Transforming External Legacy Metadata for ArchivesSpace

A labor of love

Bria Parker, Discovery and Metadata Services

When the AFL-CIO collection was acquired in 2013, University of Maryland Libraries received a database export of all the metadata associated with the materials. While we already had a proven process for transforming our local metadata from The Beast into ArchivesSpace, this process would not apply. In this presentation I will outline the challenges of transforming the AFL-CIO metadata and describe the transformation workflows. Specifically, I will show the tool (Stead) that was modified and used for the majority of the transformation work, as well as review the other steps, tips, and tools I utilized to format the metadata for optimal import into ArchivesSpace.
First, a little bit about ArchivesSpace: ArchivesSpace is an open source content management system for archival metadata and materials that manages the entire lifecycle of archival collections, similar to how an Integrated Library System (ILS) manages library resources.
SCUA and SCPA have spent the past few years migrating our archival metadata into ArchivesSpace from an Access database lovingly and aptly named The Beast. We developed a process for cleaning up accessions in spreadsheets to be loaded via an importer within the system. We developed a workflow and tools for getting finding aid metadata from The Beast into ArchivesSpace. But this presentation isn’t about that. Some of those processes were not going to work for our next challenge, since we didn’t even have EAD XML to start with for AFL-CIO. We had to get to that point before we could reuse any processes and tools.
Here's a quick view of the workflow we used for the finding aid metadata in the Beast - first there was a Java converter that got the metadata from the Beast into EAD XML (it's a black box, though, and no one remains that really knows what's up with it. Then Josh Westgard created a python script to do some batch processing on the EAD XML to correct some common problems that were going to cause import errors. Next, a date normalization process, followed by a stylesheet to add top container barcodes, and a schematron to do error checking. Then it was on to import, and some post-import processing.
But, for the AFL-CIO metadata, those first two steps were not applicable.
As I mentioned in the beginning, when we acquired the AFL-CIO collection, we received a database export of all the metadata associated with the materials. This metadata had been stored in an Eloquent database, Eloquent being the name of an actual database system, not necessarily the adjective that would be used to describe the database...

And while our finding aid data was stored in a database, it was not in the same format, and could not use the same conversion tools to get it into ArchivesSpace, so we needed to find a different approach.
So here's a snippet view of what received: spreadsheet dumps of the data - one spreadsheet for each table in the database. We had one big spreadsheet of the collection, series, and subseries metadata, another with box, folder, and item level metadata, as well as others for subjects and names.
We began exploring our options - how would we get this in? AFL-CIO is a big, important collection, and we had very little discoverability for it. There was a collection guide online that gave brief descriptions of the collections, but we really needed to make these more broadly discoverable for researchers.
Challenges

- Spreadsheets not in the same format as those from The Beast
  - (plus, we had a converter we could use to get from The Beast to EAD)
- Database exports needed to be separated out from the master tables by collection and merged.
  - This was already done by SCUA
- No .csv import of finding aid data function native in ArchivesSpace
  - Plugins available, but that required developer time that was not available
We had previously heard about and played around with a tool called Steady, created by Jason Ronallo at NCSU. This tool would take a csv and convert it to EAD XML. Which we *could* import into ArchivesSpace. But! Again, it didn’t capture as much detail as we liked (if we had the metadata, I wanted to use it, rather than have a lot of manual work post import!), PLUS, you had to convert one csv at a time.
I liked Steady a lot, though, so I started digging. I wondered if I could alter Steady to accept and map more columns. THEN, in looking at the github repository for it, I saw that Steady was actually a wrapper that worked with a Ruby gem he had created called Stead (which stood for Spreadsheets to Encoded Archival Description). Stead is what was doing the heavy lifting of the converting. Steady was the mechanism for passing files back and forth between the gem and the user.

So, I began looking into Stead more. since it required a pretty specific .csv template, I had my GA at the time, Sydney Vaile, work on mapping collection inventories that Jen Eidsen’s GA had created from the Eloquent data into the Stead csv template
In the meantime I needed to roll up my sleeves and get comfy with Ruby.

I copied the github repository to my machine and began exploring what changes could be made.

The first thing I did was add additional arguments to set collection-level metadata (and by arguments I'm referring to the actual input provided in the command line when running the program).

I added arguments for including resource id, extent, scope and content notes metadata at the collection level, while also parsing out dates for normalization.
Here is a snippet of the file that controls the input arguments...

```ruby
#!/usr/bin/env ruby

$LOAD_PATH.unshift File.join(File.dirname(__FILE__), '..', 'lib')
require 'pp'
require 'stead'
require 'trollop'

opts = Trollop::options do
  banner "-e EDS"
  This script takes a .csv file with a name in the format <eadid>_container_list.csv
  and creates a stub EAD XML document.

  Usage:
  csv2ead --csv /path/to/<eadid>_container_list.csv [options]

where options are:
  -e EDS
  -csv, "A CSV file", :required => true, :type => String
  -baseurl, "Base URL for adding on the eadid", :type => String
  -url, "Full URL for this collection guide", :type => String
  -template, "Specify using a different EAD XML template", :type => String
  -ncsu, "Use NCSU specific template"
  -extra, "Full path to a Sted::Extra file to add in other data", :type => String
  -output, "Save the file by specifying the filename", :type => String
  -pretty, "If --output is specified this will pretty indent the container list."
  -tod, "Output full EAD to terminal"
  -idcontainers, "Add id attributes to containers to show parent child relationships among containers in same com"
  -unittitle, "Add a unittitle to ead/eadheader/archdesc/did/unittitle", :type => String
  -unitid, "Add a unitid to ead/eadheader/archdesc/did/unitid", :type => String
  -extent, "Add an extent to ead/eadheader/archdesc/did/physdesc/extent", :type => String
  -unidate, "Add a unidate to ead/eadheader/archdesc/did/unidate", :type => String
  -begindate, "Add a start date to ead/eadheader/archdesc/did/unidate/@normal", :type => String
  -enddate, "Add an end date to ead/eadheader/archdesc/did/unidate/@normal", :type => String
  -scope, "Add a scopecontent note to ead/archdesc/scopecontent/p", :type => String
```
def collectiondate_node
    @ead.xpath('//xmlns:archdesc/xmlns:did/xmlns:unittitle').first
end

def add_normal_date
    if @begindate
        collectiondate_node['normal'] = @begindate + '/' + @enddate
    end
end

def scope_node
    @ead.xpath('//xmlns:archdesc/xmlns:scopecontent/xmlns:ps').first
end

def add_scope
    if @scope
        scope_node.content = @scope
    end
end

def to_ead
    @ead = template.dup
    add_eadid
    add_eadid_url
    add_archdesc_unittitle
    add_title Proper
    add_archdesc_unittid
    add_eadid
    add_archdesc_extent
    add_normal_date
    add_archdesc_unidades
    add_scope
    @dsc = @ead.xpath('//xmlns:archdesc/xmlns:dsc').collect
    if series?
        add_series
        add_subseries if subseries?
    end
    @component_parts.each do |cp|
        c = node(file_component_part_name(cp))
        c['level'] = 'file'
    end

And here is a snippet showing these arguments (or input) getting mapped to an element within the EAD, using XPath.

(Xpath is a method for navigating XML elements and attributes - so it's like a map with directions to say "take this and follow the XML tree to *here* and put the data there).
In addition to accommodating additional collection metadata at the command line, I was able to adjust the script to handle the additional metadata we had at the series level, such as scope and contents notes, series extent, related subjects and names, as well as date parsing for normalization, so the function for creating a series went from taking 25 lines of code and fitting on one slide....
To taking nearly 70 lines of code and barely fitting on two slides.
I made the script more precise and accommodated more metadata in the transformation, which meant less manual work for curators and GAs down the road. Everyone wins!
So, I had conquered series level metadata, and container lists for archival objects within those series, but was still fumbling with the subseries.
Enter Caitlin. In the summer of 2017 I had a field study student, Caitlin Rizzo who was interested in learning more about metadata transformation work, specifically through code. We developed a project for her to work with me to further modify the Stead code to accept and properly map subseries metadata. She was ultimately successful and was able to incorporate the additional subseries level metadata we had that was not in the original code - scope and contents notes, extents, subjects, people and organizations!!
Now, we had a tool for transforming nearly all of the AFL-CIO metadata.
And I could replace the first two steps of the previous process with just the Stead converter. Now the transformation could begin!
Now for some real talk. Not everything worked the way I wanted it to. I converted the bulk of the collection level metadata spreadsheet into a tab delimited text that incorporated the argument tags, with the thought of developing a bash scrip that I could run at the command line that would go line by line through the file and convert them en masse. I spent a lot of time trying to make that work. It did not. I don't know if it was the nature of the ruby gem, or limits to my ability (PROBABLY), but I abandoned that work in favor of it just getting done. So I just copied and pasted each line into the command line. I tell you this to illustrate that sometimes something just doesn't work. And that it's okay.
So, here is what it looked like in the command line when I'd run it
Csv2ead was the name of the function within the gem - the -c argument gives the
filename to convert, ….etc.

When it finished processing, the output file was sitting in the same directory as the
source csv.
Normalization makes the dates machine readable, which leads to better date searching/faceting for our users.

So, now I had EAD XML, but it still wasn't quite ready to be imported to ArchivesSpace. At this point, I could actually re-use and repurpose processes and tools used for The Beast data. I re-used the process for normalizing dates (thanks Bentley Library for documenting their process and sharing their code!). Why are normalized dates important? Normalization makes the dates machine readable, which leads to better date searching/faceting for our users.

This date normalization process involved a python script to fix simple date format cases, then a python script to extract dates the first script couldn't fix, then open refine work to parse and normalize the dates, and a python script to load those normalized dates back into the XML.
I was also able to tweak XSLT I had developed for creating ArchivesSpace-compliant top containers out of the container lists. The top container part was pretty crucial, as it gives us better control over the boxes in a collection - each box gets a barcode (and since our boxes aren't barcoded yet, I generated dummy barcodes) and that box is now Box 1 in a certain collection, but also is this exact box that lives on this floor and is not the same as the Box 1 in this other collection, and operates much like how we manage books and items in Aleph.

Once all that was complete, I also had a schematron I could run to check that everything would pass ArchivesSpace import.
Now for import! At this point, with this process, I have imported over 85 resources/finding aids into ArchivesSpace production, accounting for approximately 2400 linear feet of materials.
They are still undergoing review by Jen, et al in SCUA, but many have already been approved! So here's a quick view of the new ArchivesSpace public user interface, and you can a little along the side about series...
Next steps

• Clean up code revisions to Stead
  ○ Documentation
  ○ Share back with Jason Ronallo/NCSU

• Problem items
  ○ Some collections require A LOT of additional manual work to get into the template and go through the Stead conversion

• Final review by Labor.
  ○ Everything is getting reviewed, and having content that could not be added via conversion added after the fact prior to being published.

Next steps - finish cleaning up the code, and documenting it, and sharing back with Jason Ronallo for possible inclusion into his original codebase, or to maybe at least have him link to our expanded STEAD code.

There are still a handful of problem collections that have required additional manual work and research to get into the stead template, and there are some outstanding questions remaining on those.

And, as I mentioned, the final review by Labor, as well as some work that can be done using the API to add parts that we just weren't able to accommodate using Stead.
So, we're pretty close to the finish line
I wanted to acknowledge a few folks who have personally helped me with this project along the way, or contributed significant work on this, specifically, Megan O'Hern, Sydney Vaile, Caitlin Rizzo, Joanne Archer, Jen Eidson, and Dave Mayo.
Questions?

Thanks!
blparker@umd.edu