The Canvas learning management system and LaTeXML

The LaTeX workflow is still the best

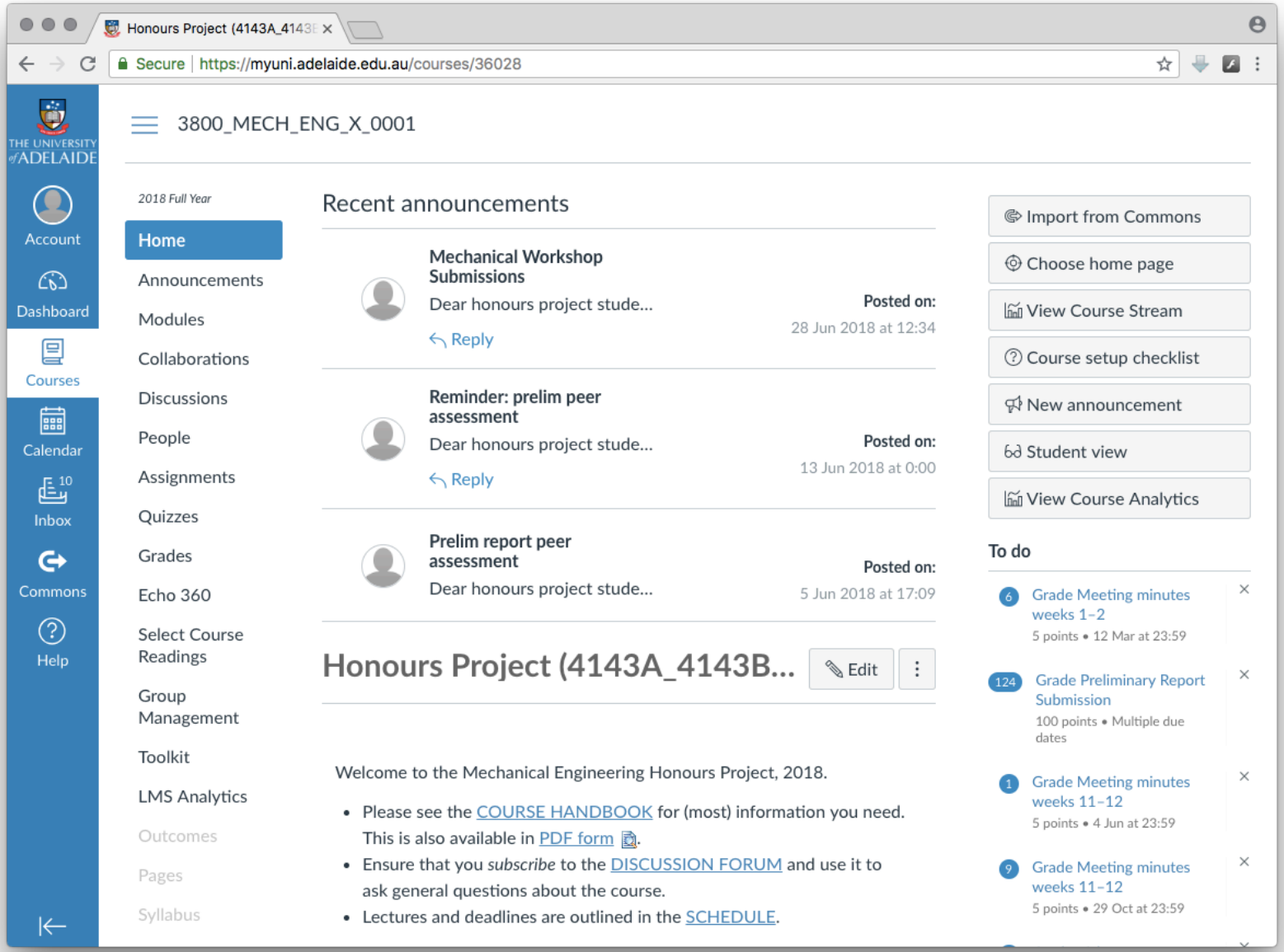
Will Robertson

July 20, 2018

Background

- The University of Adelaide recently switched to *Canvas* as its LMS
- The LMS has an API!
- I wanted to sensibly manage ‘content’
- Generate both HTML and PDF
- A good excuse to learn LaTeXML…

This is an LMS



Motivation

1. In the old days, had a monolithic (impossible) Word file
2. Care and maintenance of content
   - Click click click
   - No good tools to manage content globally
3. Re-use of content for a comprehensive PDF
   - 'One stop shop' for reference
   - Handy to be able to distribute
     (for students, for supervisors, for accreditation)

- One source, multiple outputs
- Macros to ensure up-to-date info
  - Names of certain people
  - Due dates
  - Weightings of assessments
- Reliable (easy setup, actively developed, etc.)
- LaTeX based (sorry, not sorry)

A number of LaTeX to HTML possibilities, including:

- TeX4ht — (?), in TeX Live
- lwarp — Lua, in TeX Live (951 p. manual)
- LaTeXML — Perl, not in TeX Live
- HEVEA — OCaml, not in TeX Live

(Non-exhaustive list. All are actively developed.)

- Hopefully the choice is not too important!
- I.e., the conversion aspect should be modular
- I wanted to try LaTeXML, and I've been happy (enough) with it to date.

Architecture of LaTeXML

Caveat: I'm only a user!

- Reimplements some TeX scanning in Perl
- Therefore handles basic LaTeX 2_ε programming
  - `\newcommand`
  - `\newcounter`, `\stepcounter`
  - etc.
- Provides Perl interfaces to emulate classes and packages

A combination of LaTeX 2_ε, Perl, XSLT, HTML, CSS, …

Anyway, the basics works well:

```
\newcommand\honourscoord{Will Robertson}
```

Running LaTeXML

Two-phase process: `latexml` then `latexmlpost`

```
latexml tex/$FILENAME.tex | latexmlpost - \
  --xsltparameter=SIMPLIFY_HTML:true \
  --sourcedirectory=tex \
  --format=html5 \
  --destination=html/$FILENAME.html \
  --splitat=chapter \
  --splitnaming=label
```

Canvas programming

- A so-called REST API
  Wikipedia: 'REST-compliant web services allow the requesting systems to access and manipulate textual representations of web resources by using a uniform and predefined set of stateless operations.'
- Using `curl`:

  ```
  curl -X GET -H "$CANVASAUTH" $CANVASCOURSE/$1
  ```

  - `$CANVASAUTH` = secret token
  - `$CANVASCOURSE` = URL to course
  - `$1` = `users` or `rubrics` or `assignments?search_term=charter` etc.
- E.g.:

  ```
  curl -X GET -H 'Authorization: Bearer 81...Jx'
       https://myuni.adelaide.edu.au/api/v1/courses/36028/assignments
  ```

Uploading a file to *Canvas*

Clear as mud

```
curl -X POST  -H "$CANVASAUTH" "$CANVASCOURSE/files"\
     -F "name=$1" -F "parent_folder_path=upload" > tmp.json ;
URL=`cat tmp.json | jq '.upload_url'` ;
KEYS=`cat tmp.json | jq '.upload_params' | jq -r -j \
  "to_entries | map(\"-F \(.key)=\(.value|tostring) \")|.[]"` ;
echo curl -D response.tmp $URL $KEYS -F file=@$1 | bash ;
LOC=`sed -n -e 's/Location: \(.*\)/\1/p' response.tmp`;
LOC=${LOC%$'\r'}
curl -X POST -H "$CANVASAUTH" "$LOC" | jq ;
```

- I have forgotten how this works!
- Allows me to, say, upload the typeset PDF automatically after updating content.

Evolution of my support scripts

- Started with `curl`
- A few small-ish Bash functions and scripts
- Have now started with Lua programming
  - Requesting data with many items (‘all submitted assignments’, say) is returned in multiple ‘pages’ so iteration is required
  - Much prefer doing real programming not in Bash
  - Start looking into the Lua package ecosystem

Package proliferation: not just a LaTeX problem

- `luajson` by harningt
- `lunajson` by grafi
- `rapidjson` by xpol
- `dkjson` by dhkolf
- `jwt-jitsi` by pawelgawel88
- `JSON4Lua` by luarocks
- `jwt` by olivine-labs
- `mjolnir._asm.data.json` by _asm
- `ngxjsonform` by rtbz
- `dromozoa-json` by moyu
- `lua-cjson-ol` by olivine-labs
- `lua-cjson2` by CriztianiX
- `json-lua` by jiyinyiyong
- …

Lua equivalent to `curl`

```
local http   = require("ssl.https")
local ltn12  = require("ltn12")

local body, code, headers, status = http.request{
    method = "GET",
    url = canvas_url .. req .. "?" .. opt,
    headers = {
      ["authorization"] = "Bearer " .. canvas_token,
      ["content-type"]  = "application/json"
    },
    sink = ltn12.sink.table(canvas_result),
}
```

- This returns JSON, which is 'decoded' into a Lua table.

The source

```
\documentclass{report}
\usepackage{latexml}
...
\begin{document}
\title{...}\author{...}\date{...}
\maketitle
\tableofcontents

\input{../texdata/data-limits.tex}
\input{../texdata/data-marks.tex}

\part{Introduction}
\label{part-intro}

\input{../pages/introduction}
\input{../pages/course-schedule}
\input{../pages/week-planner}
```

- Each `.tex` file contains one chapter:

  ```
  \chapter{Introduction}
  \label{introduction}% same as filename!
  ...
  ```

- LaTeXML does not convert file by file
- Rather, output HTML is split by chapter
- With consistent naming, this produces one `.html` file per `.tex` file
- A 'table of contents' page is also generated

Top matter

```
<!DOCTYPE html>
<html>
<head>
  <title>XX Chapter title</title>
  <meta http-equiv="Content-Type"
        content="text/html; charset=UTF-8">
  <link rel="stylesheet" href="LaTeXML.css"
        type="text/css">
  [...]
</head>
<body>
```

Bottom matter

```
<footer class="ltx_page_footer">
  [...]
</footer>
</div>
</body>
</html>
```

‘Content’

```
<div class="ltx_page_main">
<header class="ltx_page_header">
  [...]
</header>
<div class="ltx_page_content">
  <section class="ltx_chapter ltx_authors_1line">
    <h1 class="ltx_title ltx_title_chapter">
      <span class="ltx_tag ltx_tag_chapter">Chapter X </span>
      Title of chapter goes here
    </h1>
    <div class="ltx_date ltx_role_creation"></div>
    <section id="S1" class="ltx_section">
      [...]
      ALL THE CONTENT
      [...]
    </section>
  </section>
</div>
```

‘Content’

```
<section class="ltx_chapter ltx_authors_1line">
  <h1 class="ltx_title ltx_title_chapter">
    <span class="ltx_tag ltx_tag_chapter">Chapter X </spa
    Title of chapter goes here
  </h1>
  <div class="ltx_date ltx_role_creation"></div>
  <section id="S1" class="ltx_section">
    [...]
    ALL THE CONTENT
    [...]
  </section>
</section>
```

One line of awk

- Snipping is done with

```
awk '/\<section.*\>/,/\<\/section\>/' \
  html/$BASE >> snip/$BASE
```

- Possible to take the raw XML and develop own workflows, but unnecessary for me since I want HTML anyway.

Not much to say here:

- Using sensible file structure, match up location of images for the PDF on the disk, and for the HTML in the server
- Same thing for files, but currently I don't link files! (TODO)

Uncertainties

- Maths? Not well supported in *Canvas*, yet. That's okay — for this course I don't need maths.
  But this is really important!!
- Bash scripts? One step at a time.
- All Lua, eventually — cross-platform and a bit more sensible.
- How far do I go writing a general *Canvas* interface?

Is LaTeXML the right approach?

- The Perl layer is a little foreign to me but appears well-designed.
- The XML output appears highly flexible.
- In the long-run, what does LaTeX itself need to provide?
- Could LaTeXML be written in TeX itself? Or LuaTeX?

Live demo???

(Not sure if this is a good idea…)

Summary

- From LaTeX (to PDF, and) to LaTeXML to *Canvas*.
- LaTeX is still the pre-eminent document preparation system.
- LuaTeX opens the door for a tightly integrated approach for bidirectional transfer of information between LaTeX and *Canvas*.