Managing Archival Metadata Migration

(is Maddening)

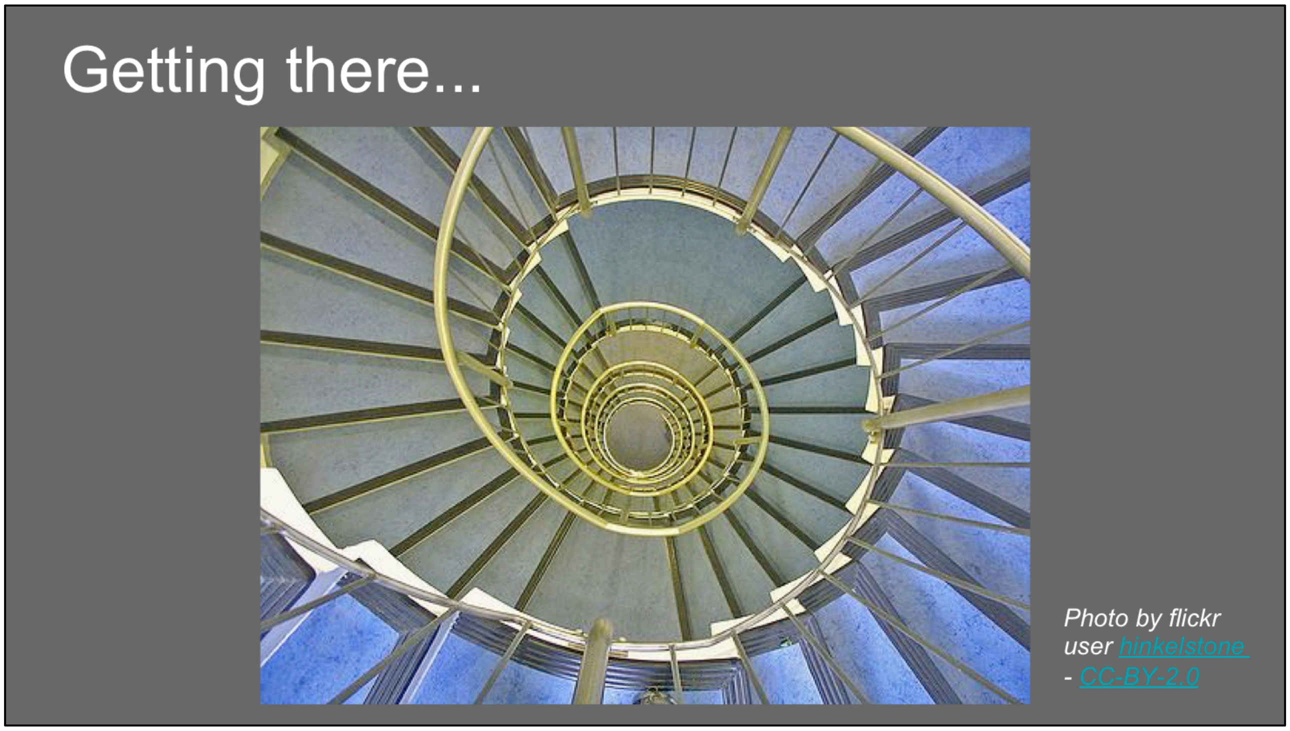
Bria Parker
Head, Discovery and Metadata Services
University of Maryland Libraries

Where we were...

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

Where we wanted to be...

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.

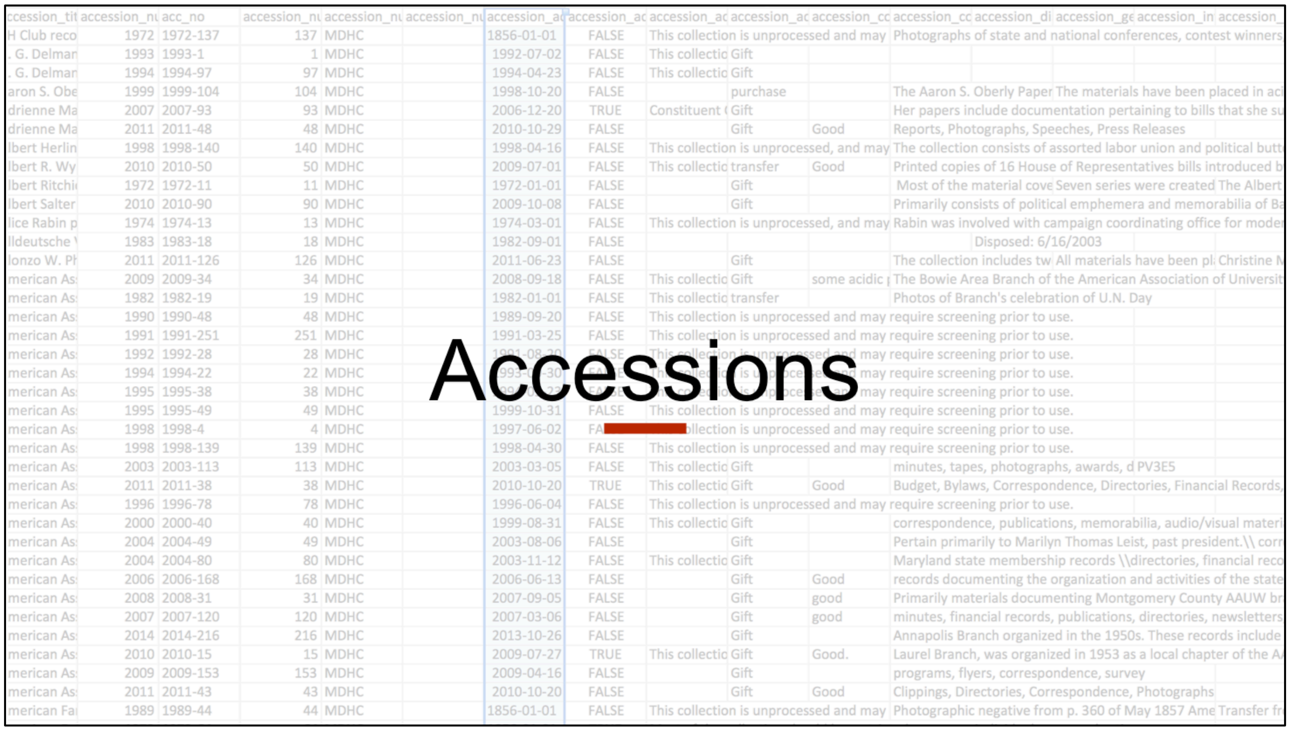


Getting there...

Photo by flickr user hinkelstone - CC-BY-2.0

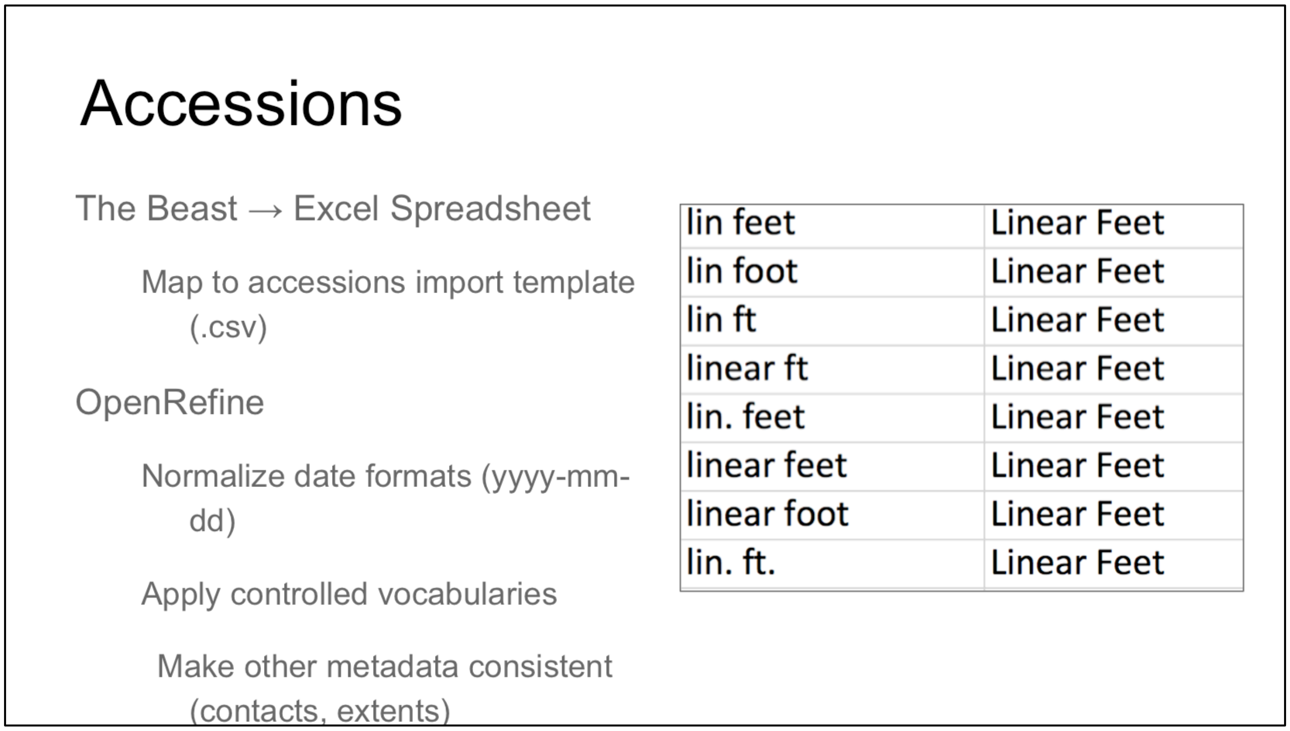
But how would we get there?
I will be focusing today on accessions and finding aids. And getting from a database to archivesspace.
There was not some single magical solution. No one tool could solve all of our issues (or, rather, we did not necessarily have all the resources to develop one tool)
In the 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

Accessions



ccession_tit	accession_nu	acc_no	accession_nu	accession_nu	accession_nu	accession_ac	accession_ac	accession_ac	accession_ac	accession_cc	accession_cc	accession_di	accession_ge	accession_in	accession_
H Club reco	1972	1972-137	137	MDHC		1856-01-01	FALSE	This collection is unprocessed and may			Photographs of state and national conferences, contest winners				
. G. Delman	1993	1993-1	1	MDHC		1992-07-02	FALSE	This collectio	Gift						
. G. Delman	1994	1994-97	97	MDHC		1994-04-23	FALSE	This collectio	Gift						
aron S. Obe	1999	1999-104	104	MDHC		1998-10-20	FALSE		purchase		The Aaron S. Oberly Paper		The materials have been placed in aci		
drienne Ma	2007	2007-93	93	MDHC		2006-12-20	TRUE	Constituent (	Gift		Her papers include documentation pertaining to bills that she su				
drienne Ma	2011	2011-48	48	MDHC		2010-10-29	FALSE		Gift	Good	Reports, Photographs, Speeches, Press Releases				
lbert Herlin	1998	1998-140	140	MDHC		1998-04-16	FALSE	This collection is unprocessed, and may			The collection consists of assorted labor union and political butt				
lbert R. Wy	2010	2010-50	50	MDHC		2009-07-01	FALSE	This collectio	transfer	Good	Printed copies of 16 House of Representatives bills introduced b				
lbert Ritchi	1972	1972-11	11	MDHC		1972-01-01	FALSE		Gift		Most of the material cove		Seven series were created		The Albert
lbert Salter	2010	2010-90	90	MDHC		2009-10-08	FALSE		Gift		Primarily consists of political emphemera and memorabilia of Ba				
lice Rabin p	1974	1974-13	13	MDHC		1974-03-01	FALSE	This collection is unprocessed, and may			Rabin was involved with campaign coordinating office for moder				
lldeutsche \	1983	1983-18	18	MDHC		1982-09-01	FALSE					Disposed: 6/16/2003			
lonzo W. Ph	2011	2011-126	126	MDHC		2011-06-23	FALSE		Gift		The collection includes tw		All materials have been pl		Christine M
merican As	2009	2009-34	34	MDHC		2008-09-18	FALSE	This collectio	Gift	some acidic p	The Bowie Area Branch of the American Association of Universit				
merican As	1982	1982-19	19	MDHC		1982-01-01	FALSE	This collectio	transfer		Photos of Branch's celebration of U.N. Day				
merican As	1990	1990-48	48	MDHC		1989-09-20	FALSE	This collection is unprocessed and may require screening prior to use.							
merican As	1991	1991-251	251	MDHC		1991-03-25	FALSE	This collection is unprocessed and may require screening prior to use.							
merican As	1992	1992-28	28	MDHC		1991-08-20	FALSE	This collection is unprocessed and may require screening prior to use.							
merican As	1994	1994-22	22	MDHC		[illegible]	FALSE	This collection is unprocessed and may require screening prior to use.							
merican As	1995	1995-38	38	MDHC		[illegible]	FALSE	This collection is unprocessed and may require screening prior to use.							
merican As	1995	1995-49	49	MDHC		1999-10-31	FALSE	This collection is unprocessed and may require screening prior to use.							
merican As	1998	1998-4	4	MDHC		1997-06-02	FA	ollection is unprocessed and may require screening prior to use.							
merican As	1998	1998-139	139	MDHC		1998-04-30	FALSE	This collection is unprocessed and may require screening prior to use.							
merican As	2003	2003-113	113	MDHC		2003-03-05	FALSE	This collectio	Gift		minutes, tapes, photographs, awards, d			PV3E5	
merican As	2011	2011-38	38	MDHC		2010-10-20	TRUE	This collectio	Gift	Good	Budget, Bylaws, Correspondence, Directories, Financial Records,				
merican As	1996	1996-78	78	MDHC		1996-06-04	FALSE	This collection is unprocessed and may require screening prior to use.							
merican As	2000	2000-40	40	MDHC		1999-08-31	FALSE	This collectio	Gift		correspondence, publications, memorabilia, audio/visual materi				
merican As	2004	2004-49	49	MDHC		2003-08-06	FALSE		Gift		Pertain primarily to Marilyn Thomas Leist, past president.\\ corr				
merican As	2004	2004-80	80	MDHC		2003-11-12	FALSE	This collectio	Gift		Maryland state membership records \\directories, financial reco				
merican As	2006	2006-168	168	MDHC		2006-06-13	FALSE		Gift	Good	records documenting the organization and activities of the state				
merican As	2008	2008-31	31	MDHC		2007-09-05	FALSE		Gift	good	Primarily materials documenting Montgomery County AAUW br				
merican As	2007	2007-120	120	MDHC		2007-03-06	FALSE		Gift	good	minutes, financial records, publications, directories, newsletters				
merican As	2014	2014-216	216	MDHC		2013-10-26	FALSE		Gift		Annapolis Branch organized in the 1950s. These records include				
merican As	2010	2010-15	15	MDHC		2009-07-27	TRUE	This collectio	Gift	Good.	Laurel Branch, was organized in 1953 as a local chapter of the A				
merican As	2009	2009-153	153	MDHC		2009-04-16	FALSE		Gift		programs, flyers, correspondence, survey				
merican As	2011	2011-43	43	MDHC		2010-10-20	FALSE		Gift	Good	Clippings, Directories, Correspondence, Photographs				
merican Fa	1989	1989-44	44	MDHC		1856-01-01	FALSE	This collection is unprocessed and may			Photographic negative from p. 360 of May 1857 Ame				Transfer fr

So, we started with accessions - we figured this metadata was the lowest hanging fruit...



Accessions

The Beast → Excel Spreadsheet

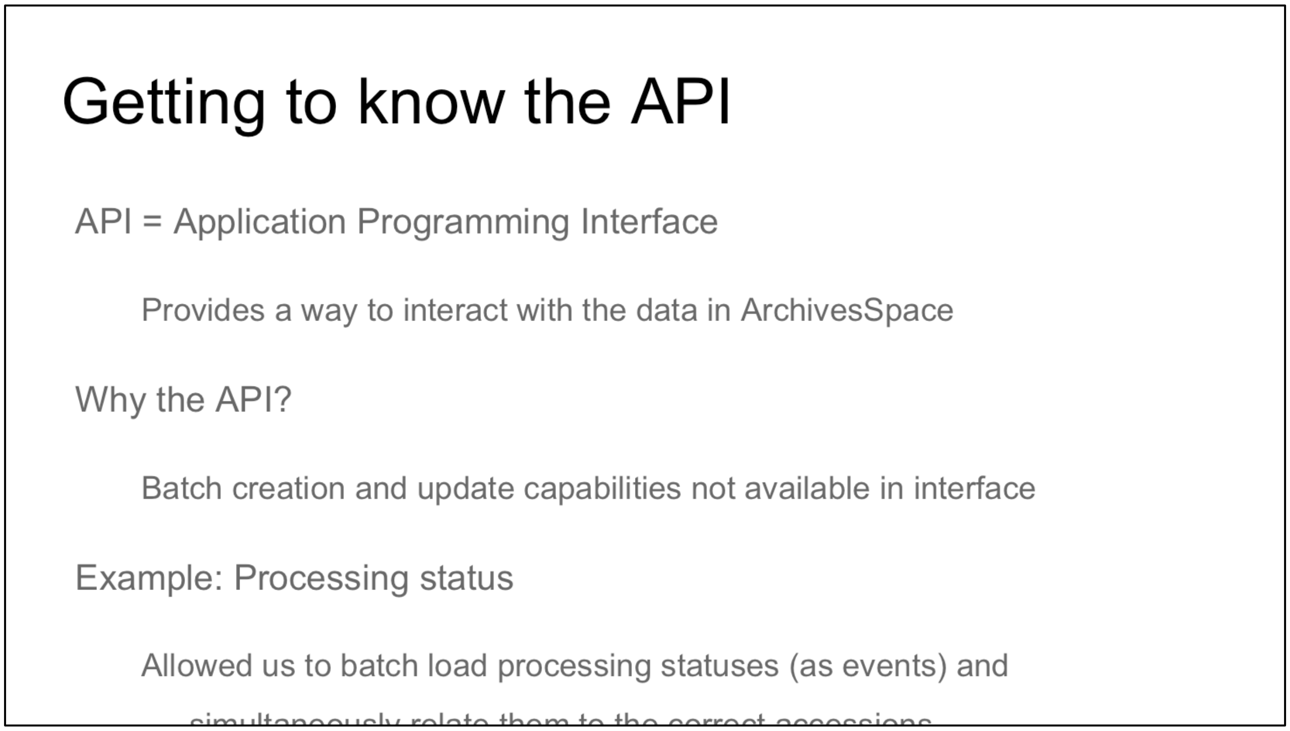
- Map to accessions import template (.csv)

OpenRefine

- Normalize date formats (yyyy-mm-dd)
- Apply controlled vocabularies
- Make other metadata consistent (contacts, extents)

lin feet	Linear Feet
lin foot	Linear Feet
lin ft	Linear Feet
linear ft	Linear Feet
lin. feet	Linear Feet
linear feet	Linear Feet
linear foot	Linear Feet
lin. ft.	Linear Feet

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, use facets and such to map to/apply controlled vocabularies for extent types, etc., 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.



Getting to know the API

API = Application Programming Interface

Provides a way to interact with the data in ArchivesSpace

Why the API?

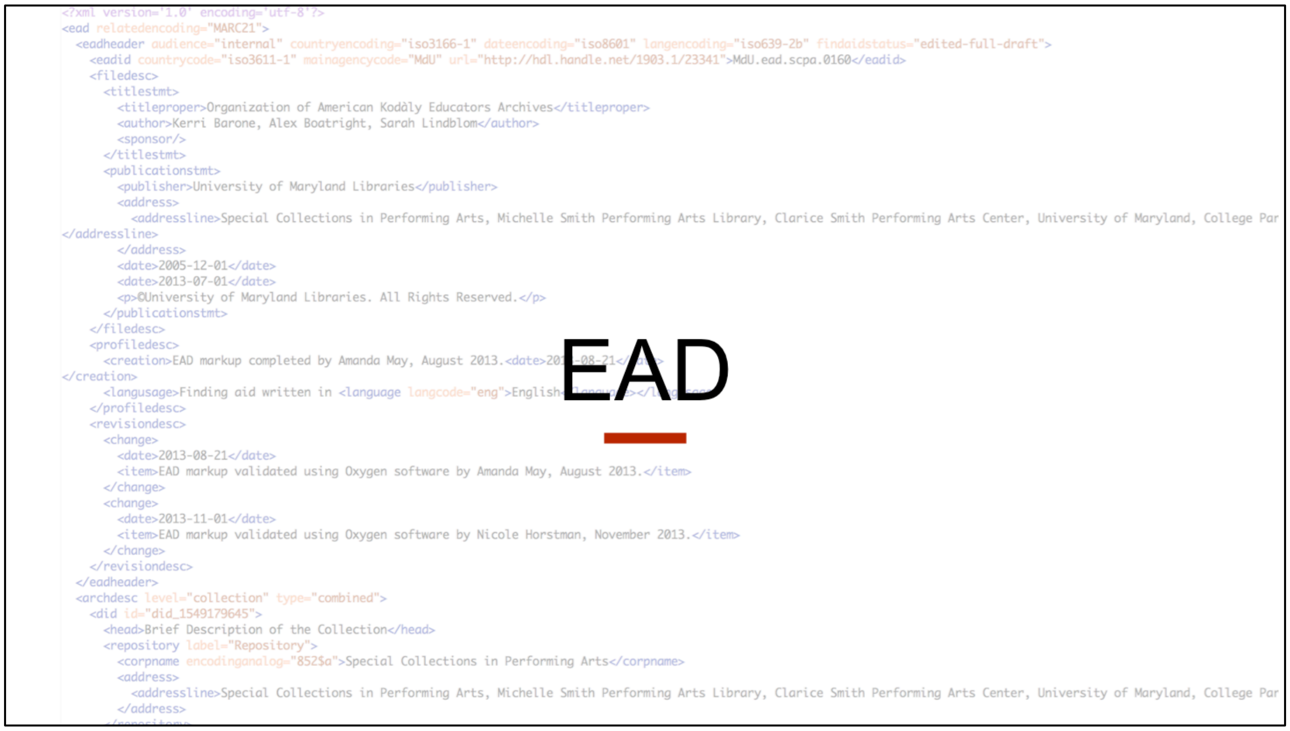
Batch creation and update capabilities not available in interface

Example: Processing status

Allowed us to batch load processing statuses (as events) and simultaneously relate them to the correct accessions

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.

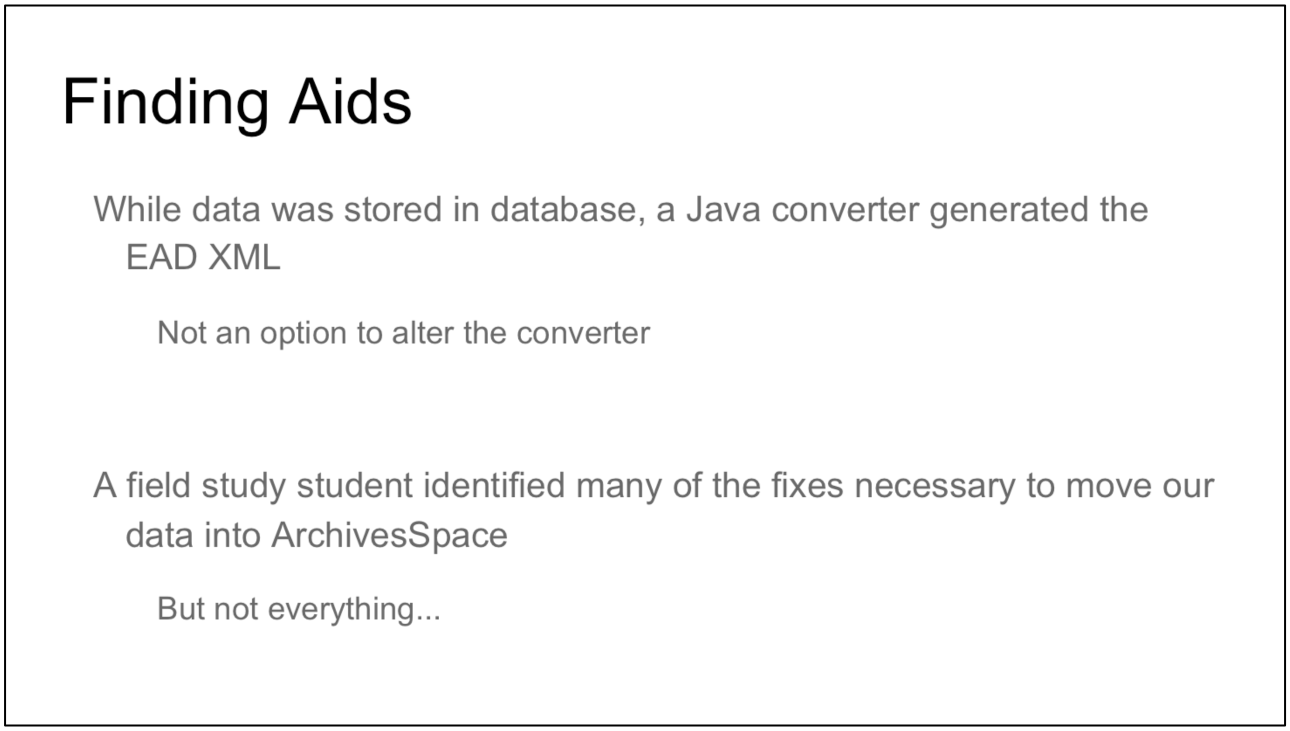
So, I got the accession in, pulled the URI back out (with accession_id to match it) then used openrefine to make a separate set of data with the processing status, some other static metadata (event type) and the accession URI to relate to, made a json template for formatting the data, and were able to 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.



EAD

```xml
<?xml version='1.0' encoding='utf-8'?>
<ead relatedencoding="MARC21">
  <eadheader audience="internal" countryencoding="iso3166-1" dateencoding="iso8601" langencoding="iso639-2b" findaidstatus="edited-full-draft">
    <eadid countrycode="iso3611-1" mainagencycode="MdU" url="http://hdl.handle.net/1903.1/23341">MdU.ead.scpa.0160</eadid>
    <filedesc>
      <titlestmt>
        <titleproper>Organization of American Kodàly Educators Archives</titleproper>
        <author>Kerri Barone, Alex Boatright, Sarah Lindblom</author>
        <sponsor/>
      </titlestmt>
      <publicationstmt>
        <publisher>University of Maryland Libraries</publisher>
        <address>
          <addressline>Special Collections in Performing Arts, Michelle Smith Performing Arts Library, Clarice Smith Performing Arts Center, University of Maryland, College Par
</addressline>
        </address>
        <date>2005-12-01</date>
        <date>2013-07-01</date>
        <p>©University of Maryland Libraries. All Rights Reserved.</p>
      </publicationstmt>
    </filedesc>
    <profiledesc>
      <creation>EAD markup completed by Amanda May, August 2013.<date>20[illegible]-08-21</[illegible]>
</creation>
      <langusage>Finding aid written in <language langcode="eng">English[illegible]
    </profiledesc>
    <revisiondesc>
      <change>
        <date>2013-08-21</date>
        <item>EAD markup validated using Oxygen software by Amanda May, August 2013.</item>
      </change>
      <change>
        <date>2013-11-01</date>
        <item>EAD markup validated using Oxygen software by Nicole Horstman, November 2013.</item>
      </change>
    </revisiondesc>
  </eadheader>
  <archdesc level="collection" type="combined">
    <did id="did_1549179645">
      <head>Brief Description of the Collection</head>
      <repository label="Repository">
        <corpname encodinganalog="852$a">Special Collections in Performing Arts</corpname>
        <address>
          <addressline>Special Collections in Performing Arts, Michelle Smith Performing Arts Library, Clarice Smith Performing Arts Center, University of Maryland, College Par
        </address>
      </repository>
```

Once we finished accessions, we started really planning for EAD - this would be a much more involved clean up process



Finding Aids

- While data was stored in database, a Java converter generated the EAD XML
  - Not an option to alter the converter
- A field study student identified many of the fixes necessary to move our data into ArchivesSpace
  - But not everything...

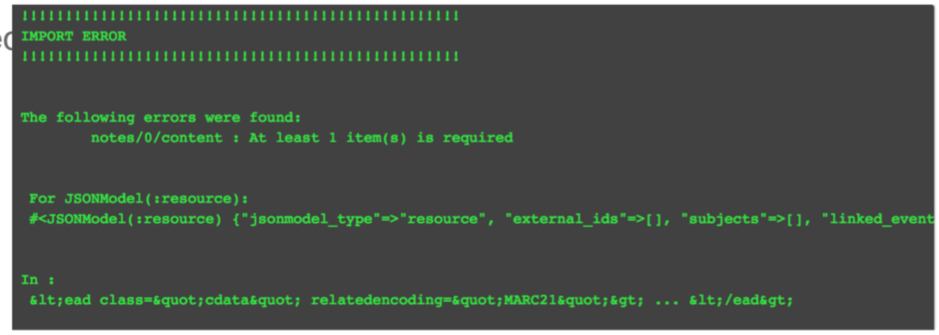
While all of the collection data was stored in The Beast, a converter, written in Java, was used to generate the EAD XML. While the EAD generated by this converter was valid, it was not “ArchivesSpace” valid. However, no one wanted to try to alter the converter, the expertise around the development and 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

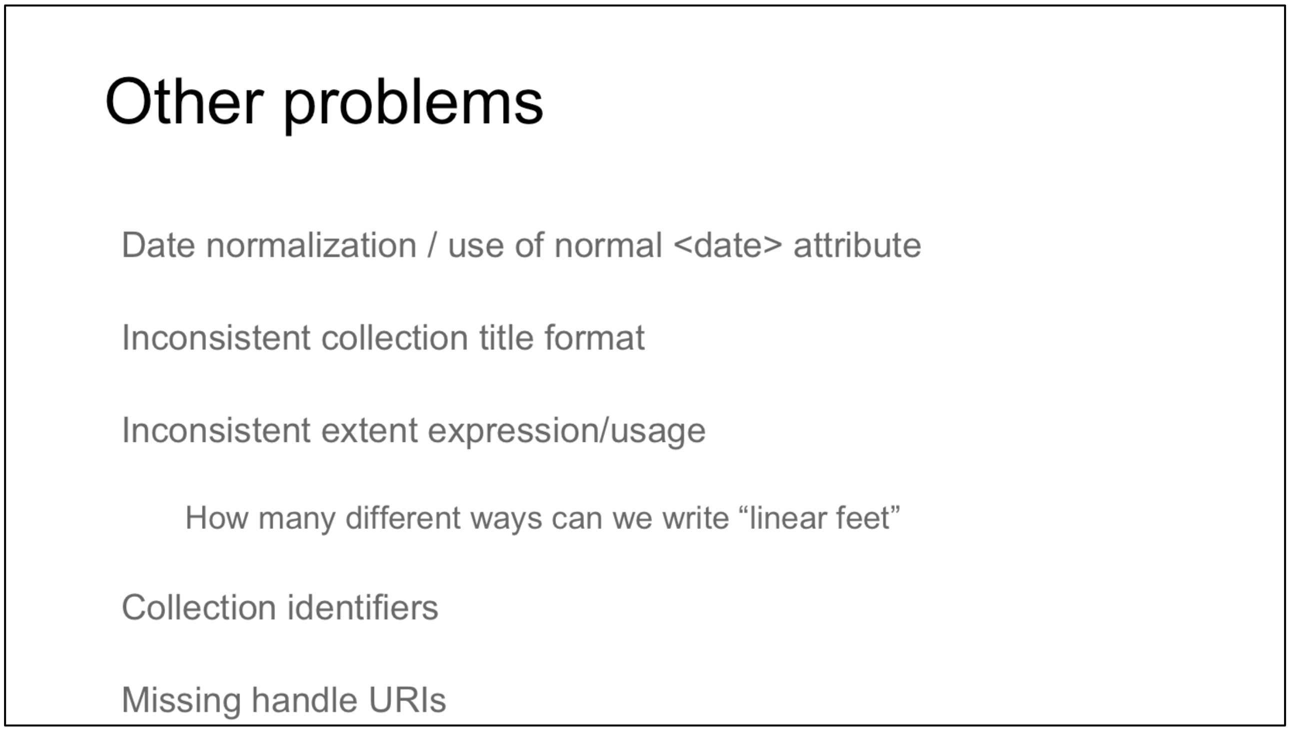
Spec



```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
IMPORT ERROR
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!


The following errors were found:
        notes/0/content : At least 1 item(s) is required


 For JSONModel(:resource):
 #<JSONModel(:resource) {"jsonmodel_type"=>"resource", "external_ids"=>[], "subjects"=>[], "linked_event


In :
 &lt;ead class=&quot;cdata&quot; relatedencoding=&quot;MARC21&quot;&gt; ... &lt;/ead&gt;
```

The key problems the field study student discovered were various missing nested tags, and attributes, tags/sections with no content (so, entire sections with just an empty <p> tag, and character encoding problems

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



Other problems

Date normalization / use of normal <date> attribute

Inconsistent collection title format

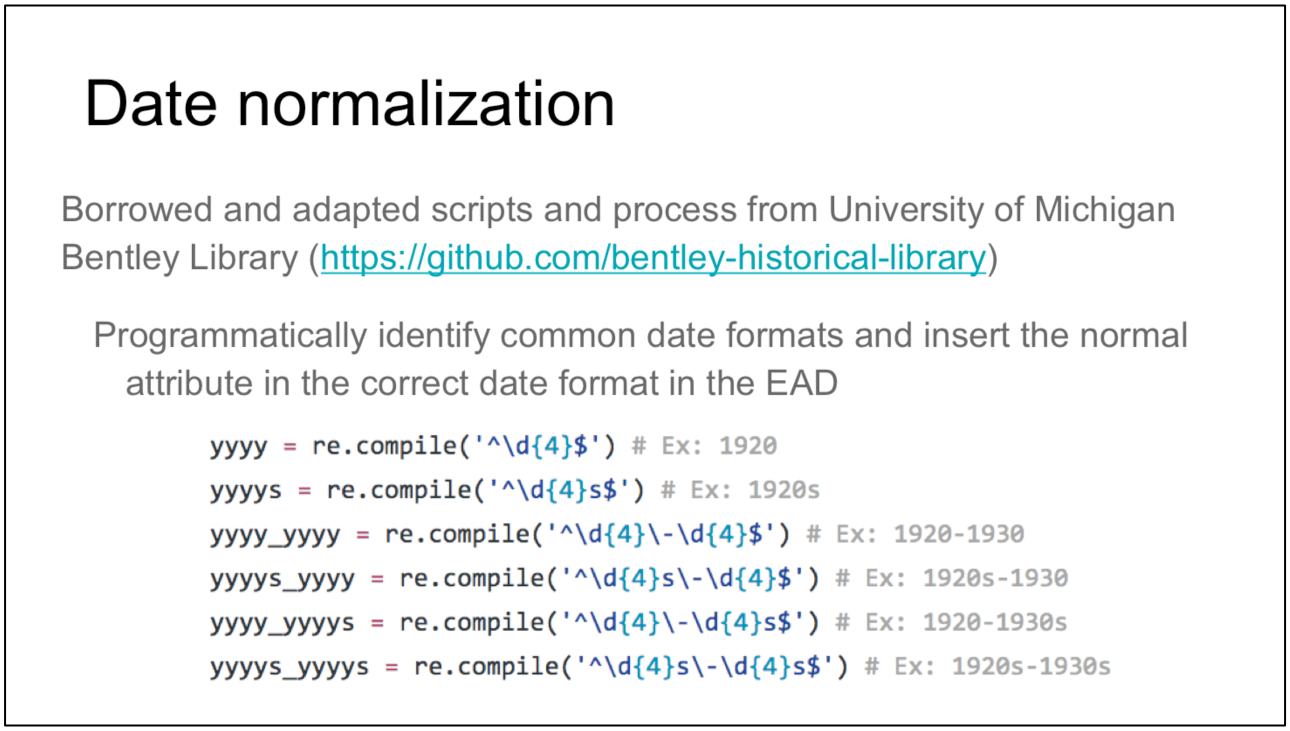
Inconsistent extent expression/usage

- How many different ways can we write “linear feet”

Collection identifiers

Missing handle URIs

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.



Date normalization

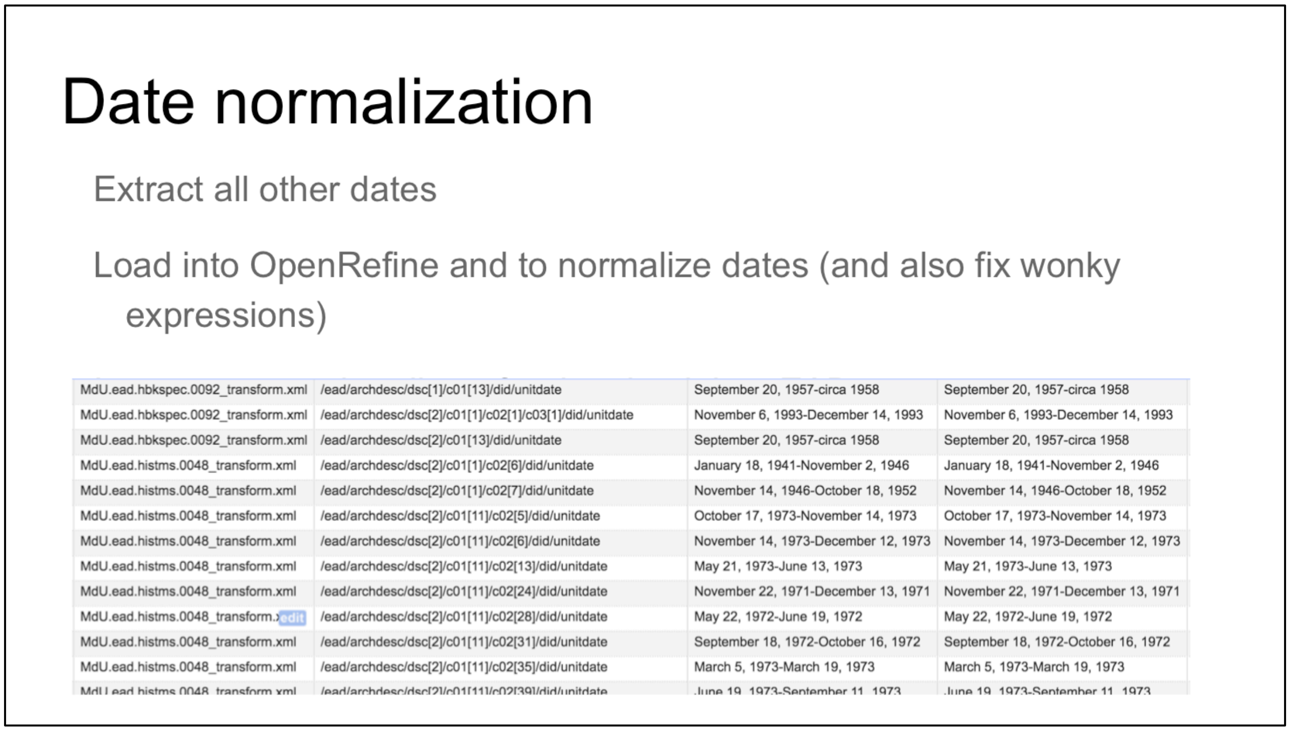
Borrowed and adapted scripts and process from University of Michigan Bentley Library (https://github.com/bentley-historical-library)

- Programmatically identify common date formats and insert the normal attribute in the correct date format in the EAD

```
yyyy = re.compile('^\d{4}$') # Ex: 1920
yyyys = re.compile('^\d{4}s$') # Ex: 1920s
yyyy_yyyy = re.compile('^\d{4}\-\d{4}$') # Ex: 1920-1930
yyyys_yyyy = re.compile('^\d{4}s\-\d{4}$') # Ex: 1920s-1930
yyyy_yyyys = re.compile('^\d{4}\-\d{4}s$') # Ex: 1920-1930s
yyyys_yyyys = re.compile('^\d{4}s\-\d{4}s$') # Ex: 1920s-1930s
```

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



Date normalization

Extract all other dates

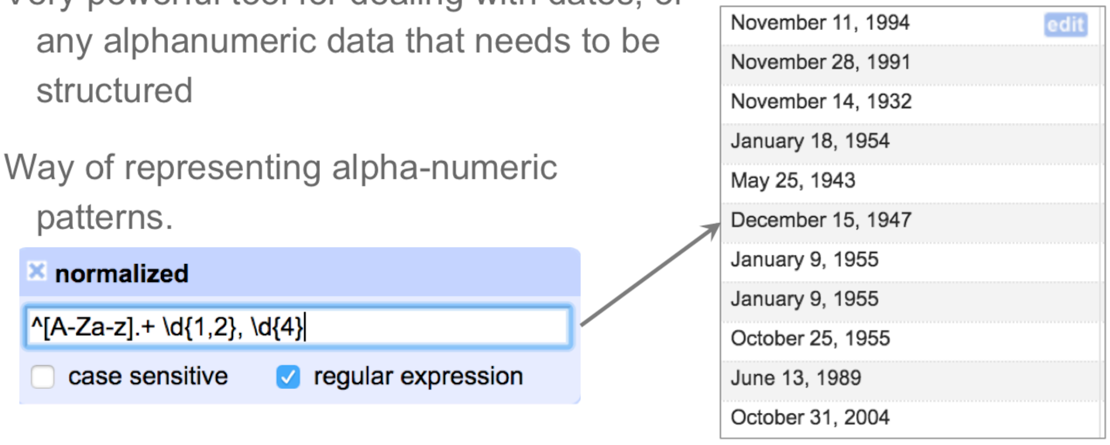
Load into OpenRefine and to normalize dates (and also fix wonky expressions)

MdU.ead.hbkspec.0092_transform.xml	/ead/archdesc/dsc[1]/c01[13]/did/unitdate	September 20, 1957-circa 1958	September 20, 1957-circa 1958
MdU.ead.hbkspec.0092_transform.xml	/ead/archdesc/dsc[2]/c01[1]/c02[1]/c03[1]/did/unitdate	November 6, 1993-December 14, 1993	November 6, 1993-December 14, 1993
MdU.ead.hbkspec.0092_transform.xml	/ead/archdesc/dsc[2]/c01[13]/did/unitdate	September 20, 1957-circa 1958	September 20, 1957-circa 1958
MdU.ead.histms.0048_transform.xml	/ead/archdesc/dsc[2]/c01[1]/c02[6]/did/unitdate	January 18, 1941-November 2, 1946	January 18, 1941-November 2, 1946
MdU.ead.histms.0048_transform.xml	/ead/archdesc/dsc[2]/c01[1]/c02[7]/did/unitdate	November 14, 1946-October 18, 1952	November 14, 1946-October 18, 1952
MdU.ead.histms.0048_transform.xml	/ead/archdesc/dsc[2]/c01[11]/c02[5]/did/unitdate	October 17, 1973-November 14, 1973	October 17, 1973-November 14, 1973
MdU.ead.histms.0048_transform.xml	/ead/archdesc/dsc[2]/c01[11]/c02[6]/did/unitdate	November 14, 1973-December 12, 1973	November 14, 1973-December 12, 1973
MdU.ead.histms.0048_transform.xml	/ead/archdesc/dsc[2]/c01[11]/c02[13]/did/unitdate	May 21, 1973-June 13, 1973	May 21, 1973-June 13, 1973
MdU.ead.histms.0048_transform.xml	/ead/archdesc/dsc[2]/c01[11]/c02[24]/did/unitdate	November 22, 1971-December 13, 1971	November 22, 1971-December 13, 1971
MdU.ead.histms.0048_transform.xedit	/ead/archdesc/dsc[2]/c01[11]/c02[28]/did/unitdate	May 22, 1972-June 19, 1972	May 22, 1972-June 19, 1972
MdU.ead.histms.0048_transform.xml	/ead/archdesc/dsc[2]/c01[11]/c02[31]/did/unitdate	September 18, 1972-October 16, 1972	September 18, 1972-October 16, 1972
MdU.ead.histms.0048_transform.xml	/ead/archdesc/dsc[2]/c01[11]/c02[35]/did/unitdate	March 5, 1973-March 19, 1973	March 5, 1973-March 19, 1973
MdU.ead.histms.0048_transform.xml	/ead/archdesc/dsc[2]/c01[11]/c02[39]/did/unitdate	June 19, 1973-September 11, 1973	June 19, 1973-September 11, 1973

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.

Regular Expressions

- Very powerful tool for dealing with dates, or any alphanumeric data that needs to be structured
- Way of representing alpha-numeric patterns.



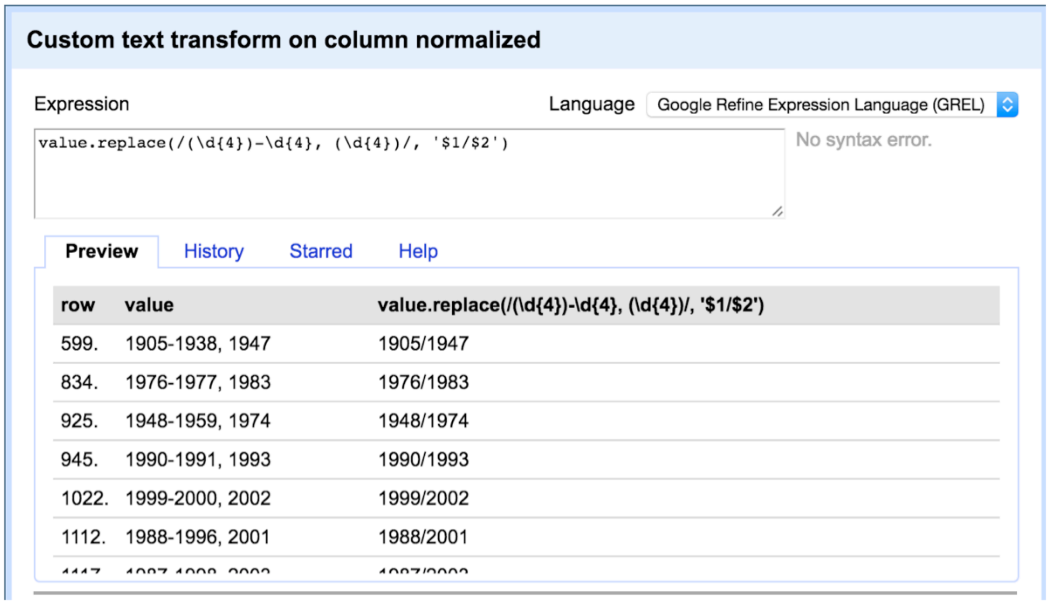
```
^[A-Za-z].+ \d{1,2}, \d{4}
```

[ ] case sensitive [✓] regular expression

```
November 11, 1994
November 28, 1991
November 14, 1932
January 18, 1954
May 25, 1943
December 15, 1947
January 9, 1955
January 9, 1955
October 25, 1955
June 13, 1989
October 31, 2004
```

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

Expression

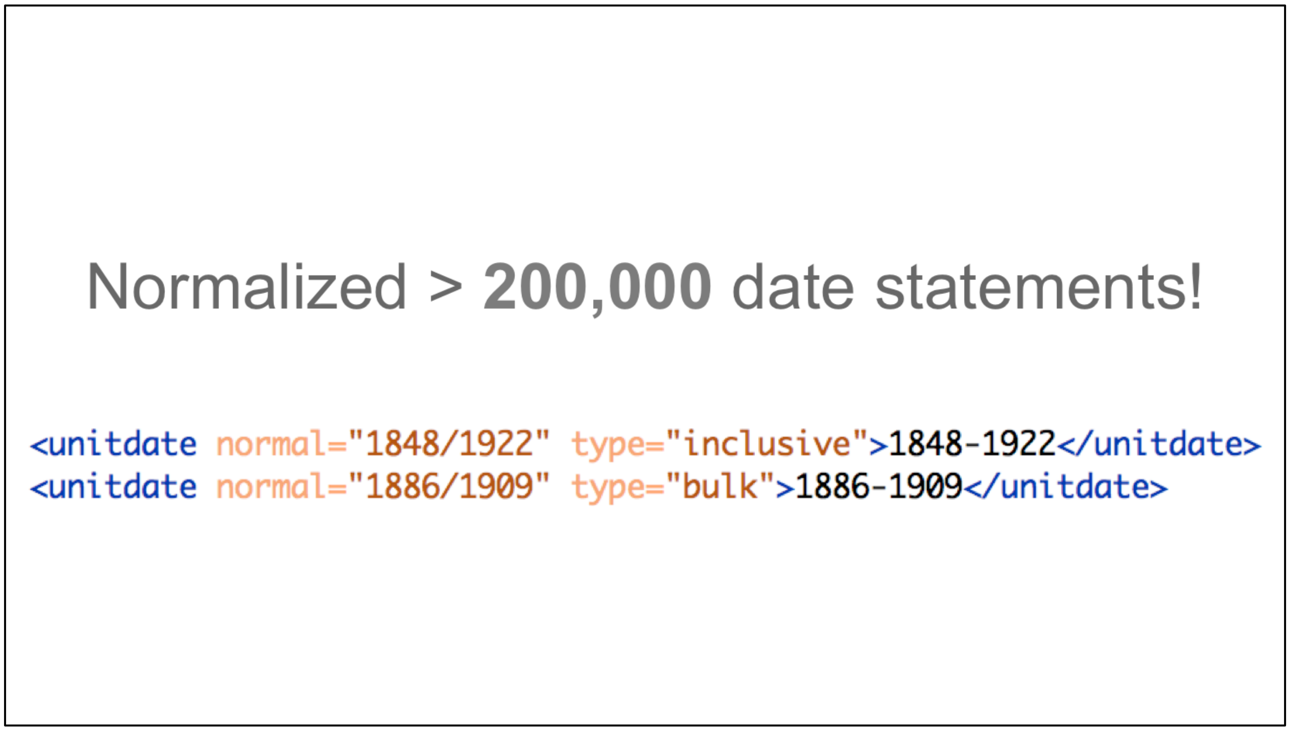
```
value.replace(/(\d{4})-\d{4}, (\d{4})/, '$1/$2')
```

Language: Google Refine Expression Language (GREL)

No syntax error.

row	value	value.replace(/(\d{4})-\d{4}, (\d{4})/, '$1/$2')
599.	1905-1938, 1947	1905/1947
834.	1976-1977, 1983	1976/1983
925.	1948-1959, 1974	1948/1974
945.	1990-1991, 1993	1990/1993
1022.	1999-2000, 2002	1999/2002
1112.	1988-1996, 2001	1988/2001

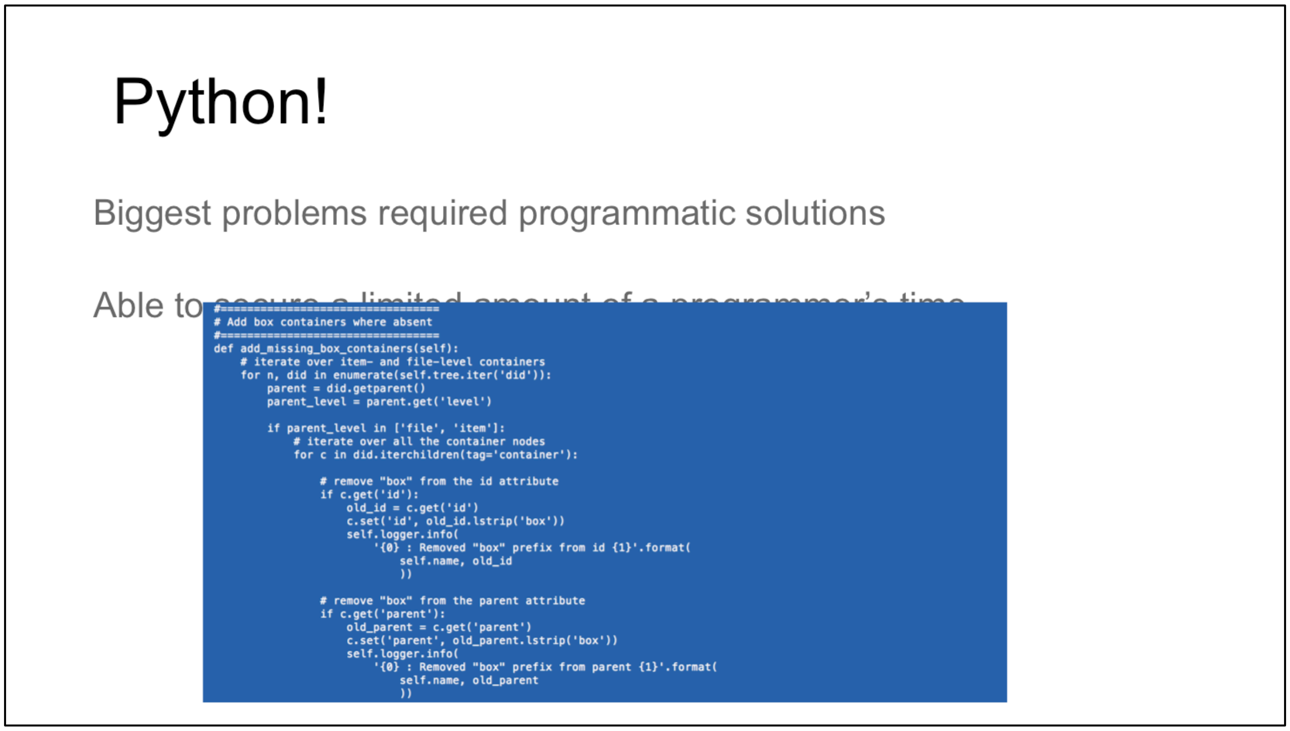
```
value.replace(/(\d{4})-\d{4}, (\d{4})/, '$1/$2')
```



Normalized > **200,000** date statements!

```
<unitdate normal="1848/1922" type="inclusive">1848-1922</unitdate>
<unitdate normal="1886/1909" type="bulk">1886-1909</unitdate>
```

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



Python!

Biggest problems required programmatic solutions

Able to secure a limited amount of a programmer's time

```
#==================================
# Add box containers where absent
#==================================
def add_missing_box_containers(self):
    # iterate over item- and file-level containers
    for n, did in enumerate(self.tree.iter('did')):
        parent = did.getparent()
        parent_level = parent.get('level')

        if parent_level in ['file', 'item']:
            # iterate over all the container nodes
            for c in did.iterchildren(tag='container'):

                # remove "box" from the id attribute
                if c.get('id'):
                    old_id = c.get('id')
                    c.set('id', old_id.lstrip('box'))
                    self.logger.info(
                        '{0} : Removed "box" prefix from id {1}'.format(
                            self.name, old_id
                            ))

                # remove "box" from the parent attribute
                if c.get('parent'):
                    old_parent = c.get('parent')
                    c.set('parent', old_parent.lstrip('box'))
                    self.logger.info(
                        '{0} : Removed "box" prefix from parent {1}'.format(
                            self.name, old_parent
                            ))
```

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 trie to get as much as possible - but there's not always much time.

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

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...

So we’re done right?

Nope.

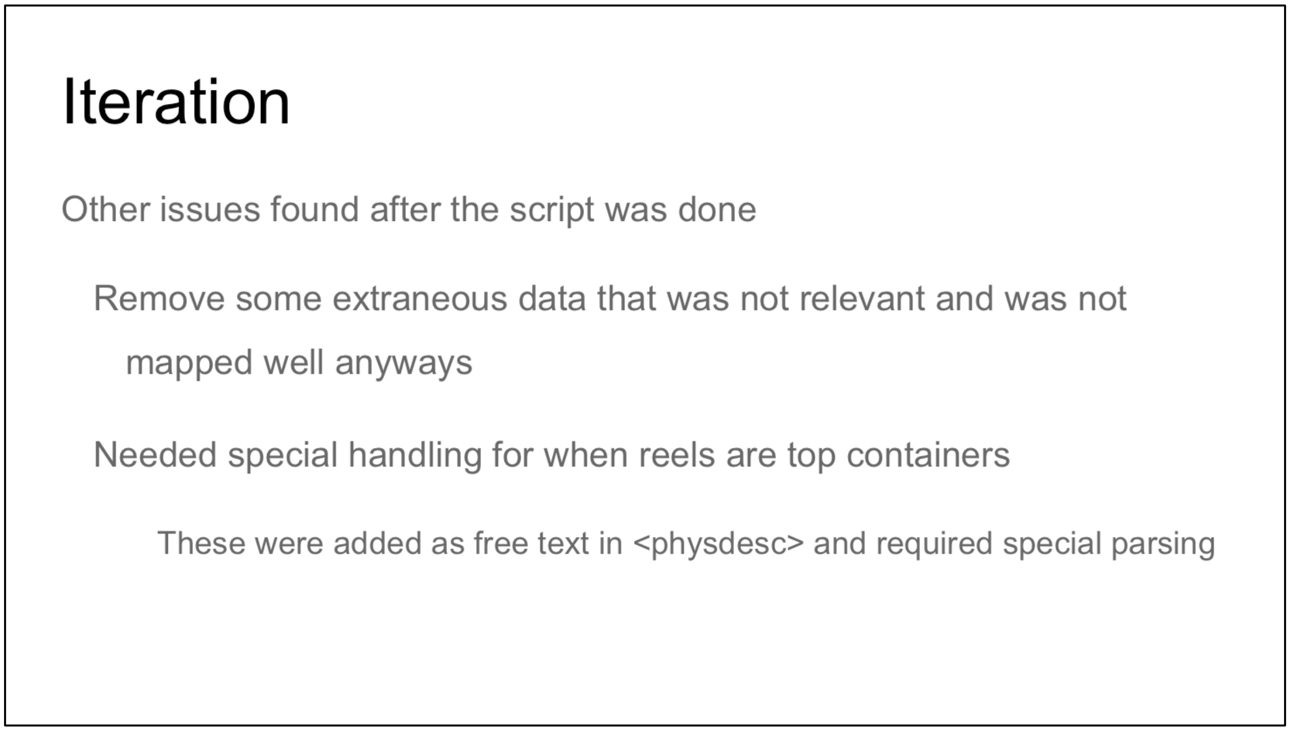
Yeah, not quite

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)

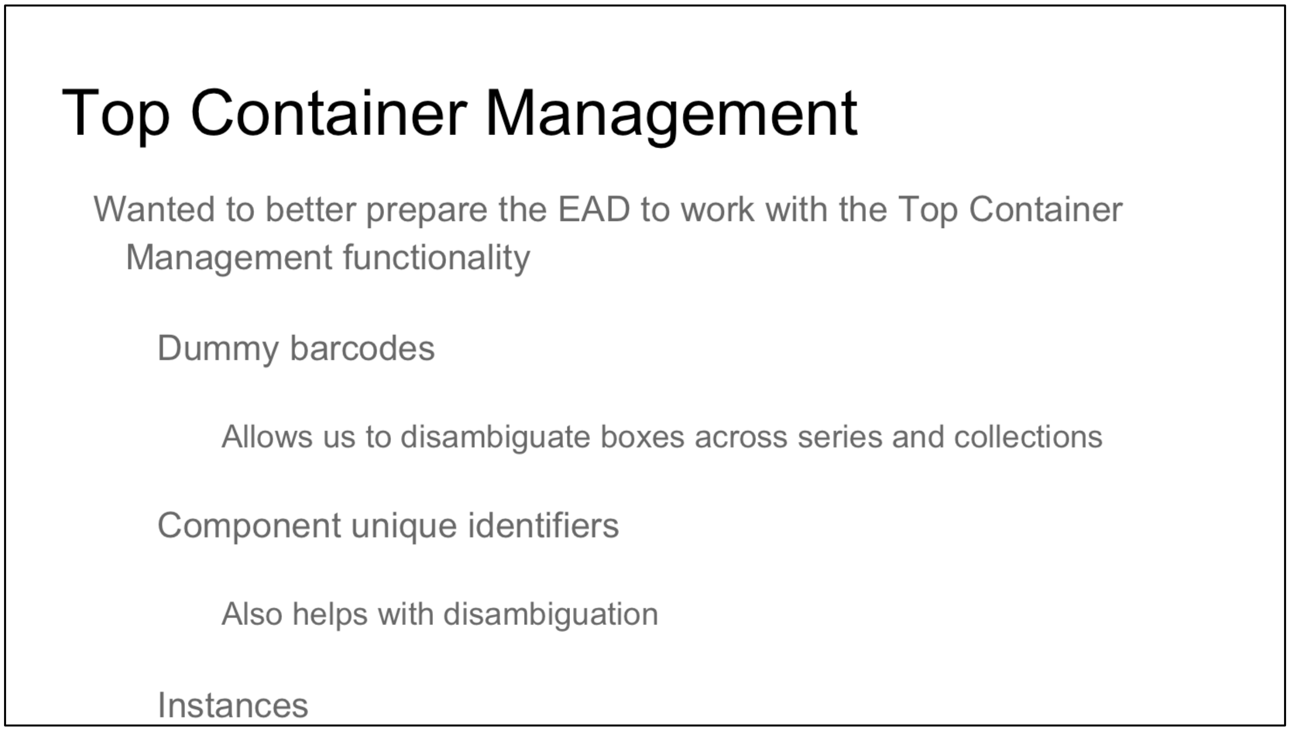


Iteration

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

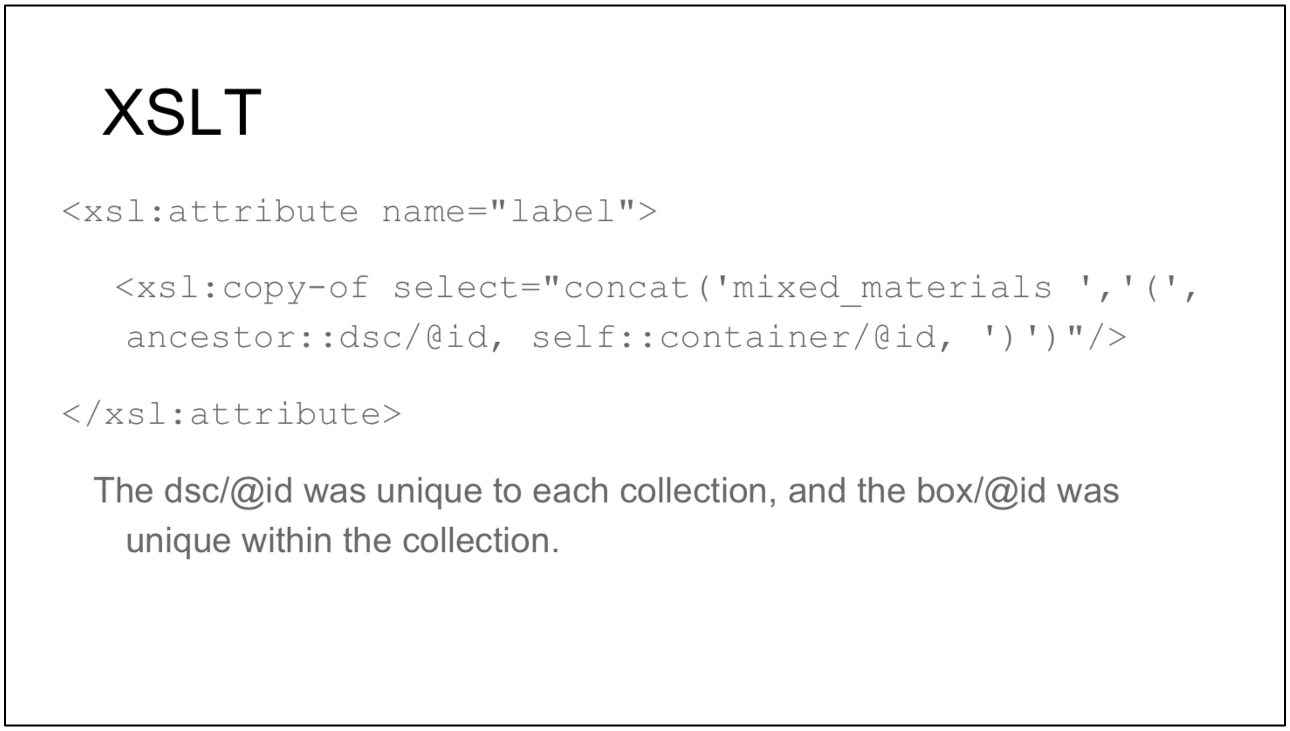
Once we were able to fix the EAD enough to use the importer, we were able to uncover other problems. - so we found that what we had in physloc did not map well at all, and we wouldn’t need it anyways since we’d use the location stuff in archivesspace. We also found collections that had their own quirks 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?



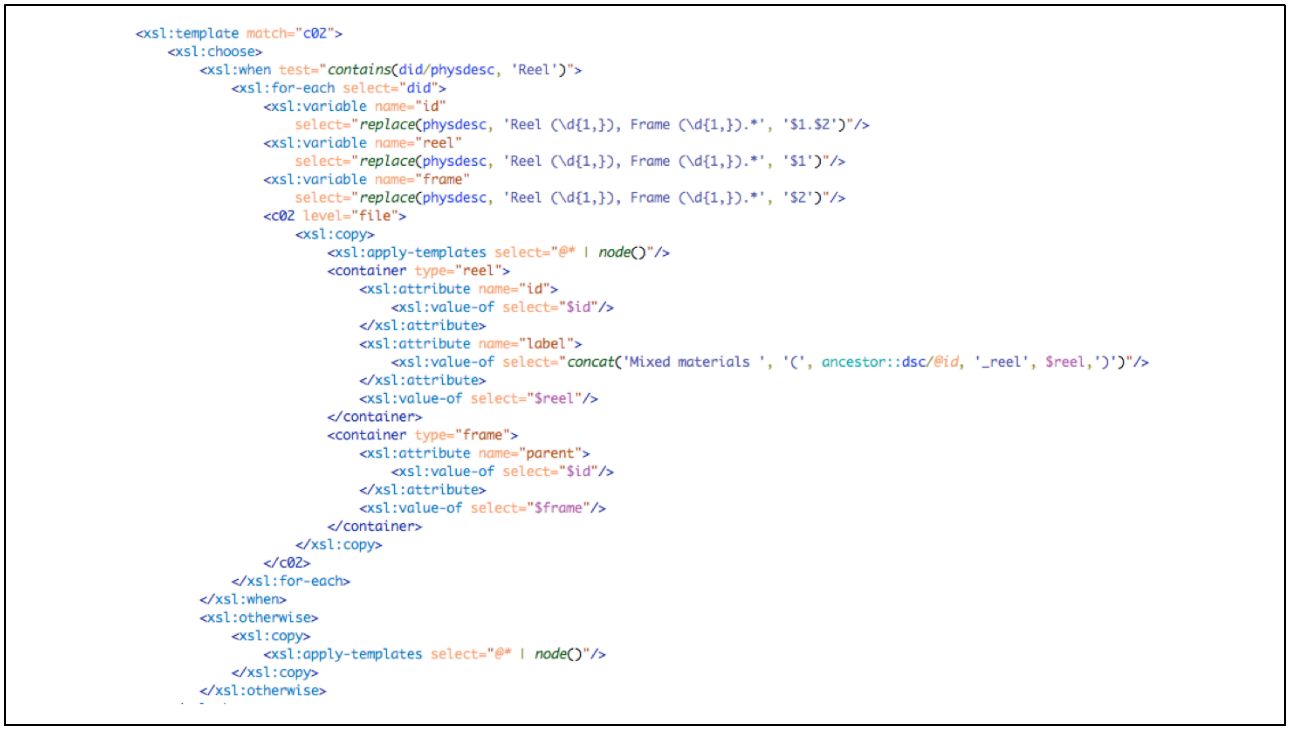
XSLT

```
<xsl:attribute name="label">

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

</xsl:attribute>
```

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

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).

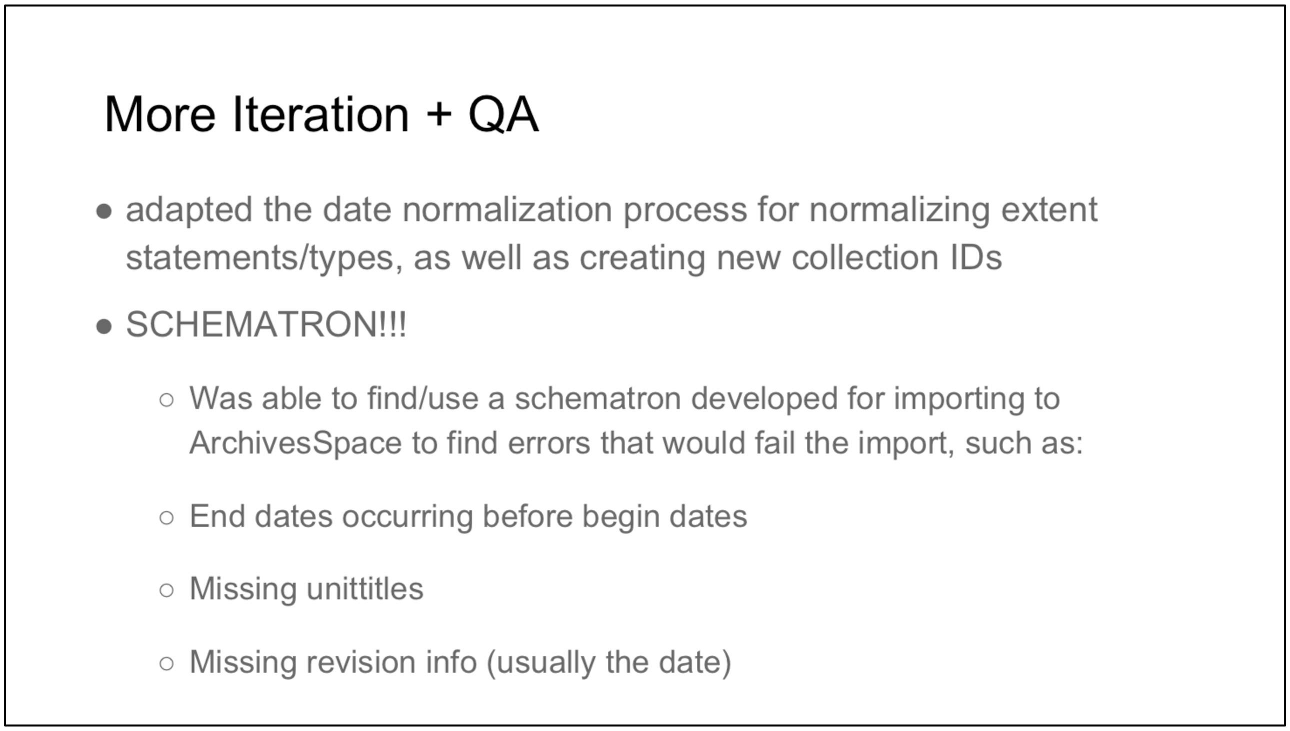
```
<xsl:copy-of
 select="concat('Mixed
 materials ', '(',
 ancestor::dsc/@id,
 self::container/@id, ')')"/>
```



```xml
<xsl:template match="c02">
    <xsl:choose>
        <xsl:when test="contains(did/physdesc, 'Reel')">
            <xsl:for-each select="did">
                <xsl:variable name="id"
                    select="replace(physdesc, 'Reel (\d{1,}), Frame (\d{1,}).*', '$1.$2')"/>
                <xsl:variable name="reel"
                    select="replace(physdesc, 'Reel (\d{1,}), Frame (\d{1,}).*', '$1')"/>
                <xsl:variable name="frame"
                    select="replace(physdesc, 'Reel (\d{1,}), Frame (\d{1,}).*', '$2')"/>
                <c02 level="file">
                    <xsl:copy>
                        <xsl:apply-templates select="@* | node()"/>
                        <container type="reel">
                            <xsl:attribute name="id">
                                <xsl:value-of select="$id"/>
                            </xsl:attribute>
                            <xsl:attribute name="label">
                                <xsl:value-of select="concat('Mixed materials ', '(', ancestor::dsc/@id, '_reel', $reel,')')"/>
                            </xsl:attribute>
                            <xsl:value-of select="$reel"/>
                        </container>
                        <container type="frame">
                            <xsl:attribute name="parent">
                                <xsl:value-of select="$id"/>
                            </xsl:attribute>
                            <xsl:value-of select="$frame"/>
                        </container>
                    </xsl:copy>
                </c02>
            </xsl:for-each>
        </xsl:when>
        <xsl:otherwise>
            <xsl:copy>
                <xsl:apply-templates select="@* | node()"/>
            </xsl:copy>
        </xsl:otherwise>
```

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.

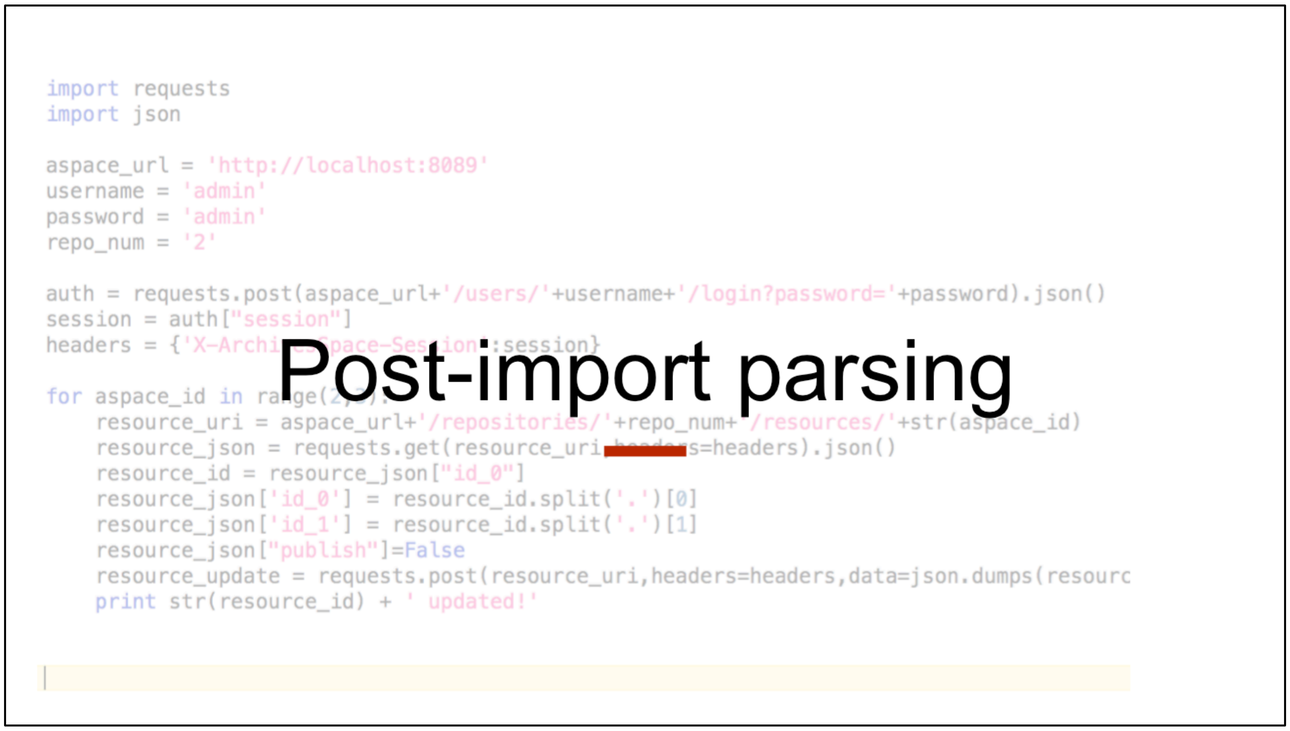


More Iteration + QA

- 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!)

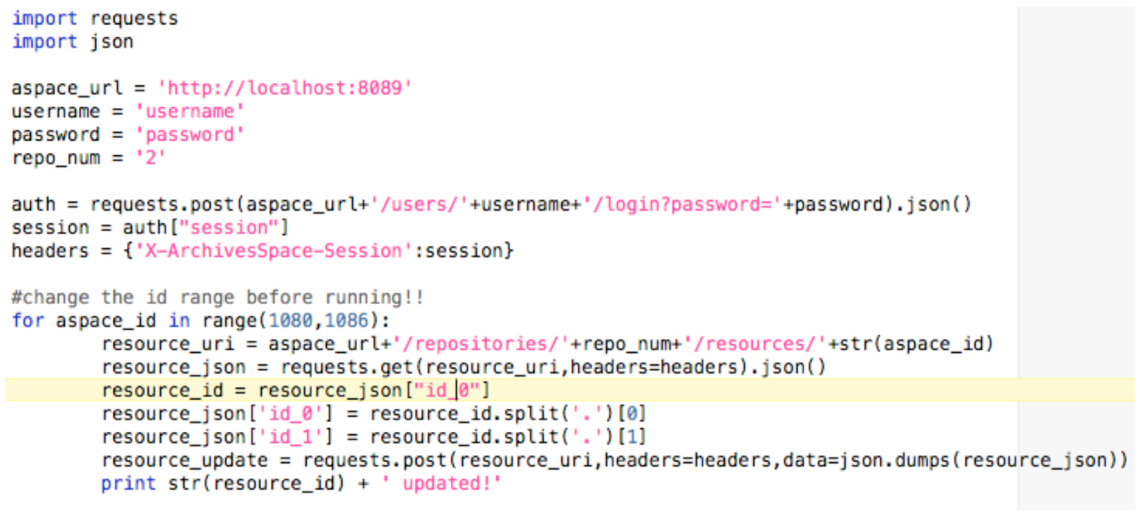


Post-import parsing

```
import requests
import json

aspace_url = 'http://localhost:8089'
username = 'admin'
password = 'admin'
repo_num = '2'

auth = requests.post(aspace_url+'/users/'+username+'/login?password='+password).json()
session = auth["session"]
headers = {'X-ArchivesSpace-Session':session}

for aspace_id in range(2,3):
    resource_uri = aspace_url+'/repositories/'+repo_num+'/resources/'+str(aspace_id)
    resource_json = requests.get(resource_uri,headers=headers).json()
    resource_id = resource_json["id_0"]
    resource_json['id_0'] = resource_id.split('.')[0]
    resource_json['id_1'] = resource_id.split('.')[1]
    resource_json["publish"]=False
    resource_update = requests.post(resource_uri,headers=headers,data=json.dumps(resourc
    print str(resource_id) + ' updated!'
```

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



```
import requests
import json

aspace_url = 'http://localhost:8089'
username = 'username'
password = 'password'
repo_num = '2'

auth = requests.post(aspace_url+'/users/'+username+'/login?password='+password).json()
session = auth["session"]
headers = {'X-ArchivesSpace-Session':session}

#change the id range before running!!
for aspace_id in range(1080,1086):
        resource_uri = aspace_url+'/repositories/'+repo_num+'/resources/'+str(aspace_id)
        resource_json = requests.get(resource_uri,headers=headers).json()
        resource_id = resource_json["id_0"]
        resource_json['id_0'] = resource_id.split('.')[0]
        resource_json['id_1'] = resource_id.split('.')[1]
        resource_update = requests.post(resource_uri,headers=headers,data=json.dumps(resource_json))
        print str(resource_id) + ' updated!'
```

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!).

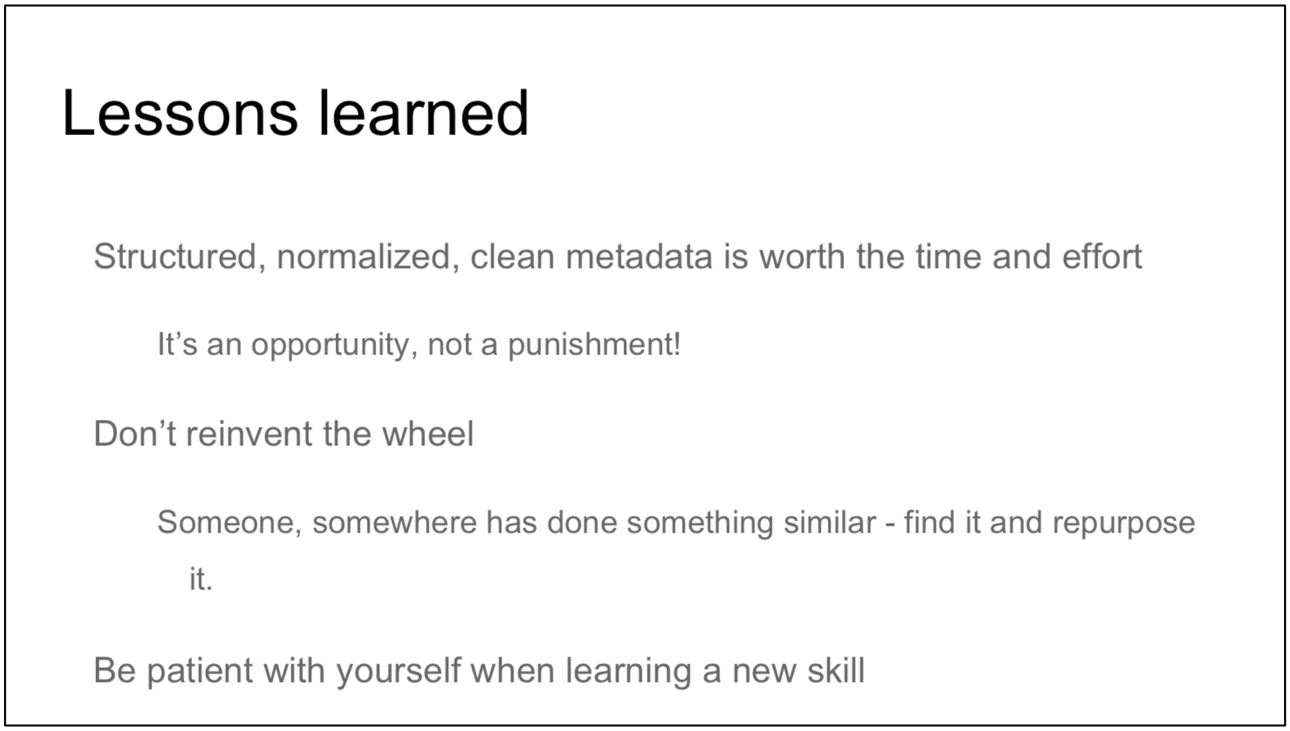
APIs are powerful!

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