total-points (+ total-points (string-to-number points)))
                     (princ (format "|%s|%s|  |\n|-\n" ref points))))))))

(princ (format "| |Total points| %s|\n" total-points))
(princ "|-\n")
```

Table 1: The grade table

Problem	Possible Points	Points earned
1.1.1	5	
1	10	
2	4	
1.1.3	25	
	Total points	44

Note this table gets munged by Mathjax in HTML. I am not sure that is fixable. You can open the pdf and see the results.

This works ok. There is still some work to be done. For example the boxes in the grade table are not very large. The references are a little odd in this case, but that is an artifact of the fact that you cannot nest a section deeper than 3 levels in LaTeX without some work, and I nested my exam

[2] Note this table gets munged by Mathjax in HTML.

section in a second level heading so it would appear in the blog post. A real application of this would not have all these other sections, and would not export the build section. It is a tad tedious to hand build the table, but not too bad.

It might be better to use CUSTOM_ID labels in the sections, rather than try to build up the references. You still need to think about what the labels would be, and we are used to seeing numbers!

1.3 building the pdf

I have gotten in the habit of building the latex file from commands, and manually running them with C-c C-c.

```
(let ((org-latex-default-packages-alist
       '(("" "minted" nil)
         ("linktocpage,
  pdfstartview=FitH,
  colorlinks,
  linkcolor=blue,
  anchorcolor=blue,
  citecolor=blue,
  filecolor=blue,
  menucolor=blue,
  urlcolor=blue" "hyperref" t)))
      (async nil)
      (subtreep nil)
      (visible-only nil)
      (body-only nil))

  (org-latex-export-to-latex async subtreep visible-only body-only '()))
```

```
(progn
  (shell-command "pdflatex -shell-escape writing-exams-in-orgmode")
  (shell-command "pdflatex -shell-escape writing-exams-in-orgmode"))
```