Enhancing Metadata though Standardization and Validation

Practical Application at the University of Kansas Libraries

Erin Wolfe
Metadata Librarian
University of Kansas
edw@ku.edu

October 26, 2017
DC-2017, Washington, D.C.

Project Overview

Image: CC0

Disclaimer

Wabi-sabi

> “Nothing lasts,
> nothing is finished,
> and nothing is perfect.”
>
> - Richard Powell

System design

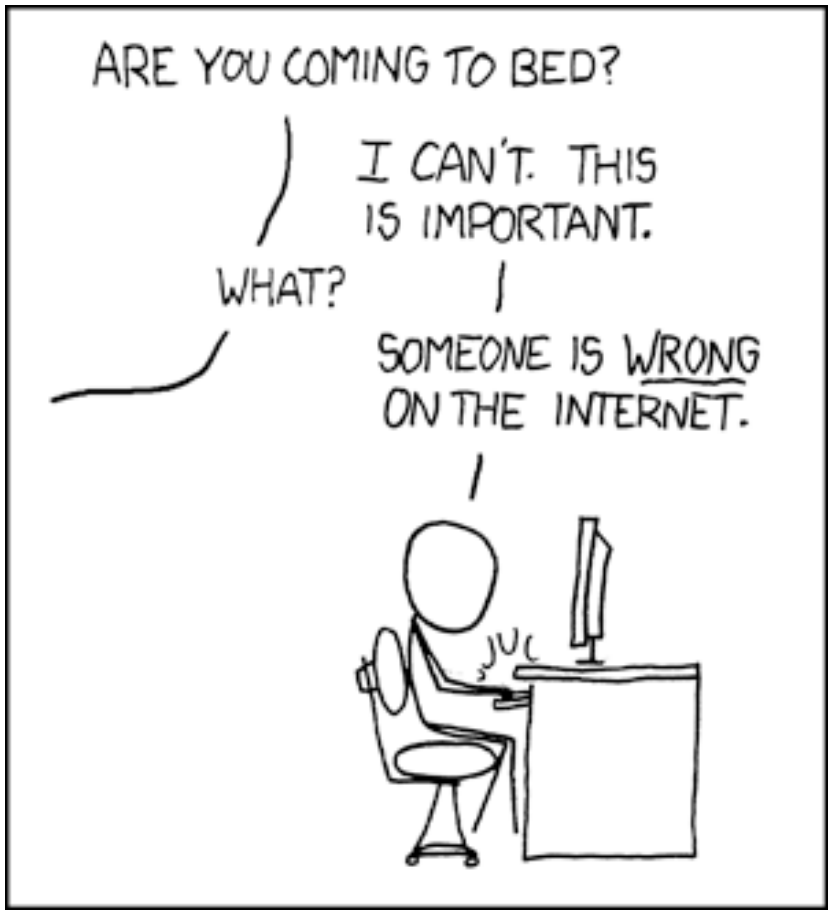
> “Make it work.
> Make it right.
> Make it fast.”
>
> - Kent Beck

http://bit.ly/mdprocess

Disclaimer

Cunningham's Law

> "The best way to get the right answer on the internet is not to ask a question; it's to post the wrong answer."
>
> - Ward Cunningham



"Duty Calls"
https://xkcd.com/386/ (CC BY-NC 2.5)

http://bit.ly/mdprocess

Where to start?

“Begin at the beginning," the King said, very gravely,
"and go on till you come to the end: then stop.”

— Lewis Carroll, Alice in Wonderland

What is the goal?

- Consistent format
- Standardized record
- As complete as possible
    - All known metadata
    - Enhanced when possible
- Record linked to object
- Well-formed and valid records
- Repeatable process

What is the goal?

- Consistent format
- Standardized record
- As complete as possible
    - All known metadata
    - Enhanced when possible
- Record linked to object
- Well-formed and valid records
- Repeatable process

"When I was growing up, I always wanted to be someone.

Now I realize I should have been more specific."

— Lily Tomlin

Define end goal first

Title	Required
Rights	
Identifier (filename)	
Institutional Identifier	
Record Identifier	
Name	Recommended
Date	
Subject	
Language (if Type not ‘still image’)	
Type	
Form	
Genre	
Extent	
Internet Media Type	

Define end goal first

Title
Rights
Identifier (filename)
Institutional Identifier
Record Identifier
Name
Date
Subject
Language (if Type not ‘still image’)
Type
Form
Genre
Extent
Internet Media Type

Required

Recommended

MODS XML

Digital Library Federation / Aquifer Implementation Guidelines for Shareable MODS Records

Define end goal first

Title	<mods:titleInfo><mods:title>
Rights	<mods:accessCondition>
Identifier (filename)	<mods:identifier type="local">
Institutional Identifier	<mods:relatedItem type="host"> <mods:location><mods:physicalLocation>
Record Identifier	<mods:recordInfo><mods:recordIdentifier>
Name	<mods:name><mods:namePart>
Date	<mods:originInfo><dateSubelement>
Subject	<mods:subject><subjectSubelement>
Language (if Type not ‘still image’)	<mods:language>

Define end goal first

<mods:titleInfo><mods:title>	<mods:titleInfo><mods:title>
<mods:accessCondition>	<mods:accessCondition type="use and reproduction” xlink:href="https://creativecommons.org/publicdomain/mark/1.0/"> This work is free of known copyright restrictions.
<mods:identifier type="local">	<mods:identifier type="local">
<mods:relatedItem type="host"> <mods:location> <mods:physicalLocation>	<mods:relatedItem type="host"> <mods:location> <mods:physicalLocation authority="fast" valueURI="http://id.worldcat.org/fast/530405"> University of Kansas
<mods:recordInfo> <mods:recordIdentifier>	<mods:recordInfo><mods:recordIdentifier>
<mods:name><mods:namePart>	<mods:name @type=“”><mods:namePart>
<mods:originInfo> <dateSubelement>	<mods:originInfo><dateSubelement @encoding=“w3cdtf”>

Where to start?

“Begin at the beginning," the King said, very gravely,
"and go on till you come to the end: then stop.”

— Lewis Carroll, Alice in Wonderland

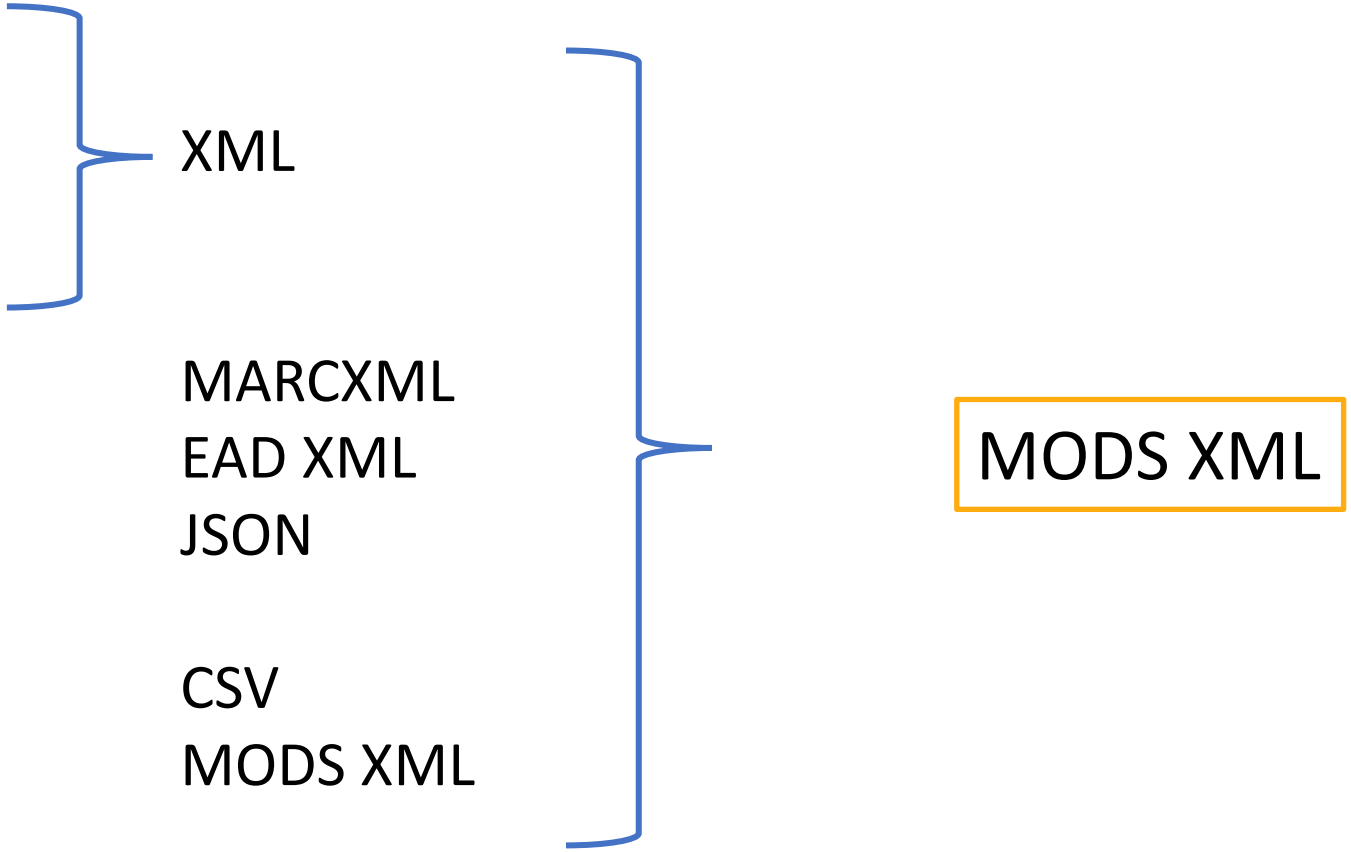
The beginning…

- Luna Insight
    - No rules!
    - Customized MODS
    - VRACore
    - Dublin Core
    - Local schema
- MARC
- ArchivesSpace
- DSpace
- Local storage
    - Structured file
    - MODS XML

The beginning…

…The End

- Luna Insight
    - No rules!
    - Customized MODS
    - VRACore
    - Dublin Core
    - Local schema
- MARC
- ArchivesSpace
- DSpace
- Local storage
    - Structured file
    - MODS XML



1) Create MODS

Existing record - Luna XML

```
<?xml version="1.0" encoding="UTF-8"?>
  <record>
    <entity name="title">
      <field name="title">
        <value>Abilene, Kansas : 1884</value>
      </field>
    </entity>
    <entity name="detail">
      <field name="description">
        <value>sheet number: 1</value>
      </field>
    </entity>
```

1) Create MODS

Luna XML (Sanborn) Entity/Field	MODS (Sanborn)	MODS (Sanborn expanded)
title/title	<titleInfo><title>	<titleInfo><title>
detail/description	<abstract>	<abstract>
creator/name	<name><namePart>	<name><namePart type="corporate">
		|--<role><roleTerm type="text" authority="marcrelator">publisher
		|--<role><roleTerm type="code" authority="marcrelator">pbl
type_of_resource/type	<typeOfResource>cartographic	<typeOfResource>cartographic
genre/genres	<genre>	<genre>map
origination_information/publisher	<originInfo><publisher>	<originInfo><publisher>
origination_information/location	|--<place><placeTerm>	|--<place><placeTerm type="text">
		|--|--<placeTerm type="code" authority="marccountry">nyu

1) Create MODS

Existing record - Luna XML

```
<?xml version="1.0" encoding="UTF-8"?>
  <record>
    <entity name="title">
      <field name="title">
        <value>Abilene, Kansas : 1884</value>
      </field>
    </entity>
    <entity name="deta
      <field name="des
        <value>sheet n
      </field>
    </entity>
```

XSL stylesheets

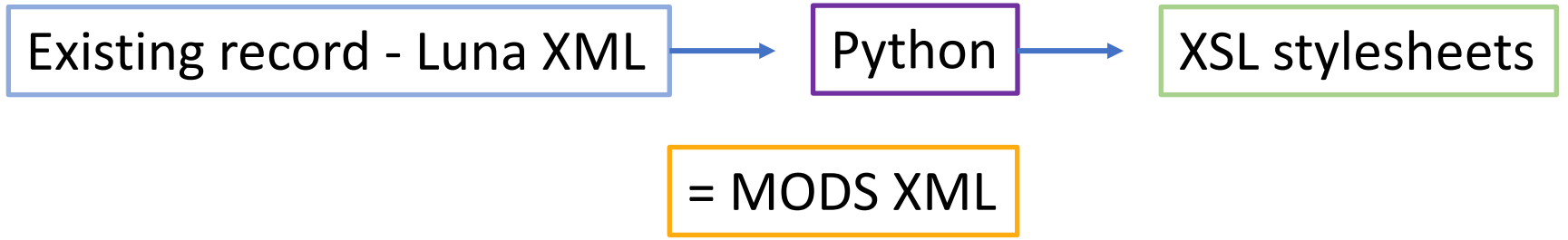
```
<xsl:for-each select="//record">
    <mods:mods xmlns:xsi="http://www.loc.gov/mods/v3"
        xsi:schemaLocation="http://www.loc.gov/standards/mods/v3/mods-3-6.xsd">
    <xsl:if test="entity[@name='title']">
        <mods:titleInfo>
        <mods:title>
        <xsl:value-of select="entity[@name='title']/field[@name='title']/value"/>
        </mods:title>
    </mods:titleInfo>
    </xsl:if>
```

1) Create MODS

Existing record - Luna XML → Python → XSL stylesheets

```
xsltree = ET.parse(transformfile)
transform = ET.XSLT(xsltree)

if not os.path.isdir(outdir):
    os.mkdir(outdir)

tree = ET.parse(infile)
root = tree.getroot()

for record in root.findall('record'):
    outfile = record.find("field[@name='identifier']/value").text
    outfile += '_m.xml'
    outfile = os.path.join(outdir, outfile)

    outtree = transform(record)
    outtree.write(outfile,pretty_print=True,encoding='UTF-8')
```

1) Create MODS




```
<?xml version="1.0" encoding="UTF-8"?>
<mods:mods xmlns:mods="http://www.loc.gov/mods/v3" xmlns:xsi="http://www.loc.gov/mods/v
   <mods:titleInfo>
      <mods:title>Abilene, Kansas : 1884</mods:title>
   </mods:titleInfo>
   <mods:abstract>sheet number: 1 (from a set of 4 sheets)</mods:abstract>
   <mods:name type="corporate">
      <mods:namePart>Sanborn Map Company</mods:namePart>
      <mods:role>
         <mods:roleTerm type="text" authority="marcrelator">publisher</mods:roleTerm>
      </mods:role>
```

Repeatable, sort of…

“No man ever steps in the same river twice,
for it's not the same river
and he's not the same man.”

— Heraclitus

2) Add elements

- Identifiers
  - Filename(s)
  - Record identifier
  - Institutional ID
- Access conditions
- Handles / Persistent URI
- Any other element that can be added systematically

2) Add elements

```
<mods:relatedItem type="host">
  <mods:location>
    <mods:physicalLocation authority="fast"
                           valueURI="http://id.worldcat.org/fast/530405">
      University of Kansas
    </mods:physicalLocation>
  </mods:location>
</mods:relatedItem>
```

```
for file in glob.glob('*.xml'):

    parser = etree.XMLParser(remove_blank_text=True)
    tree = etree.parse(file, parser)
    root = tree.getroot()

    related = etree.SubElement(root, '{http://www.loc.gov/mods/v3}relatedItem')
    related.attrib['type'] = "host"
    location = etree.SubElement(related, '{http://www.loc.gov/mods/v3}location')
    physicalLoc = etree.SubElement(location, '{http://www.loc.gov/mods/v3}physicalLocation')
    physicalLoc.attrib['authority'] = "fast"
    physicalLoc.attrib['valueURI'] = "http://id.worldcat.org/fast/530405"
    physicalLoc.text = "University of Kansas"

    tree.write(file, encoding='UTF-8', pretty_print=True, xml_declaration=True)
```

2) Add elements

< xsl: >

Namespaces

```
<xsl:template match="mods:mods">
    <mods:mods
        xmlns:mods="http://www.loc.gov/mods/v3"
        xmlns:xlink="http://www.w3.org/1999/xlink"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.loc.gov/standards/mods/v3/mods-3-5.xsd"
        version="3.5">

        <xsl:apply-templates select="@*|node()"/>
    </mods:mods>
</xsl:template>
```

3) Linked subject headings

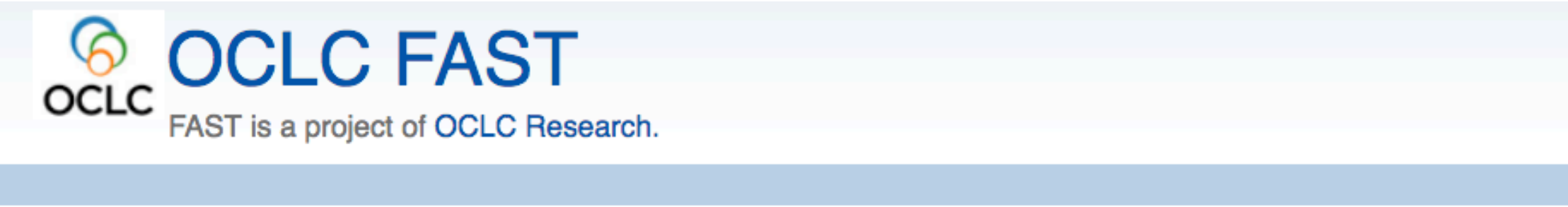
Python

Open Refine

XSL stylesheets

< xsl: >

3) Linked subject headings



FAST (Faceted Application of Subject Terminology)

FAST (Faceted Application of Subject Terminology) is an enumerative, faceted subject heading schema derived from the Library of Congress Subject Headings (LCSH). The purpose of adapting the LCSH with a simplified syntax to create FAST is to retain the very rich vocabulary of LCSH while making the schema easier to understand, control, apply, and use. The schema maintains upward compatibility with LCSH, and any valid set of LC subject headings can be converted to FAST headings. The FAST Authority file contains links to LCSH Authorities as will as other authoritative sources such as VIAF, GeoNames, and Wikipedia. We will continue to add other links where possible.

http://fast.oclc.org/

3) Linked subject headings

README.md

An OpenRefine reconciliation service for FAST.

> FAST is available as Linked Data, which is an approach to publishing data which enhances the utility of information on the web by making references to persons, places, things, etc. more consistent and linkable across domains.

The service queries the FAST AutoSuggest API and provides normalized scores across queries for reconciling in Refine.

Run locally as:

```
$ python reconcile.py --debug
```

https://github.com/lawlesst/fast-reconcile

3) Linked subject headings

Extract subject headings from XML

1

MODS XML

```
<mods:subject>
  <mods:topic>Kansas, Larned</mods:topic>
</mods:subject>
```

2

XSL stylesheets

```
<!-- exclude if fast authority and link are already present -->
<xsl:if test="not(@authority = 'fast') or not(@valueURI)">

        <xsl:value-of select="normalize-space(.)"/>
        <xsl:value-of select="$tab"/>
        <xsl:value-of select="local-name()"/> <!-- type of heading -->
        <xsl:value-of select="$tab"/>
        <xsl:value-of select="local-name(parent::*)"/>
        <xsl:value-of select="$tab"/>
        <xsl:value-of select="$identifier"/>
        <xsl:value-of select="$newline"/>
</xsl:if>
```

3

TSV file

```
Kansas, Larned  topic   subject ksrl.kc.sm_larned_1884_001
Larned Pawnee County Kansas United States   hierarchicalGeographic  subject ksrl.kc.sm_larned_1884_001
Larned  city    hierarchicalGeographic  ksrl.kc.sm_larned_1884_001
Pawnee County   county  hierarchicalGeographic  ksrl.kc.sm_larned_1884_001
Kansas  state   hierarchicalGeographic  ksrl.kc.sm_larned_1884_001
United States   country hierarchicalGeographic  ksrl.kc.sm_larned_1884_001
```

3) Linked subject headings

32016 rows

Show as: **rows** records Show: 5 10 **25** 50 rows

All	original_term	term_type	eadid_term_source
1.	Kansas--Abilene	topic	ksrl.kc.sm_abilen_1884_001
2.	Abilene Dickinson County Kansas United States	hierarchicalGeographic	ksrl.kc.sm_abilen_1884_001
3.	Abilene	city	ksrl.kc.sm_abilen_1884_001
4.	Dickinson County	county	ksrl.kc.sm_abilen_1884_001
5.	Kansas	state	ksrl.kc.sm_abilen_1884_001
6.	United States	country	ksrl.kc.sm_abilen_1884_001
7.	Kansas--Abilene	topic	ksrl.kc.sm_abilen_1884_002
8.	Abilene Dickinson County Kansas United States	hierarchicalGeographic	ksrl.kc.sm_abilen_1884_002
9.	Abilene	city	ksrl.kc.sm_abilen_1884_002
10.	Dickinson County	county	ksrl.kc.sm_abilen_1884_002

3) Linked subject headings

	original_term	cleaned	term_type	eadid_term_source
1.	Kansas--Abilene	Kansas--Abilene	topic	ksrl.kc.sm_abilen_1884_001
2.	Abilene Dickinson County Kansas United States	Abilene Dickinson County Kansas United States	hierarchicalGeographic	ksrl.kc.sm_abilen_1884_001
3.	Abilene	Abilene	city	ksrl.kc.sm_abilen_1884_001
4.	Dickinson County	Dickinson County	county	ksrl.kc.sm_abilen_1884_001
5.	Kansas	Kansas	state	ksrl.kc.sm_abilen_1884_001
6.	United States	United States	country	ksrl.kc.sm_abilen_1884_001
7.	Kansas--Abilene	Kansas--Abilene	topic	ksrl.kc.sm_abilen_1884_002
8.	Abilene Dickinson County Kansas United States	Abilene Dickinson County Kansas United States	hierarchicalGeographic	ksrl.kc.sm_abilen_1884_002
9.	Abilene	Abilene	city	ksrl.kc.sm_abilen_1884_002
10.	Dickinson County	Dickinson County	county	ksrl.kc.sm_abilen_1884_002

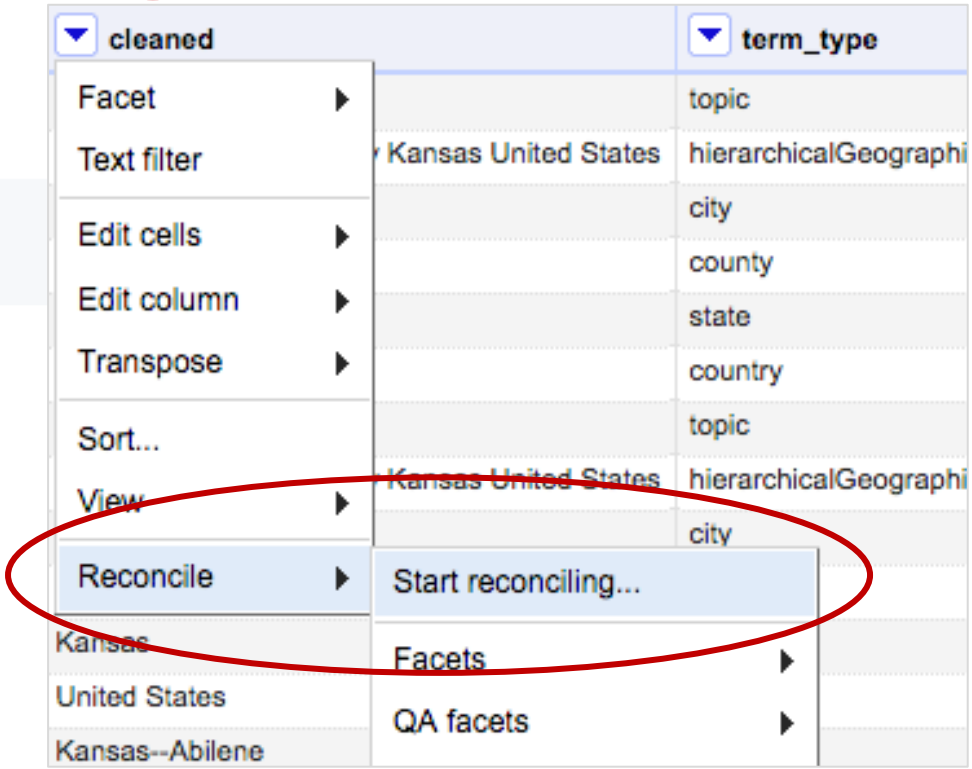
3) Linked subject headings

1

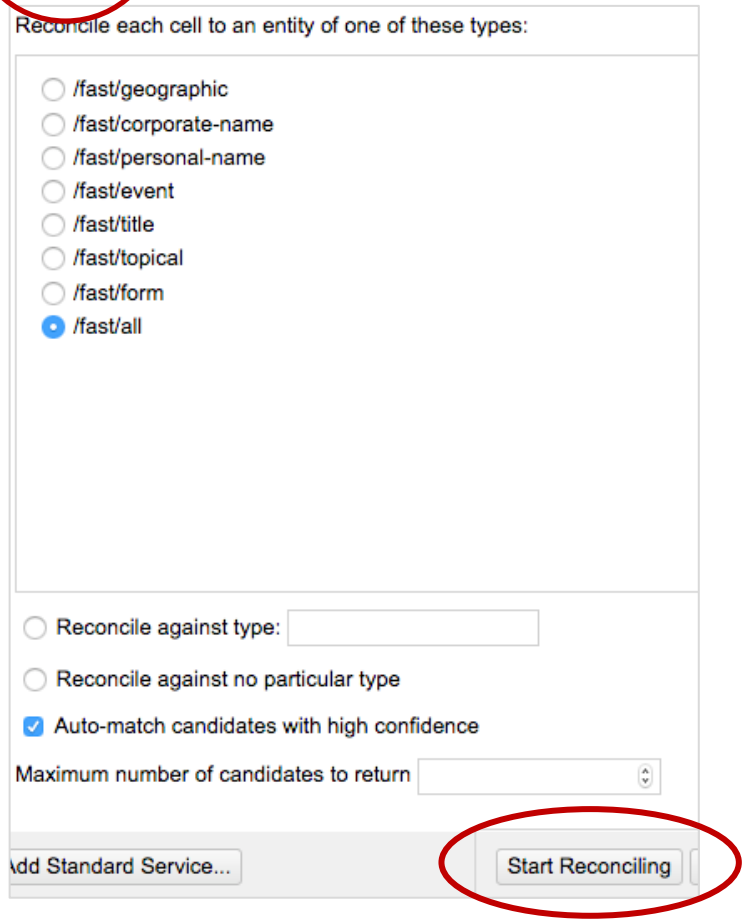
Run locally as:

```
$ python reconcile.py --debug
```

2



3



3) Linked subject headings

	original_term	cleaned
1.	Kansas--Abilene	Kansas--Abilene Choose new match
2.	Abilene Dickinson County Kansas United States	Abilene Dickinson County Kansas United States Create new topic
3.	Abilene	Abilene Texas--Abilene (70) Kansas--Abilene (67) Abilene State School (52) Create new topic
4.	Dickinson County	Dickinson County Iowa--Dickinson County (86) Kansas--Dickinson County (82) Michigan--Dickinson County (78) Create new topic

3) Linked subject headings

Put that Subject Heading back where it came from, or so help me…

a) Export from Open Refine
b) XSL stylesheet to get it to the right format
c) Python script to match on filename and subject

MODS XML

```
<mods:subject>
  <mods:topic authority="fast" valueURI="http://id.worldcat.org/fast/1008726">Maps
</mods:topic>
 </mods:subject>
 <mods:subject>
  <mods:topic authority="fast" valueURI="http://id.worldcat.org/fast/1218848">Kansas--Larned</mods:topic>
 </mods:subject>
```

3) Linked subject headings

ITIS

pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$

4) Verify filenames

```
for xmlfile in glob.glob(xmldir + '*.xml'):
    objXmlMatch = 0

    xmlfilename = os.path.splitext(os.path.basename(xmlfile))[0]

    for f in glob.glob(objdir + '**/*.*', recursive=True):

        objfilename = os.path.splitext(os.path.basename(f))[0]
        if xmlfilename == objfilename:

            objXmlMatch = objXmlMatch + 1

            break

    if objXmlMatch < 1:

        with open(errorfile, 'a') as f:

            f.write('XML file ' + xmlfilename + ' had no matching digital object\n')

if not os.path.exists(errorfile):
    print("All XML files have a matching digital object. No log created.")
else:
    print("Some discrepencies found. Log created at " + os.path.abspath(xmldir))
```

5) Validate records

1. Are they valid MODS?
2. Do they meet minimum requirements?
3. Do they meet recommended requirements?

Python

Schematron

5) Validate records

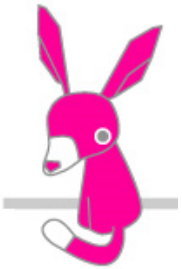
lxml

1. Are they valid MODS?

```python
# Validate mods against MODS Schema
def validate_mods_xsd(doc: str, mods_schema: str) -> bool:
    xmlschema_doc = etree.parse(mods_schema)
    xmlschema = etree.XMLSchema(xmlschema_doc)
    xml_doc = doc
    result = xmlschema.validate(xml_doc)

    if result:
        validate_mods_requirements()

    else:
        report_invalid = [file,current_time,'This MODS file is not valid against the MODS 3.6 Schema']
        invalid_output = os.path.join(outdir_csv, csv_invalid)

        with open(invalid_output, 'a') as csvfile:
            invalid_mods = csv.writer(csvfile)
            invalid_mods.writerow(report_invalid)

        # move invalid mods to subfolder for easier reprocessing
        if not os.path.isdir(error_dir):
            os.mkdir(error_dir)

        os.rename((os.path.join(mods_dir, file)), (os.path.join(error_dir, file)))
```

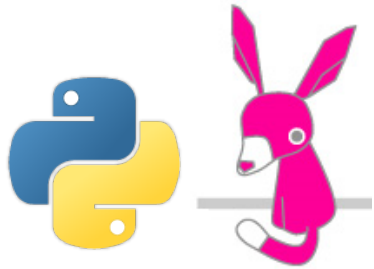
5) Validate records

1. Are they valid MODS?
2. Do they meet minimum requirements?
3. Do they meet recommended requirements?



```
<!--Validate that minimimum elements are present and have a value -->
<sch:pattern id="minimum-requirements">
    <sch:rule context="mods:mods" id="top-level-elements">
        <sch:report test="mods:titleInfo/mods:title[text()[normalize-space()]]">Passed: mods:title</sc
        <sch:assert test="mods:titleInfo/mods:title[text()[normalize-space()]]">Failed: title element
        <sch:report test="mods:accessCondition[text()[normalize-space()]]">Passed: mods:accessConditio
        <sch:assert test="mods:accessCondition[text()[normalize-space()]]">Failed: rights statement is
        <sch:report test="mods:identifier[@type='local'][text()[normalize-space()]]">Passed: mods:iden
        <sch:assert test="mods:identifier[@type='local'][text()[normalize-space()]]">Failed: identifie
        <sch:report test="mods:relatedItem/mods:location/mods:physicalLocation[@authority='fast'][@val
        <sch:assert test="mods:relatedItem/mods:location/mods:physicalLocation[@authority='fast'][@val
        <sch:report test="mods:recordInfo/mods:recordIdentifier[text()[normalize-space()]]">Passed: mo
        <sch:assert test="mods:recordInfo/mods:recordIdentifier[text()[normalize-space()]]">Failed: a
    </sch:rule>
</sch:pattern>
```

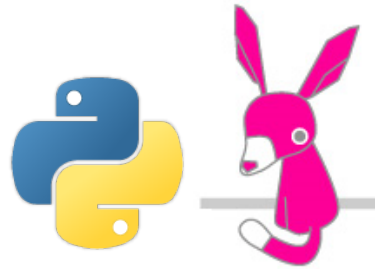
5) Validate records



1. Are they valid MODS?
2. Do they meet minimum requirements?
3. Do they meet recommended requirements?

```
def validate_mods_requirements():
    # Validate against minimum md schema
    schematron_min = isoschematron.Schematron(sct_min, store_report = True)
    validation_result_min = schematron_min.validate(doc)
    report_min = schematron_min.validation_report
    global csv_file

    if not validation_result_min:
        report_error = open(report_file, 'wb')
        report_min.write(report_error, pretty_print=True,xml_declaration=True,encoding='UTF-8')
        report_error.close()
        csv_file = csv_output_error
        transform()

    else:
        # Validate against recommended md schema
        schematron_rec = isoschematron.Schematron(sct_rec, store_report=True)
        validation_result_rec = schematron_rec.validate(doc)
        report_rec = schematron_rec.validation_report
        report = open(report_file, 'wb')
        report_rec.write(report,pretty_print=True,xml_declaration=True,encoding='UTF-8')
        report.close()
```

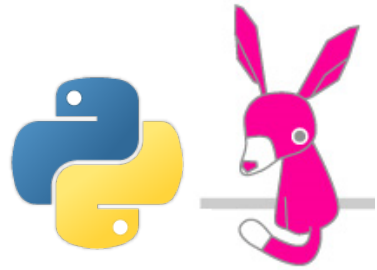
5) Validate records



1. Are they valid MODS?
2. Do they meet minimum requirements?
3. Do they meet recommended requirements?

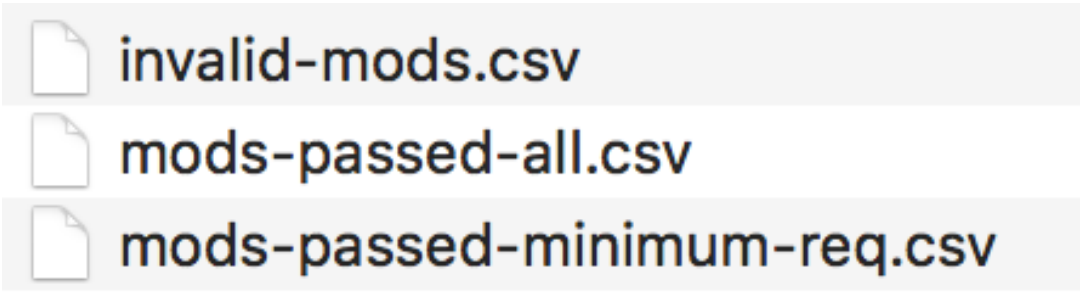
```
<svrl:successful-report test="mods:physicalDescription/mods:extent[text()
  <svrl:text>Passed: mods:physicalDescription/mods:extent</svrl:text>
</svrl:successful-report>
<svrl:failed-assert test="mods:physicalDescription/mods:internetMediaType
  <svrl:text>Missing recommended media type element - mods:physicalDescri
</svrl:failed-assert>
```

5) Validate records



1. Are they valid MODS?
2. Do they meet minimum requirements?
3. Do they meet recommended requirements?

```
<svrl:successful-report test="mods:physicalDescription/mods:extent[text()
  <svrl:text>Passed: mods:physicalDescription/mods:extent</svrl:text>
</svrl:successful-report>
<svrl:failed-assert test="mods:physicalDescription/mods:internetMediaType
  <svrl:text>Missing recommended media type element - mods:physicalDescri
</svrl:failed-assert>
```



6) Keep track

Erin Wolfe
Metadata Librarian
University of Kansas
edw@ku.edu

Thank you!

http://bit.ly/mdprocess