Managing Archival Metadata Migration

(is Maddening)

Bria Parker
Head, Discovery and Metadata Services
University of Maryland Libraries
Our legacy archival system resides in an Access Database lovingly named The Beast. Having the data in a database provides the opportunity and ability to maintain a semblance of structure in the data. However, after years of use by MANY people, quirks form, data is formatted differently over time, conventions change, and, honest, innocent errors are made. So, our legacy data was in no way consistent.
Oh how we longed to take advantage of a system where we could better enforce controlled language and standards, while also having our data structured in a more machine actionable manner and be better able to manipulate and report on our data.
But how would we get there?
I will be focusing today on accessions and finding aids. And getting from a database to archivesspace.
There was not some single magical solution. No one tool could solve all of our issues (or, rather, we did not necessarily have all the resources to develop one tool)
In the end, it was an iterative process - the data was gone over multiple time - fixing certain things, so that we could then expose more problems and fix more things in the resulting metadata
So, we started with accessions - we figured this metadata was the lowest hanging fruit...
The accessions metadata could be exported out of the beast into excel spreadsheets, which was a great place to start, since it needed to then be mapped into the archivesspace accessions import template (a csv file). We ran into some challenges right off the bat, as the template changed with some of the new releases, so I’d be wondering why import failed - and oh, yeah, that part of the importer has changed. We were able to use Open Refine for the majority (or ALL?) of the normalization work needed. Open Refine - the metadata librarian’s swiss army knife - allowed us to normalize date formats (yyyy-mm-dd), apply controlled vocabularies, and also use faceting to find and remediate data inconsistencies. However, we couldn’t escape manual work, since some of the errors or missing data required going back to the control files. Time intensive but doable.
The accessions import process also gave us the opportunity to explore the archivesspace API, as while we were planning and completing the accessions cleanup process, part of the accessions record structure, and thus the importer changed, and processing status was moved to event objects. Well then. So Why did we need to use the API? We did not want to manually add each processing status, and there was not batch import function available in the interface, but we still needed a way to get this data in and relate it to the appropriate record. It was a simple-ish task to introduce us to how the API worked - event records are relatively small. Plus, I’m not a programmer, so I needed something manageable.

So, I got the accession in, pulled the URI back out (with accession_id to match it) then used openrefine to make a separate set of data with the processing status, some other static metadata (event type) and the accession URI to relate to, made a json template for formatting the data, and were able to export JSON out. (json is the format that archivesspace objects are in) Then, using a bash script and curl, I loaded those events via the API. Maybe not the most elegant solution, but it was my first attempt at this, plus, it had the added advantage of working.
Once we finished accessions, we started really planning for EAD - this would be a much more involved clean up process
While all of the collection data was stored in The Beast, a converter, written in Java, was used to generate the EAD XML. While the EAD generated by this converter was valid, it was not “ArchivesSpace” valid. However, no one wanted to try to alter the converter, the expertise around the development and maintenance of the converter had long since left the Libraries. We were fortunate enough to have a field study student review the EAD and identify the problems so we could at least get a picture of how much cleanup we needed to do.
Key Problems

Missing nested tags and attributes

Tags with no content

The following errors were found:

```
notes/0/content : At least 1 item(s) is required

For JSONModel():
  #@JSONModel({resource})
  { "jsonmodel_type": "resource",  "external_id": "[{}],  "subjects": "[{}],  "linked_events":
```

These were blockers for getting EAD imported to ArchivesSpace - not even something we could fix post-import - the importer would fail, do not pass go, do not collect $200
He also found other problems, that weren’t necessarily blockers, but fixing them would make our metadata so much better. Normalizing dates, making our collection titles more DACS compliant; normalizing extents, making our collection identifiers unique, instead of being duplicate to a related accession identifier, and inserted the handle URIs.
While we waited for some programming help with most of those problems, I took on the date normalization. The lovely people at the Bentley Library had already tackled this very issue and had shared their resources - scripts and workflows - on how to do this.

First, I used their script to run through a directory of EAD XML files, identify the most common date formats in unitdates, and based on those formats generate and insert an appropriately formatted normal attribute for that unitdate.
That couldn’t catch everything, because a world where all dates are consistently formatted is a magical unicorn world. So I also used a script to extract all the dates that were NOT normalized - that created a csv file. I could then load that into OpenRefine and use the tools in OpenRefine to normalize the dates. And again, use a Bentley script to take the normalized data and inserted it back as a normal attribute. It really is an incredible workflow and scripts that they developed, and if you google Bentley archival integration blog and look at the “date saga” tag, you'll get a wonderful overview of how to use the workflow.
As I said, I used OpenRefine to clean these dates - who’s heard of OpenRefine? Who’s actually used it and witnessed its magnificent powers?

One powerful tool in openrefine is regular expression - these give you a way of representing alpha-numeric patterns that in this case can be used for sophisticated find and replace operations. So using the text filter function in open refine, I can look for all dates in a certain format like this - allowing me to work on them as a group.
Custom text transform on column normalized

<table>
<thead>
<tr>
<th>row</th>
<th>value</th>
<th>value.replace(/\d{4}-\d{4}, (\d{4})/, '$1/$2')</th>
</tr>
</thead>
<tbody>
<tr>
<td>599</td>
<td>1905-1938, 1947</td>
<td>1905/1947</td>
</tr>
</tbody>
</table>

value.replace(/(\d{4})-(\d{4}), (\d{4})/, '$1/$2')
With creating regular expressions you can normalizes large numbers of dates in relatively little time. With this workflow, I was able to normalize other 200,000 date statements. Makes the dates machine readable, which leads to better date searching/faceting for our users.
So, on to the rest of the problems
One of our challenges - and one I know that many of you face - is getting the time from programmers to help with some of these solutions. So when we were able to get the time of a programmer, we try to get as much as possible - but there’s not always much time.
He was able to put together a Python script that solved a lot of our problems - the missing tags, the missing attributes, some of the formatting and normalization, and to remove/alter some of our local practices to fit better into the archivesspace resource metadata model. He ran all of the EAD through, fixed character encoding as well, so...

**Programmer was able to:**

- Add handle URI into the `<eadid>`
- Add title attribute to `<dao>`
- Add missing `<extent>` tags
- Normalize collection titles
- Move scope and content notes
- Remove unnecessary elements (alternate abstracts)
- Add missing container tags
So we’re done right?

Nope.

Yeah, not quite
Iteration

The Python transformation fixed A LOT, but not everything.

Still had issues with title attributes in <dao>

Python script ignored titles with internal tags (i.e. italics)

Container attribute mismatches

Here’s where iteration really became the reality - the programmer took care of a lot of our problems, but did not have the time to solve everything. We still had issues with attributes, both in dao tags and in containers (the parent/child attributes were wonky)
Iteration

Other issues found after the script was done

Remove some extraneous data that was not relevant and was not mapped well anyways

Needed special handling for when reels are top containers

These were added as free text in <physdesc> and required special parsing

Once we were able to fix the EAD enough to use the importer, we were able to uncover other problems. - so we found that what we had in physloc did not map well at all, and we wouldn’t need it anyways since we’d use the location stuff in archivesspace. We also found collections that had their own quirks that required some special handling - collections with reels that did not use container tags, but free text in physdesc tags.
Top Container Management

Wanted to better prepare the EAD to work with the Top Container Management functionality

 Dummy barcodes
   Allows us to disambiguate boxes across series and collections

 Component unique identifiers
   Also helps with disambiguation

Instances

So, let's say a few things about top container management - that was another problem? Issue? Shortcoming? We found with our EAD. We wanted to take advantage of Top Container management functionality in ArchivesSpace and to do so we needed to make some not insignificant changes in our EAD. We needed to insert barcodes - and since none of our boxes had barcodes, I created dummy barcodes - to better disambiguate boxes and so that if we had multiple box 1s in different series, we could differentiate between those boxes. I also need to add component unique identifiers, which also helped with that disambiguation. Thankfully, the importer was set up to read a label attribute in the container tag to import the dummy barcode and instance type, and look for component unique id in unitid. Cool! How am I going to do that?
The way container id attributes had been created actually helped us here! Even though box number did start over at the series for many collections, the box id was actually unique within the collection (number of boxes.boxnumber in series). We could then account for collections where the numbering started over at the series level, but also know how many boxes were in the collection, since the number before the decimal was always the running total, while the number after was the number in the series (in those cases).

The dsc/@id was unique to each collection, and the box/@id was unique within the collection.

```xml
<xsl:attribute name="label">
  <xsl:copy-of select="concat('mixed_materials ', ', ', ancestor::dsc/@id, self::container/@id, ', ')')"/>
</xsl:attribute>
```

```xml
<xsl:copy-of
  select="concat('Mixed materials ', ', ', ancestor::dsc/@id, self::container/@id, ', ')')"/>
```
We also had some special cases where multiple instances are described, but the second instance is not in a container element, but instead noted in physDesc. We used xslt to pull out the reel/frame data, parse it, and create new instances for the objects, including instance type, reel/frame number, component_unique_id, and dummy barcode

Y’all, I’m pretty proud of this little bit of xslt - so this slide is pretty much a not-so-humble brag.
More Iteration + QA

● adapted the date normalization process for normalizing extent statements/types, as well as creating new collection IDs
● SCHEMATRON!!!
  ○ Was able to find/use a schematron developed for importing to ArchivesSpace to find errors that would fail the import, such as:
    ○ End dates occurring before begin dates
    ○ Missing unittitles
    ○ Missing revision info (usually the date)

But why stop there? I decided to make another iteration over the EAD and modify the date normalization scripts and processes to normalize extent statements and types so we could have more uniform extents! I also modified the process to derive and insert unique collection IDs.

I would still run into problems at import, so I was able to use a schematron (found it linked in a blog post from Yale!)
Well, we got everything in!! But! There were still things the importer could NOT do for us, regardless of how we formatted our data.
Python + ArchivesSpaceAPI = <3

For example, the import would only put unitid into the id_0 field and would not split out into the id_1, id_2, etc. fields available. For this we used Python and the API.

Lora has probably explained how this worked much more elegantly - but at this point, I'm comfortable enough working with the API through python (I've basically been able to teach myself enough Python to do this - mostly by copying other people's work!).
APIs are powerful!

Use same type of script to look at a collection’s title and set a resource_type

Unpublish resources

Set collection-level access restrictions based on container level

Quickly get data out (when current reports don’t cut it)
Lessons learned

Structured, normalized, clean metadata is worth the time and effort

It’s an opportunity, not a punishment!

Don’t reinvent the wheel

Someone, somewhere has done something similar - find it and repurpose it.

Be patient with yourself when learning a new skill

Be patient - you will make mistakes, it probably won’t work the first time, but realize it’s that way for just about everyone.
Acknowledgements

We would not have gotten this far without standing on the shoulders of...

Noah Huffman
Dave Mayo
Maureen Callahan
Dallas Pillen
Max Eckard
Questions?
Thank you!

Bria Parker
blparker@umd.edu
@blparker