Eliot Scott
Professor Gary Geisler
Independent Study – Spring 2011
10 May 2011

Creating a Digital Archive on a Small Budget

A member of the Congregational Church of Austin approached me to work on a web-based digital archive of their physical archives, which date back to the early 1900's. The archive had several requirements which we discussed. The first was that it could be easily edited and added to by general computer users at the church and possibly people visiting the church website. The second was that the work done on the archive could be easily portable to other platforms as computer technologies evolved. The third was that the archive could be easily maintained by church members once I had moved on. The final requirement was that the archive would be inexpensive to maintain. 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.

Project Beginnings

In the initial project plan, we planned to scan and perform OCR (Optical Character Recognition) on the archival documents. The church members would perform the majority of the transcriptions to clean the OCR'ed text in an as yet undefined OCR package once I had created a process that would be easy to emulate. 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 for accessibility along with the archival PDF.

I began by going to the church's archive room every Wednesday morning for three to five hours and scanning documents. Over the course of five weeks, I scanned 242 pages in 54 records as well a few dozen photos that have yet to be processed using a variety of methods to ascertain the easiest, most useful and most affordable processes for future members to add records to the archive.

In the first week I scanned the pages in 24bit color at 600dpi using Adobe Acrobat. I was hoping to utilize Acrobat to scan, perform OCR and create final archival PDF's so that the church would only have to purchase one piece of software. Unfortunately, it quickly became apparent that Acrobat's OCR capabilities gave results that were less than desirable. The majority of the OCR'ed text was incorrect, particularly on older documents and moreover, it was difficult to edit the numerous OCR mistakes Acrobat made in the final PDF, particularly for novice users. Given the problems with Acrobat's OCR capabilities, I decided to abandon the software as a feasible solution for novice computer users at the church to generate machine readable text.

For the next two weeks, I scanned pages at the same settings with ABBY Fine Reader. The OCR capabilities in ABBY were much better than Acrobat and it had a way to learn letter patterns in difficult to read documents, which were numerous in the pre-printer/typewriter/handwritten era of the archives. I decided to test various settings in ABBY as I went along scanning pages to find a solution. Some of the settings created major problems for archival quality images, particularly the "Split dual pages" and "Detect page orientation" options that chopped corners or edges of scanned pages incorrectly. These pages often needed to be rescanned entirely or at least generated less than optimal results that required additional

Photoshop work. Likewise, the "Train user pattern" setting that allowed ABBY to learn how to OCR many of the older pages more accurately was difficult to operate, incredibly time consuming to use and had to be adjusted for each individual document as the typewriters changed over the years. There were also numerous handwritten documents for which ABBY was mostly useless. After discussing these issues with the church committee member overseeing the project, we decided to abandon ABBY as a single all encompassing solution as well.

Ultimately, ABBY was relatively expensive for a multi-user project at $150 per license and it was unreasonable to expect project contributors to buy ABBY for their own machines or solely use the computer in the archive to transcribe and/or fix the OCR mistakes made by ABBY on every document, in addition to scanning them and creating a PDF. We did however agree that one copy of ABBY or Acrobat, or perhaps another more inexpensive OCR package such as PDF OCR, Free OCR or DocQ could be used as an initial pass to read the page and generate imperfect machine readable text that could be edited by users using another method.

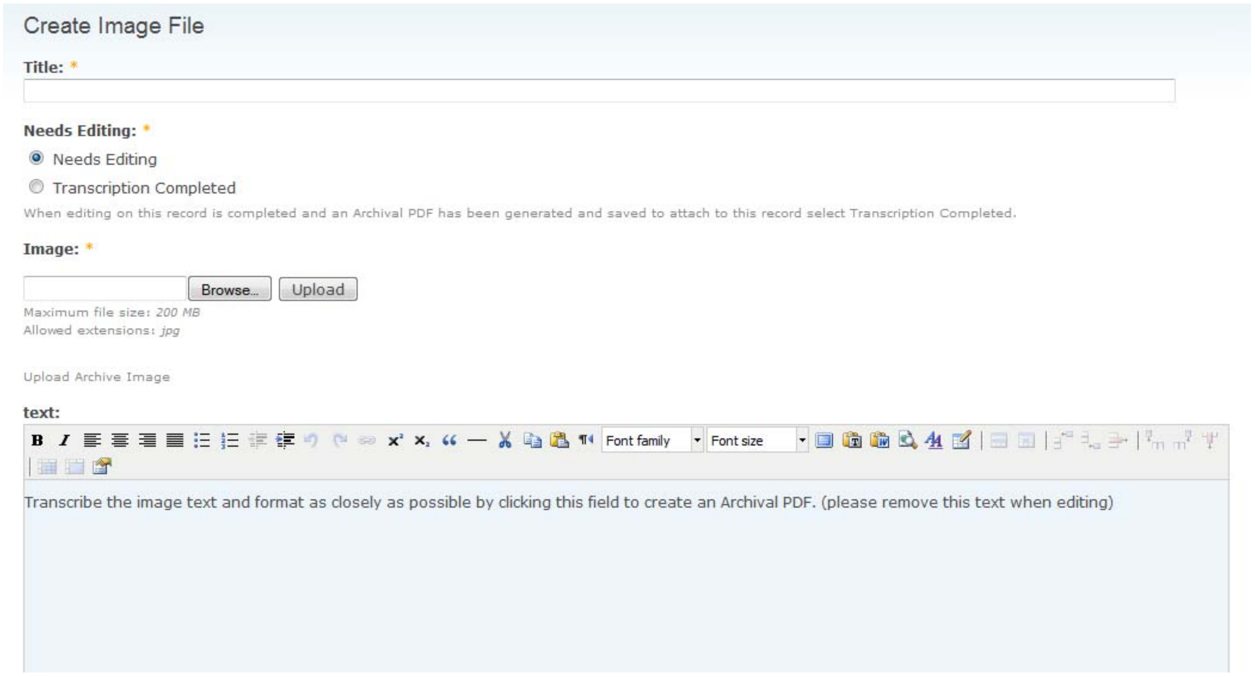
As it was impractical to expect church members to purchase pricey software individually or to spend hours doing OCR cleanup at the church archives, and I needed to create an archival PDF file that combined a high quality image with machine readable text into a single document for simple storage and access, 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 began looking into solutions to edit an OCR'ed PDF that church members or contributors could use to fix OCR mistakes. I evaluated a number of other PDF packages including FoxIt, NitroPDF, PDF OCR, and DocQ to see if any would allow users to easily edit PDF text after an initial pass by a single copy of ABBY. None did.

Evolving Project Plan

Despite my problems generating a workflow that could be utilized by members of the church to develop their archive, I elected to finish scanning enough documents to create a reasonable start to the digital archive. I was also asked to evaluate a donated scanner in addition to my own scanner that I had been using to scan the documents. I double scanned for a week using both the donated scanner and my own with two separate laptops to get more of the archives scanned. The donated scanner was missing its white backing but I elected to try it anyway. This was a serious mistake. The scanned files from the donated scanner had some black rubber in the background of pages that I digitally removed if possible or rescanned entirely in some cases, setting the project back more than the extra scanner moved it forward. I have recommended that the scanner not be used for future scanning unless its backing can be found.

After scanning the remainder of my documents (again in 24bit color at 600dpi) using the software built into the EPSON scanner driver I was using, I turned my attention toward developing the digital archive itself. The church had purchased shared server space at GreenGeeks for under $10 a month at my recommendation due its high scores as a Drupal friendly server from members of the Drupal community. I set up the server to redirect to the church's current web site and created a new sub-domain 'archives.congregationalchurchofaustin.org' where I installed the core of Drupal version 6.20. I also installed a number of Drupal community contributed modules that I thought I would need including most notably the Content Construction Kit (CCK), which allows the creation of various types of fields that can be associated with any individual Drupal Node and reused in a variety of ways that facilitate dynamic content editing and display. I also enabled the Image module so that an image could be uploaded with the record when a user created new content (See

Image A below).


Create Image File

Title: *

Needs Editing: *

[✓ Needs Editing]

Transcription Completed

When editing on this record is completed and an Archival PDF has been generated and saved to attach to this record select Transcription Completed.

Image: *

Maximum file size: *200 MB*
Allowed extensions: *jpg*

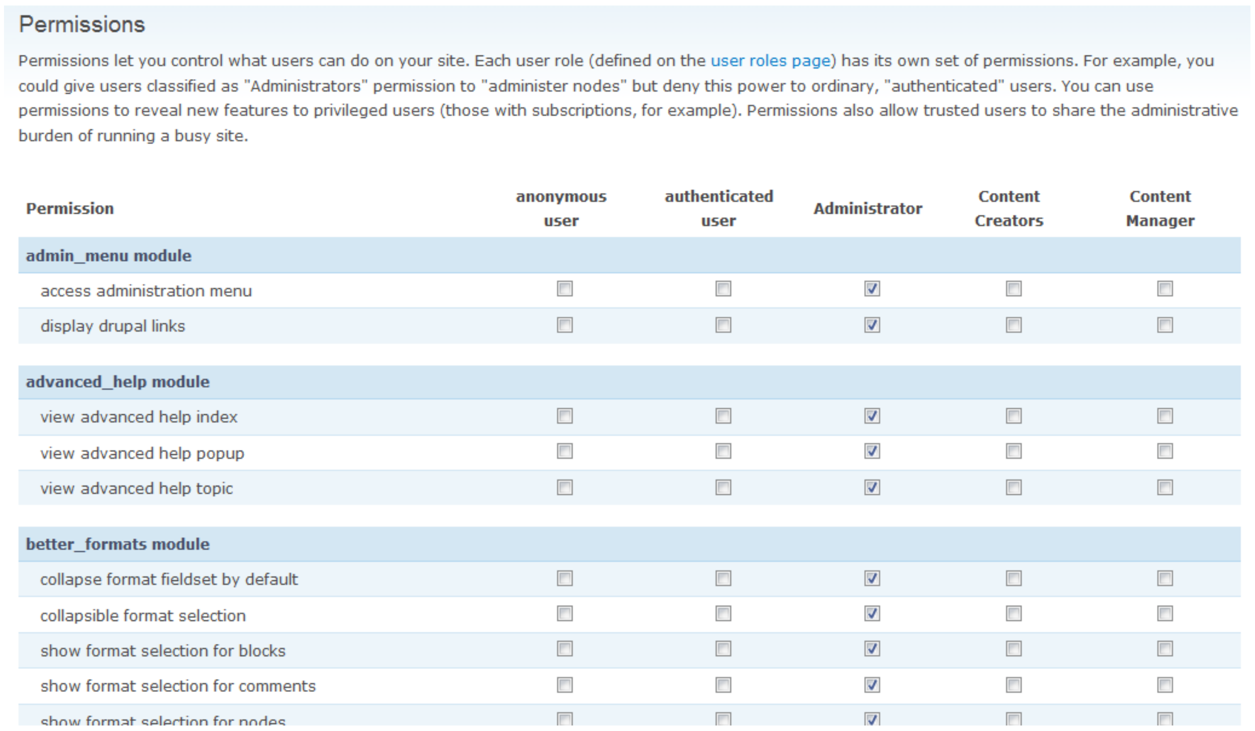
Upload Archive Image

text:

Transcribe the image text and format as closely as possible by clicking this field to create an Archival PDF. (please remove this text when editing)

Image A

I then created several user roles that I would need in addition to the default anonymous user and authenticated user, including a Content Creator, Content Manager and Administrator role. I then began setting up permissions to various installed modules, which expanded greatly as site creation continued (See Image B below).



Permissions

Permissions let you control what users can do on your site. Each user role (defined on the user roles page) has its own set of permissions. For example, you could give users classified as "Administrators" permission to "administer nodes" but deny this power to ordinary, "authenticated" users. You can use permissions to reveal new features to privileged users (those with subscriptions, for example). Permissions also allow trusted users to share the administrative burden of running a busy site.

Permission	anonymous user	authenticated user	Administrator	Content Creators	Content Manager
admin_menu module					
access administration menu	☐	☐	☑	☐	☐
display drupal links	☐	☐	☑	☐	☐
advanced_help module					
view advanced help index	☐	☐	☑	☐	☐
view advanced help popup	☐	☐	☑	☐	☐
view advanced help topic	☐	☐	☑	☐	☐
better_formats module					
collapse format fieldset by default	☐	☐	☑	☐	☐
collapsible format selection	☐	☐	☑	☐	☐
show format selection for blocks	☐	☐	☑	☐	☐
show format selection for comments	☐	☐	☑	☐	☐
show format selection for nodes	☐	☐	☑	☐	☐

Image B

As my workflow problem in transcribing the documents to machine readable text for archival PDF generation had still not been solved, I turned my attention toward several 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). I elected to try the open source solutions, utilizing the Drupal interface to not only display the archives, but to edit them and generate a final archival PDF as well. Utilizing open source products to both display and edit the archives would also significantly reduce costs and so it seemed as though this would be the ideal solution. However, I now needed to discover if I could make it work.

Unfortunately the Print module web page (http://drupal.org/project/print) stated that wkhtmltopdf, although the best of the three open source PDF packages it supported, 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 as it was more recently developed. 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 web 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 the program and print the string "$html" that the Print module was generating, appearing as a new line of code - "exit($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.

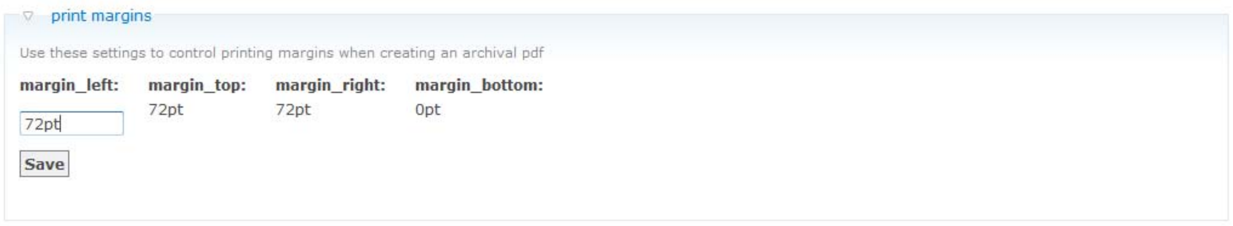
The first thing I needed to do was replace the display jpeg file used at the website with the original high resolution version for archival purposes. I achieved this by using the str_replace() function in PHP with the code "$html = str_replace('.preview', '', $html)" to replace the preview image automatically generated by the Image module upon uploading the image during Drupal Node 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 Drupal Image module to the ImageField module for greater control over the images. At first I tried to replace the preview file with a full 600dpi tif file, but the domPDF generator did not like tif files and the server did not enjoy processing a 100MB 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 it 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 put 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 web 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 PDF 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 it was 150 and not 300. This process had to be repeated numerous times as new elements were added to the project so that only the desired elements were rendered during the PDF generation.

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 each of the documents were not the same, not to mention there were some 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. 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 Drupal 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 with an image of the print margins interface following:

```
$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);
```


▽ print margins

Use these settings to control printing margins when creating an archival pdf

margin_left:	margin_top:	margin_right:	margin_bottom:
72pt	72pt	72pt	0pt

Image C

Some additional PHP code was also added to the print_pdf.pages.inc file later to strip out some the changes to the web page when the Lightbox2 module was added to facilitate browsing of the images. The Dublin Core metadata described below was later 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

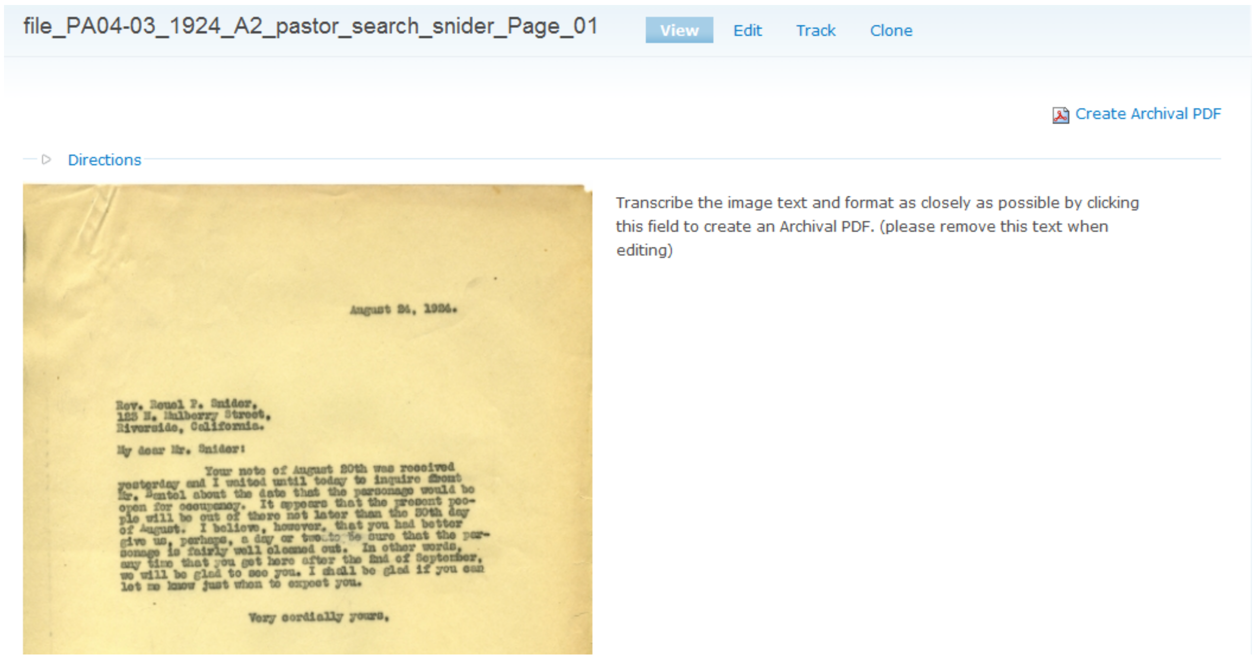
In addition to generating a PDF, as I had abandoned the use of expensive OCR software to edit the final text of the document, I also needed to facilitate crowd-sourced web based transcriptions. I wanted to avoid doing this using the usual Drupal Edit Node interface as it creates usability problems for everyday users 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. It utilized a CCK field and displayed it as click to edit text which when clicked on produced a text area that was editable. However it did not seem to work with a WYSIWYG editor, which was essential to control the machine text to match the image text. 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 WYSIWYG CKEditor, 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 module in the web browser was only invoked on the web page load and not upon opening the editable CCK field 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.

With this feature, Content Creators and Managers at the church would have the option to OCR the text in any package and paste the results when they created a new record, or they could forgo the OCR altogether and rely entirely on crowd sourced transcriptions to generate machine readable text on an as needed basis within the archive.

Creating the User Interface

In order to create the interface, facilitating browsing and searching of the records as well as editing, I began using the Drupal Panels module. Although Panels seemed like a good choice for its ability to display components in a panel or frame-like interface, it quickly became apparent that managing the records would be difficult for novice users. I therefore abandoned the Panels module and modified the content type automatically generated by the Image module. The

Image content type allowed end users to upload an image to the Drupal Node where it automatically generated a preview and thumbnail version of the image for display purposes. This seemed perfect as the original image was a 300dpi uncompressed jpeg used only to generate the archival PDF and I needed a preview image for the main record web page, as well as a thumbnail for an as yet un-built gallery. I added a CCK text field for the transcription text below the image display and modified the CSS so that the editable CCK text field appeared next to the display image (See Image D and E below).



Transcribe the image text and format as closely as possible by clicking this field to create an Archival PDF. (please remove this text when editing)

August 24, 1924.

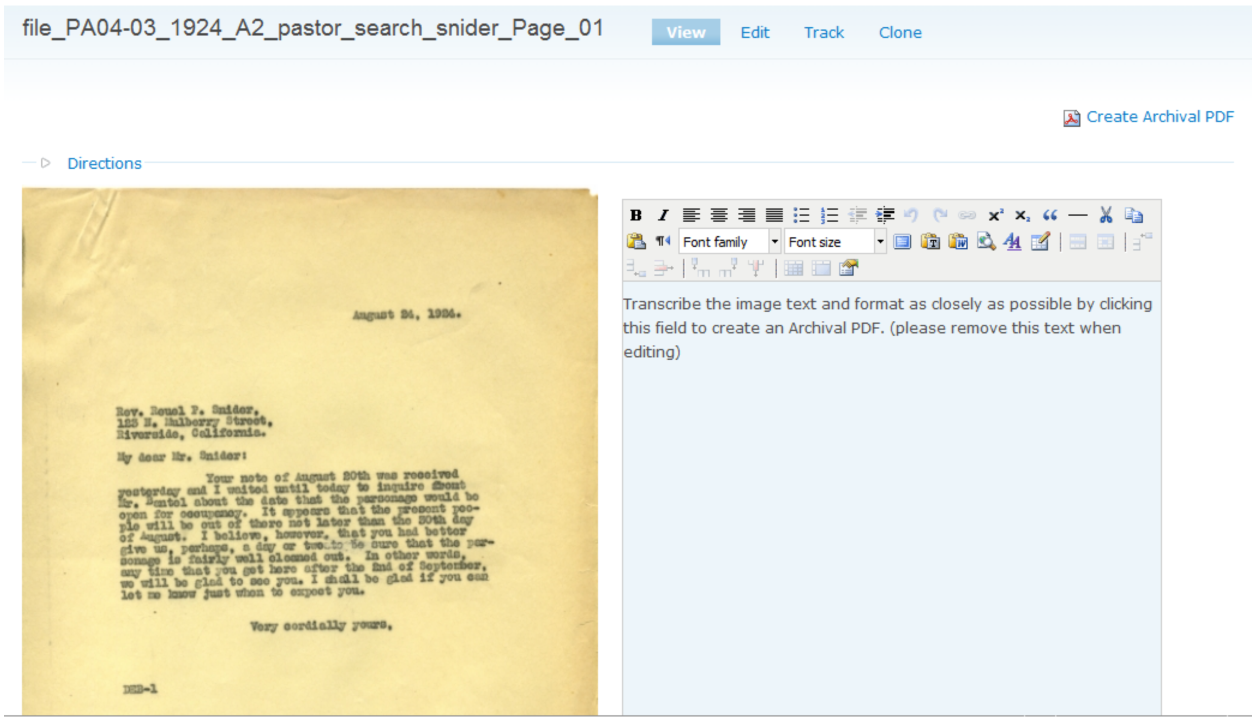
Rev. Reuel P. Snider,
123 N. Mulberry Street,
Riverside, California.

My dear Mr. Snider:

Your note of August 20th was received yesterday and I waited until today to inquire about Mr. Bentel about the date that the personage would be open for occupancy. It appears that the present people will be out of there not later than the 30th day of August. I believe, however, that you had better give us, perhaps, a day or two to be sure that the personage is fairly well cleaned out. In other words, any time that you get here after the 2nd of September, we will be glad to see you. I shall be glad if you can let me know just when to expect you.

Very cordially yours,

Image D



file_PA04-03_1924_A2_pastor_search_snider_Page_01

August 24, 1924.

Rev. Reuel P. Snider,
123 N. Mulberry Street,
Riverside, California.

My dear Mr. Snider:

Your note of August 20th was received
yesterday and I waited until today to inquire about
Mr. Bentel about the date that the personage would be
open for occupancy. It appears that the present peo-
ple will be out of there not later than the 30th day
of August. I believe, however, that you had better
give us, perhaps, a day or two to be sure that the par-
sonage is fairly well cleaned out. In other words,
any time that you get here after the End of September,
we will be glad to see you. I shall be glad if you can
let me know just when to expect you.

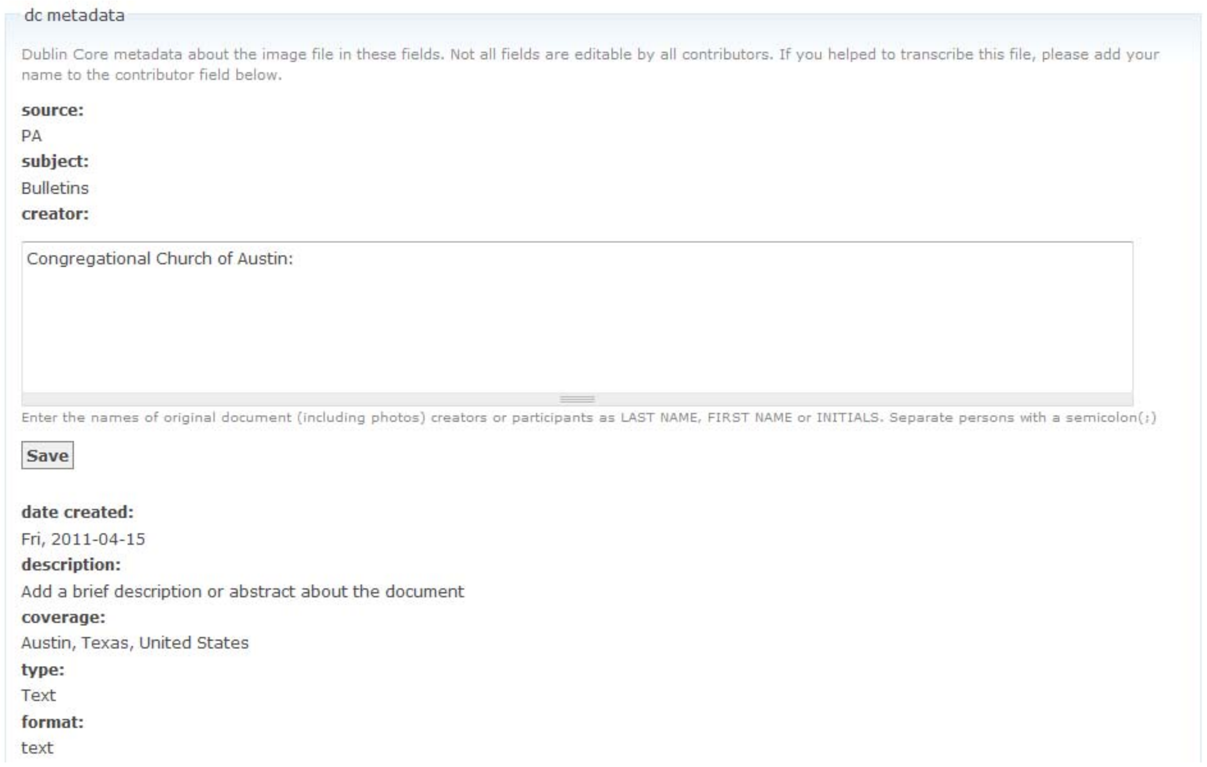
Very cordially yours,

DEB-1

Transcribe the image text and format as closely as possible by clicking this field to create an Archival PDF. (please remove this text when editing)

Image E

I then added Dublin Core metadata using the Dublin Core to CCK module, which generated DC metadata CCK fields that I then made editable using the Editablefields module as well (See Image F below).



dc metadata

Dublin Core metadata about the image file in these fields. Not all fields are editable by all contributors. If you helped to transcribe this file, please add your name to the contributor field below.

source:
PA
subject:
Bulletins
creator:

Congregational Church of Austin:

Enter the names of original document (including photos) creators or participants as LAST NAME, FIRST NAME or INITIALS. Separate persons with a semicolon(;)

Save

date created:
Fri, 2011-04-15
description:
Add a brief description or abstract about the document
coverage:
Austin, Texas, United States
type:
Text
format:
text

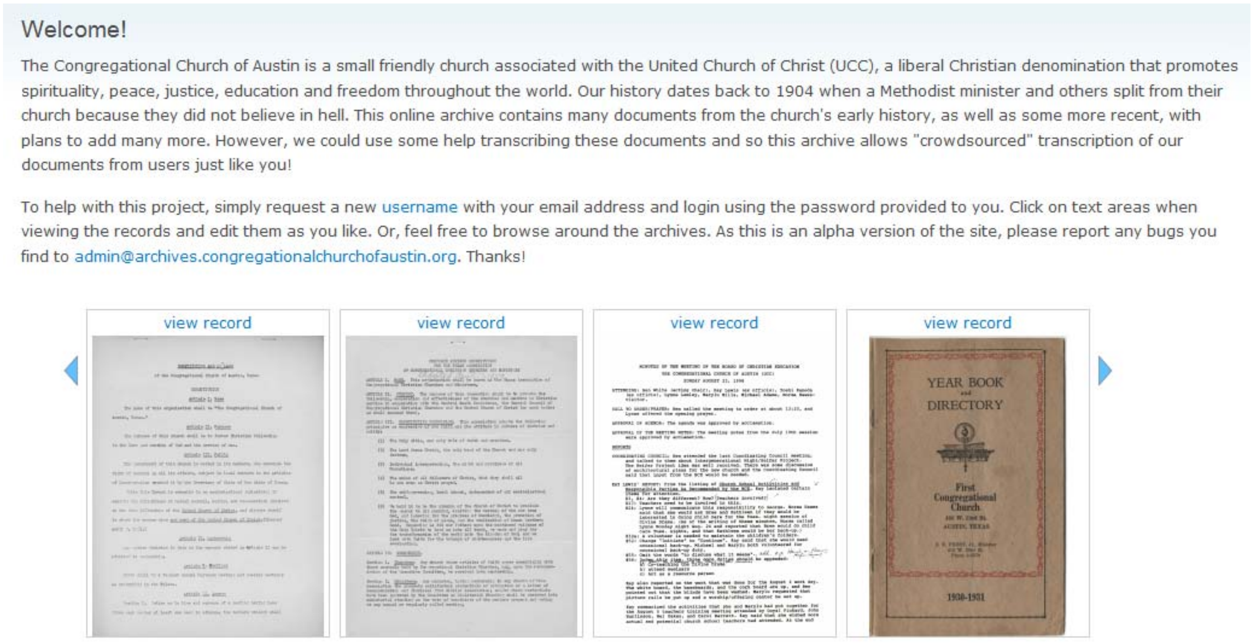
Image F

After cleaning and rotating each image in Photoshop for display purposes, I batch processed the 600dpi tif images to 300dpi uncompressed jpegs and uploaded the images via FTP. I then performed a mass Drupal Node generation of the images using the Views Bulk Operations (VBO) module.

Unfortunately, it later became apparent that the Image module was somewhat inflexible - images could not be reused in creative and dynamic ways using the Views module without becoming fields and the entire system would not be easily upgradable later as the new Drupal 7 core system had integrated CCK and converted all content to fields. I therefore converted all the Image content types to a generic content type with an image CCK field using the Image Fields module. I then migrated all the corresponding DC and text fields to the new generic content type. Although this required recreation of some elements, the CCK Image Field solution provided greater current and future flexibility and I re-uploaded the images using the new method. Using the Image Cache module, I also generated new display and thumbnail images, and this provided greater control over sizes and server cache settings.

Creating a Display Interface

After finalizing the basic interface on each record's main web page, I turned my attention toward creating a gallery. There are numerous Drupal modules that provide this functionality and none appeared to be the most utilized, and therefore tested, module. After evaluating several popular gallery interfaces, I selected Lightbox2 and jCarousel to work in conjunction with one another to generate scrollable galleries that also allowed for zooming into and out of each image in the archive (See Image G and H below).

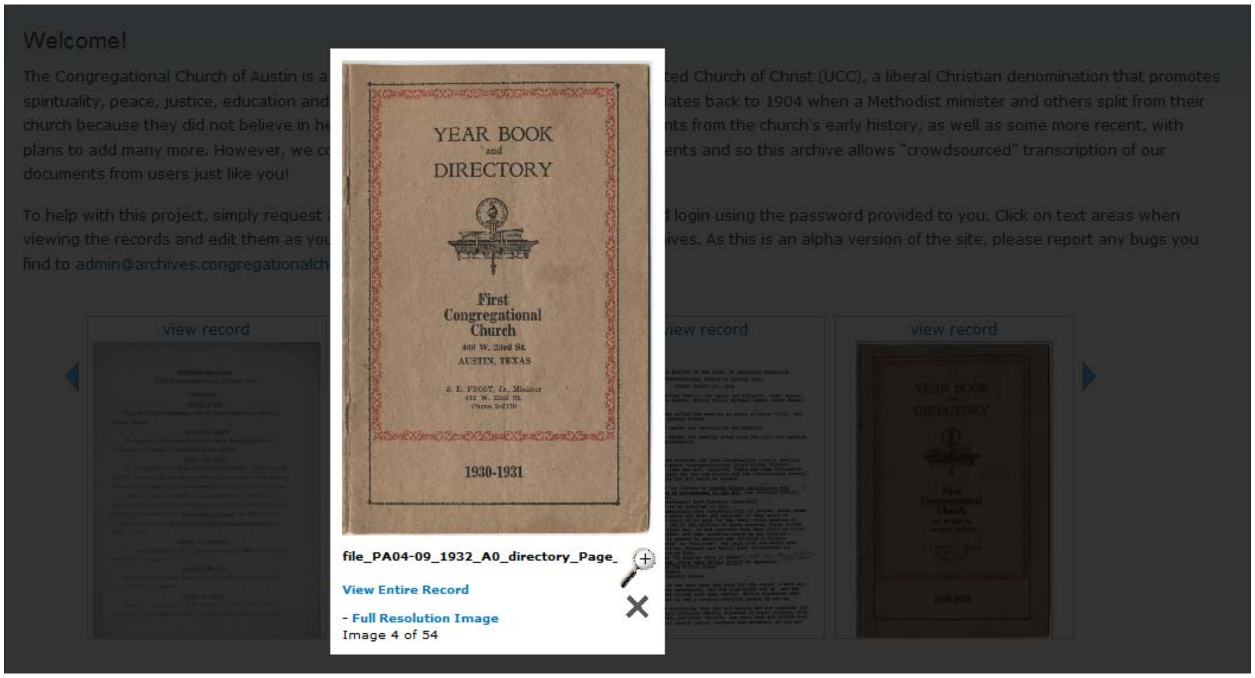


Welcome!

The Congregational Church of Austin is a small friendly church associated with the United Church of Christ (UCC), a liberal Christian denomination that promotes spirituality, peace, justice, education and freedom throughout the world. Our history dates back to 1904 when a Methodist minister and others split from their church because they did not believe in hell. This online archive contains many documents from the church's early history, as well as some more recent, with plans to add many more. However, we could use some help transcribing these documents and so this archive allows "crowdsourced" transcription of our documents from users just like you!

To help with this project, simply request a new username with your email address and login using the password provided to you. Click on text areas when viewing the records and edit them as you like. Or, feel free to browse around the archives. As this is an alpha version of the site, please report any bugs you find to admin@archives.congregationalchurchofaustin.org. Thanks!

Image G



Welcome!

The Congregational Church of Austin is a ted Church of Christ (UCC), a liberal Christian denomination that promotes
spirituality, peace, justice, education and ates back to 1904 when a Methodist minister and others split from their
church because they did not believe in he nts from the church's early history, as well as some more recent, with
plans to add many more. However, we c ents and so this archive allows "crowdsourced" transcription of our
documents from users just like you!

To help with this project, simply request d login using the password provided to you. Click on text areas when
viewing the records and edit them as yo ives. As this is an alpha version of the site, please report any bugs you
find to admin@archives.congregationalch

YEAR BOOK and DIRECTORY

First Congregational Church
408 W. 23rd St.
AUSTIN, TEXAS

S. E. FROST, Jr., Minister
412 W. 23rd St.

1930-1931

file_PA04-09_1932_A0_directory_Page_

View Entire Record

- Full Resolution Image
Image 4 of 54

Image H

I also needed to find a way to associate multiple page records with one another. As I had already created a DC relation field within the metadata, I decided to leverage that metadata. Using the Views module to take advantage of the relationship created through the DC relation field, I was able to show all the files related to a particular record on that record's web page in a jCarousel Lightbox2 display. I was also able to provide a link to a View that displayed all the associated records on a single web page (See Image I below).

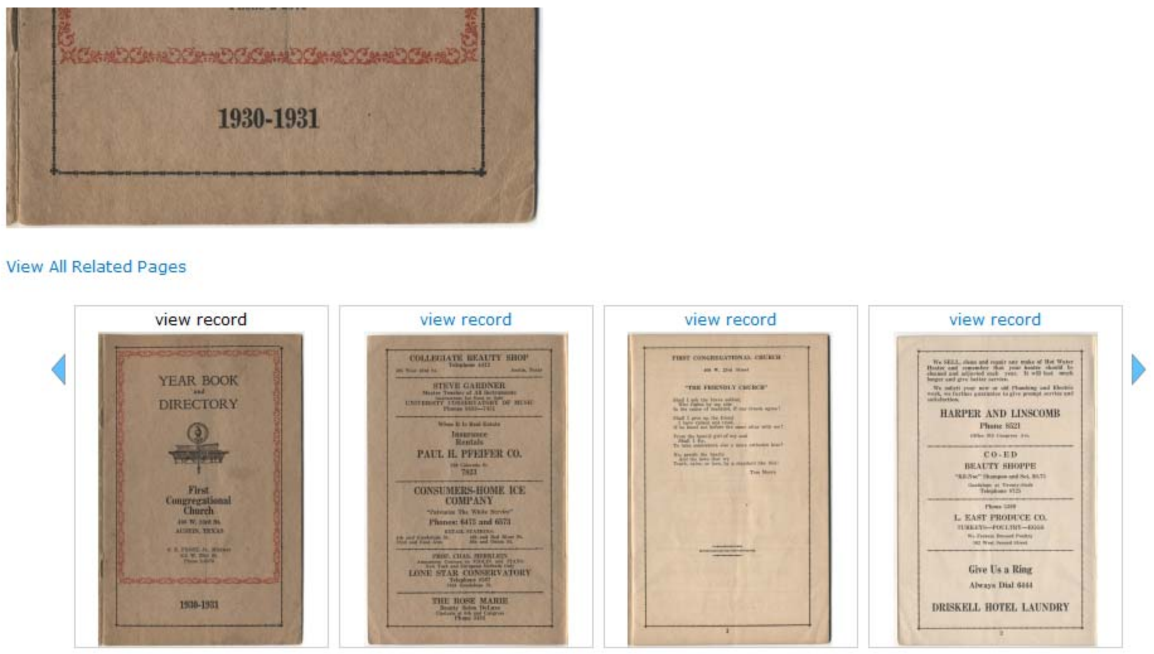
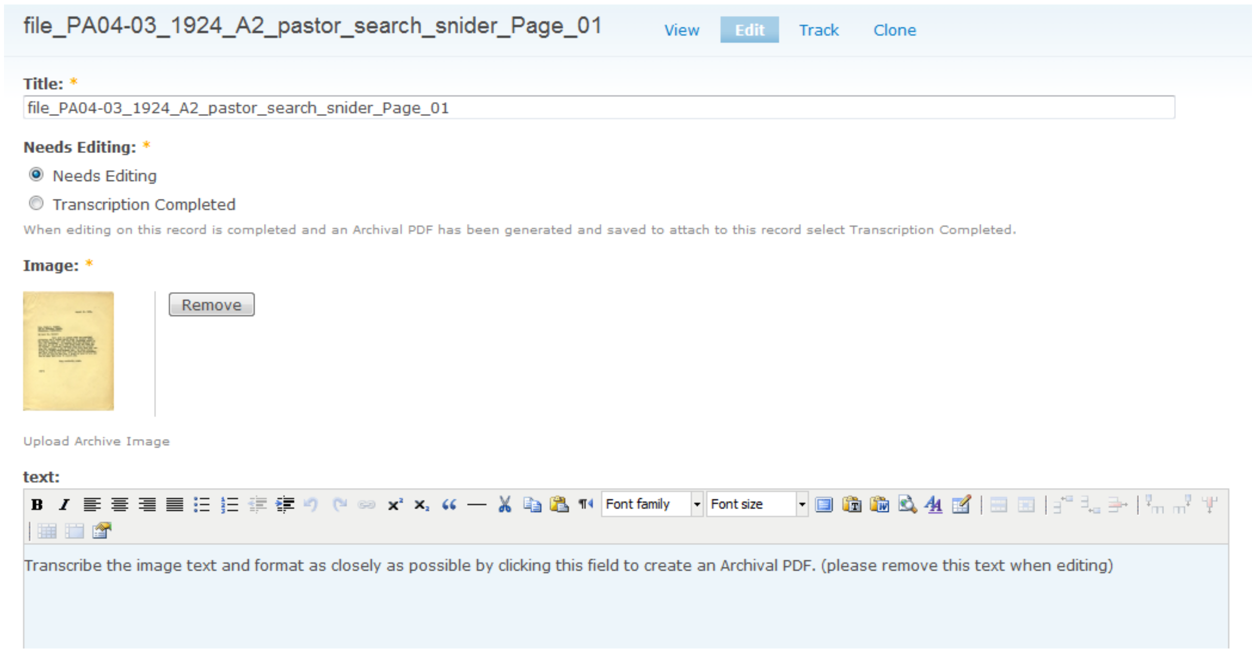


Image I

Finally, I needed to stop the record from being edited once the transcription was done and allow the archival PDF to be uploaded and associated with the record. Using the Conditional Fields and Rules modules, I created rules and conditions that allow a user with appropriate permissions to generate a high quality PDF, save it to a local storage unit, flip a switch in the Drupal Edit Node 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 (See Image J and K below).


file_PA04-03_1924_A2_pastor_search_snider_Page_01

Title: *

file_PA04-03_1924_A2_pastor_search_snider_Page_01

Needs Editing: *

[✓ Needs Editing]

Transcription Completed

When editing on this record is completed and an Archival PDF has been generated and saved to attach to this record select Transcription Completed.

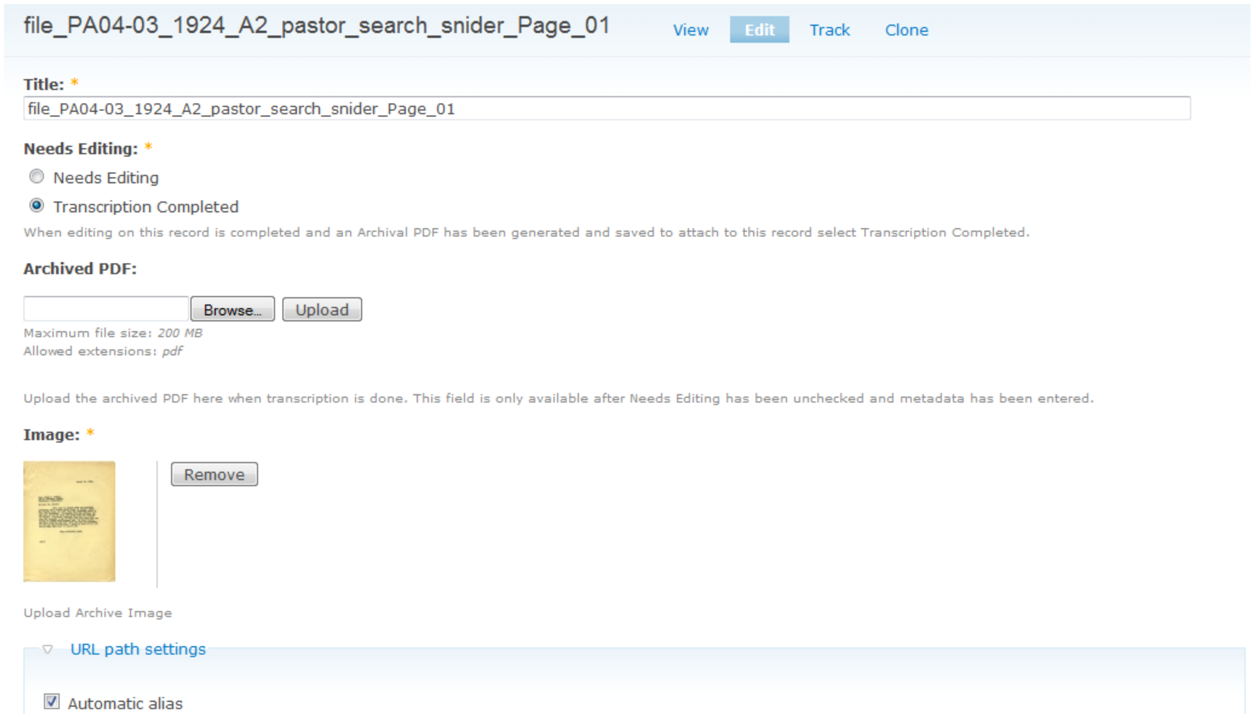
Image: *

Upload Archive Image

text:

Transcribe the image text and format as closely as possible by clicking this field to create an Archival PDF. (please remove this text when editing)

Image J


file_PA04-03_1924_A2_pastor_search_snider_Page_01

Title: *

file_PA04-03_1924_A2_pastor_search_snider_Page_01

Needs Editing: *

Needs Editing

[✓ Transcription Completed]

When editing on this record is completed and an Archival PDF has been generated and saved to attach to this record select Transcription Completed.

Archived PDF:

Maximum file size: *200 MB*
Allowed extensions: *pdf*

Upload the archived PDF here when transcription is done. This field is only available after Needs Editing has been unchecked and metadata has been entered.

Image: *

Upload Archive Image

URL path settings

[✓ Automatic alias]

Image K

Next Steps

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

1. Saving the PDF to the server and attaching to a record automatically

2. Combining multi-page records on a single web page for editing
3. Making records automatically related to one another if file names match
4. Rotating horizontal images
5. Minor adjustments - create final fields for all editable elements, make user tagging easier, adjust caching and create an auto indexing job to facilitate searches
6. Adding revisioning and/or workflow

These changes when implemented will make the archive easier to use and more functional.

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 Drupal Node telling it that the transcription is complete. A field then appears allowing the final PDF to be uploaded using the Conditional Fields module and the text is transferred to a separate CCK text field that does not allow end user editing after the Drupal Node has been saved using the Conditional Fields and the Rules modules. However, I think it might be easier to automatically save the PDF and change the Drupal Node to be un-editable automatically upon PDF generation with the resulting PDF automatically attached to the Drupal 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 in size to the image text before finalizing the record. 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 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 Views module using a jCarousel with Lightbox display for better viewing provided the records have the appropriate metadata in the dc relation field. However when I added the ability to edit the record's pages on the same web page in the Views module, 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. This perhaps explains why the WYSIWYG patch has not been implemented into the Editablefields module. Regardless, I would like to find a way to combine a multi-page record into a single web page interface generating a single multi-page PDF at some point.

Further, I would like to find a way to automatically associate records and auto generate the DC relation field based solely on the title of the record. Thus a record called "file_x_Page_01" would be automatically associated with "file_x_Page_02" and "file_x_Page_03". This would cut down significantly on the time it takes for users to associate records with one another in the dc relation field, which is essential in generating the View that shows all the related records in the jCarousel with Lightbox display format.

Another issue I have not yet dealt with is printing landscape (or vertically) oriented images. There are some browser dependent CSS tricks to rotate elements 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 significant PHP development on the Print module as this functionality is not allowed on a web page by web page basis, only a web site by web site basis (ie the entire site can be either Portrait or Landscape printed, but no options exist for printing differently web page by web page). As this will require a fair amount of development, it was a project I had to save for another time.

Some minor annoyances I would like to address include adding a finalized field for the DC metadata after the content manager flips the switch on the record noting that the transcription is complete disallowing further editing to the transcript. This switch should disallow further editing to the metadata as well. I would also like to work with the user tagging feature to make it more user friendly and more functional. Additionally, I would like to optimize the caching processes on the site and create a nightly auto refresh of the content index to enhance and automate the search functions as well as refresh the image and web page caches.

The final thing that I need to implement is a revisioning and workflow process. As the site currently allows anyone to become a user of the site, some unscrupulous users could potentially deface the archive 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 any changes. I meant to implement this on this version of the site, but simply ran out of time. I'm also nervous as to how Editablefields will behave with the Revisioning and Workflow modules 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 thinking about, generating and adding the appropriate metadata to each record to make the site function correctly. I expected the CMS 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 Views Bulk Operations on the records and left the remainder of the metadata 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 a free of charge open source CMS. Barring excessive development time and with proper directions and some intermediate computer skills, this CMS could be set up relatively easily to facilitate crowd sourced transcriptions of old documents on an as needed basis by users of any archive. These documents when transcribed can then be converted into single files in PDF format for easy storage and preservation as well as general 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.