Eliot Scott
Instructor Quinn Stewart
Survey of Digitization - 385R – Spring 2011
5 May 2011

Creating Archival PDF's from Crowd-Sourced Transcriptions via the Drupal CMS and domPDF

I was approached to work on a web-based digital archive for the Congregational Church of Austin and made it into an independent study. The archive had several requirements which we discussed. The first was that the archive could be easily edited and added to by general computer users at the church. The second was that the archive would be inexpensive to maintain. The third was that the work done on the archive could be easily portable to other platforms as computer technologies evolved. Given these requirements, I recommended that the digital archive architecture be built around an open source content management system. As I was familiar with both the Drupal CMS and its underlying architecture built on the open source LAMP (Linux, Apache, MySQL, PHP) platform, I felt it was an excellent choice to manage their archive.

Finding the right software

In the initial project plan, we planned to scan and perform OCR (Optical Character Recognition) on the archival documents with church members performing the majority of the transcriptions to clean the OCR'ed text in an as yet undefined OCR package. The scanned documents with their machine readable transcribed text would then be converted to the PDF format to allow for a high quality single file to be easily stored on both the web and on in house storage devices. The images and text would then be put into the online archive separately for greater accessibility.

After evaluating several OCR packages, including ABBY, Acrobat, PDF OCR, Free OCR and DocQ, the only one that came close to fulfilling OCR requirements for minimal transcription on documents dating back to the early 1900's was ABBY Fine Reader, and even ABBY required a fair amount of user training on letters for each individual document to achieve even moderate accuracy. As it was impractical to expect church members to purchase pricey software individually or to spend hours doing OCR cleanup at the church archives or to learn the intricacies of ABBY's user training function, and no other packages seemed to do the job even remotely adequately, we needed to find an alternative solution.

As I needed to create an archival PDF file with a high quality image and machine readable text stored in a single document for simple storage and access, and the church had a limited budget, as well as an inexperienced user base, the solution to this problem needed to be simple yet elegant, affordable yet powerful. I evaluated a number of PDF packages including Acrobat, FoxIt, NitroPDF, PDF OCR, and DocQ to see if any of these packages would allow users to easily and cheaply edit PDF text after an initial OCR pass by one copy of ABBY. None did.

So I turned my attention toward open source PDF generators - TCPDF, Latex, and wkhtmltopdf. Latex had a long history in converting computer code to print and looked promising, however it converted the code into another file type before it could generate a PDF and felt needlessly complicated, so I elected to evaluate it last. Both TCPDF and wkhtmltopdf, along with another open source PDF project named domPDF, worked within a Drupal module entitled "Printer, e-mail and PDF versions" ('Print' for short) so I elected to try those.

Unfortunately the Print module page (http://drupal.org/project/print) stated that wkhtmltopdf, although the best of the three packages, could not be run on a shared server and was resource hungry. As the church could only muster paying for shared server space, leaving wkhtmltopdf a non-option, I elected to try TCPDF first. Initial tests with TCPDF were discouraging to say the least. The PDF generation was not easily manipulated and CSS support was minimal. After working with the package for several days and reading in the Drupal Print module documentation that "TCPDF seems to be more actively developed than dompdf, but it's support for CSS is considerably worse", I elected to try domPDF.

Generating Archival PDF files

The domPDF package proved to be much more effective and flexible in rendering the PDF. It had extensive support for CSS and allowed me to hide elements I did not want rendered, as well as manipulate elements that I did want rendered in the final PDF. In order to test the layout of the CSS however, I needed to find a way to prevent the PDF from rendering and have it come up as HTML to test the layout with the Firefox add-on "Firebug". Firebug allows developers to comb through the elements of a rendered HTML page to see what CSS is associated with each element. As Drupal has numerous CSS files associated with the layout that are difficult to keep track of, this was very important. After combing through the PHP code of both the domPDF generator and the Drupal Print module to ascertain where the final HTML was being generated, I found the rendered HTML string inside the "pdf/print_pdf.pages.inc" file of the Drupal Print module in the "Generate the PDF file using the dompdf library" section of code. I used the "exit()" command in PHP to exit and print the string "$html" that the module was generating, appearing as a new line of code - "exit($html);". This allowed the Print module to generate the HTML without sending it to domPDF for PDF generation so I could test the layout with Firebug as rendered HTML in the web browser.

The first thing I needed to do was replace the display jpeg file used at the website with a high resolution version for archival purposes. I achieved this by using the str_replace() function in PHP with the code "$html = str_replace('.preview', '', $html)" to replace the preview image automatically generated by the Drupal "Image" module upon uploading the image during page creation with the original 300dpi image. This string was later converted to " $html = str_replace('imagecache/display/', '', $html);" when I elected to move from the Image module to the "ImageField" module for greater control over the images. I also utilized the "ImageCache" module to have greater control over the resizing of images upon upload into thumbnail and preview states. At first I tried to replace the preview file with a full 600dpi tif or jpg file, but the domPDF generator did not like tif files and the server did not enjoy processing a 100MB 600dpi image file either. I therefore went with 300 dpi jpegs with no compression to create a decent quality archival format. Even these 300dpi images were on average 7-10MB and I had to modify the php.ini file's memory limit to be able to process the PDF. In order to accommodate several at once, I set the memory limit to 500MB and was fortunate that the hosting company allowed for such an excessive amount.

After removing the preview image and replacing it with a full sized 300dpi jpeg file for PDF generation, I needed make the jpegs fill the print area of the PDF. To place the 300dpi image file at the top left corner of the PDF, I set the position as absolute with a top and left of 0 in the print.css file of the Print module. I then also set the transcribed text with the same CSS settings and added a z-index parameter to put the transcribed text underneath the image. The image and text were being generated properly in the web browser with all other elements on the page being hidden with the display element set to none. I then commented the exit($html) out. The PDF generated as expected with the high resolution image on top and machine text below.

To make the image take up the whole page I adjusted the dpi settings on the dompdf_config.inc.php file several times before ascertaining that changing it from the default 96dpi to 150dpi on line 171 - define("DOMPDF_DPI", "150"); - gave me the desired result consistently (although I'm not entirely sure why 150dpi worked correctly with a 300dpi image).

Adding Page Margins

As I now needed to adjust the CSS so that the text appeared as closely as possible beneath the text on the image, I modified the print.css file. It quickly became apparent that this would not work with all images as the margins on all the documents were not the same, not to mention there were cross browser CSS rendering differences between Firefox, Safari, Chrome, Opera and Internet Explorer. I therefore needed to have adjustable margins for PDF generation. I set about creating text fields through the Drupal "CCK" module (which also handled images following the switch to the ImageFields module as well as the text). I allowed anyone generating PDF's to edit these fields and provided some basic directions on setting the margins. After extensive testing, I figured out that the whole node needed to be loaded in the print_pdf.pages.inc file to read the margin settings before the CSS could be added to the HTML output for PDF generation. The final PHP code to implement this appears below:

```
$node = node_load(arg(1));
$margin_top = $node->field_margin_top[0][value];
$margin_right = $node->field_margin_right[0][value];
$margin_bottom = $node->field_margin_bottom[0][value];
$margin_left = $node->field_margin_left[0][value];
$add_to_print_css = "<style type='"."text/css'"."> .field-field-
text{margin:".$margin_top." ".$margin_right." ".$margin_bottom." ".$margin_left.
"}</style>";
$html = str_replace('</head>', $add_to_print_css.'</head>', $html);
```

Some additional PHP code was also added to the print_pdf.pages.inc file later to strip out some the changes to the page when the Lightbox2 module was added to facilitate browsing of the images. Dublin Core metadata was also added to the PDF generation for descriptive archival purposes with the "page-break-before" CSS attribute set to "always" to prevent the metadata from overlapping the image and text upon PDF generation.

Transcription Interface

In addition to generating a PDF, as I had abandoned the use of expensive OCR software, I also needed to facilitate crowd-sourced web based transcriptions. I wanted to avoid doing this using the usual Drupal Edit node function to avoid confusion for everyday users and so I began looking at several Drupal modules that would behave like Wikipedia with a little edit button above the text that when clicked on would open the text for easy editing. After testing a few click to edit text modules created by the Drupal community, the "editablefields" module proved to be the closest to what I needed. However it did not seem to work with a WYSIWYG editor. I searched and found a Drupal issue thread in editablefields support - http://drupal.org/node/787598 - which added a patch to the editablefields module to provide WYSIWYG support. Although the primary module developer spoke of implementing this patch four to six months earlier, it hadn't yet materialized in the official module release and the code lines to implement the patch had changed on the latest development version (6.x-3.x-dev). I therefore used an older version of editablefields (6.x-2.0) and applied the patch. As it still did not work with the CKEditor WYSIWYG, I switched to the TinyMCE editor as implemented by the writer of the patch. After testing with TinyMCE version 4, it still did not work so I went down to TinyMCE version 3.39 for a final try where it worked some of the time. I figured out that the JavaScript that invoked the WYSIWYG in the web browser was only invoked on the page load and not upon opening the editablefield so rather than rewriting the JavaScript, I tried adding another instance of the WYSIWYG editor already opened with its CSS display set to none to avoid showing the extra editor interface to the end user. This finally worked as the WYSIWYG had already been opened once in the hidden field allowing another instance to open without any issues. The click to edit editablefield opened the text in a WYSIWYG interface and allowed users to edit and save it.

Now when the transcriptions are done through the web based interface, a user with appropriate permissions can proceed to generate a high quality PDF, save it to a local storage unit, flip a switch in the node edit area of the record and upload the archival PDF to attach to the record for users to download as needed. This switch also shuts off the ability to further edit the text by utilizing the Drupal Conditional Fields and Rules modules.

Next Steps

Despite my successes there are several issues I feel still need work

1. Caching/saving to server and automatically attaching to node
2. Combining multiple pages
3. Rotating horizontal images
4. Adding revisioning and workflow

Currently, the user generates a PDF and then saves it to his or her local computer to review. If the file seems good, the user then clicks a switch while editing the node telling it that the transcription is complete. A field then appears allowing the final PDF to be uploaded (utilizing the "Conditional Fields" module) and the text is transferred to a separate field that does not allow end user editing after the node has been saved (Using the Conditional Fields module and the "Rules" Module). However, I think it might be easier to automatically save the PDF and change the node to be uneditable upon PDF generation with the resulting PDF automatically attached to the node. This would require some sort of PDF review however, to make sure the margins are set appropriately and the transcribed text is roughly equivalent to the image text. This review would make this feature much more complicated and so I elected not to implement it, opting instead for the local review. Nevertheless, I may revisit this issue.

Another thing I would like to do is allow for the simultaneous editing of multi-page records. Currently, a 32 page document must be transcribed one page/record at a time. I do have a link to allow someone to view all 32 pages with their corresponding machine text, and also have linked them in the Lightbox module for better viewing provided the records have the appropriate metadata, however when I added the ability to edit them simultaneously, the WYSIWYG JavaScript went berserk causing numerous issues. The issue of multiple simultaneous WYSIWYG's was brought up in the issue thread of implementing a WYSIWYG to the editablefields module by the primary module developer as a potential problem (and it is more than a potential problem after my testing), which perhaps explains why the patch has not been implemented into the module. Nevertheless, I would like to find a way to combine a multi-page file into a single interface generating a single PDF at some point.

Another issue I have not yet dealt with is printing landscape (or vertically) oriented images. There are some browser dependent CSS tricks to rotate text which could work, however I would like a more elegant solution. I would prefer to have a switch to allow full resolution vertical images with corresponding text fields that print to PDF in landscape orientation. Unfortunately, this will require me to rewrite the Print module as this functionality is not allowed on a page by page basis, only a site by site basis (ie the entire site can print in either Portrait or Landscape, but no options exist for printing differently page by page). As this will require a fair amount of development, it was a project I had to save for another time.

The final thing that I need to implement is a revisioning and workflow process. As we are currently allowing anyone to become a user of the site, some unscrupulous users could potentially deface the site rather than assisting with it. We therefore need to be able to return to a previous version if this happens, or perhaps have a workflow implemented in which a content manager could approve changes to the record. I meant to implement this functionality on this version of the site, but simply ran out of time. I'm also nervous as to how editablefields will behave with revisioning and workflow and I sincerely hope there are no major conflicts.

Conclusions

Although I enjoyed working on this project from a development standpoint, I was shocked at the amount of time and effort I needed to put in to generating and adding the appropriate metadata to each record to make the site function correctly. The time involved in metadata creation was extensive despite the fact that I used bulk operations to add the majority of the metadata - which is feature I will certainly allow content creators to utilize as well - and never completed adding metadata to all the records. I expected the development as well as the image scanning and cleaning to take a considerable amount of time and planned accordingly. However, I did not expect the metadata creation to take nearly as long as it did. I therefore added as much metadata to the 240 records as I needed to make the site function using bulk operations on the records and left the remainder for other users to fill in as necessary...which really was the whole point of the project anyway!

I sincerely hope that this project facilitates the further creation of the Congregational Church of Austin Digital Archives and this process is useful for the archival community more generally. I have tried to demonstrate that creating a digital archive does not need to be expensive or difficult to implement. With an old scanner and a laptop I scanned 240 documents in roughly 24 hours. I then uploaded the images to a shared server space costing less than $10 a month in an open source CMS that barring excessive development time and with proper directions and some intermediate computer skills, could be set up relatively easily to facilitate crowd sourced transcriptions of old documents on an as needed basis by users of the archives. These documents when transcribed can then be converted into single files in PDF format for easy storage and preservation as well as use by anyone who might be interested. Accordingly, I believe that if a single individual can create and implement such a system in that person's limited spare time over the course of a semester, other public archives, libraries, and museums have few excuses left as to why they cannot or have not implemented such a system to promote the use of their materials to further the cause of human knowledge - which is, or should be, their primary mission.