
Combine Documentation

Michigan State DPLA Service Hub

Oct 07, 2018

Contents

1	Overview	1
2	Installation	3
3	Table of Contents	5

CHAPTER 1

Overview

Combine is an application to facilitate the harvesting, transformation, analysis, and publishing of metadata records by Service Hubs for inclusion in the [Digital Public Library of America \(DPLA\)](#).

The name “Combine”, pronounced /käm̩bīn/ with a long i, is a nod to the [combine harvester used in farming](#) used for, “combining three separate harvesting operations - reaping, threshing, and winnowing - into a single process” Instead of grains, we have metadata records! These metadata records may come in a variety of metadata formats, various states of transformation, and may or may not be valid in the context of a particular data model. Like the combine equipment used for farming, this application is designed to provide a single point of interaction for multiple steps along the way of harvesting, transforming, and analyzing metadata in preparation for inclusion in DPLA.

CHAPTER 2

Installation

Combine has a fair amount of server components, dependencies, and configurations that must be in place to work, as it leverages [Apache Spark](#), among other applications, for processing on the backend.

To this end, a separate GitHub repository, [Combine-playbook](#), has been created to assist with provisioning a server with everything necessary, and in place, to run Combine. This repository provides routes for server provisioning via [Vagrant](#) and/or [Ansible](#). Please visit the [Combine-playbook](#) repository for more information about installation.

Table of Contents

If you just want to kick the tires, the [QuickStart guide](#) provides a walkthrough of harvesting, transforming, and publishing some records, that lays the groundwork for more advanced analysis.

3.1 QuickStart

3.1.1 Notes and Update

- This QuickStart guide provides a high level walkthrough of harvesting records from static files, transforming those records with XSLT, and publishing via Combine’s built-in OAI server.
- As of **9/20/2018**, with v0.3 on the horizon for release, this quickstart is becoming very outdated. Goal is to update soon, but in the interim, proceed at your own peril!

3.1.2 Overview

Sometimes you can’t beat kicking the tires to see how an application works. This “QuickStart” guide will walkthrough the harvesting, transforming, and publishing of metadata records in Combine, with some detours for explanation.

Demo data from unit tests will be reused here to avoid the need to provide actual OAI-PMH endpoints, Transformation or Validation scenarios, or other configurations unique to a DPLA Service Hub.

This guide will walk through the following areas of Combine, and it’s recommended to do so in order:

- *sshing into server*
- *python environment*
- *starting / stopping Combine*
- *Combine data model*
 - Organizations, RecordGroups, Jobs, Records
- *configuration*

- setting up OAI-PMH endpoints
- creating Transformation Scenarios
- creating Validation Scenarios
- *harvesting Records*
- *transforming Records*
- *looking at Jobs and Records*
- *duplicating / merging Jobs*
- *publishing Records*
- *analysis jobs*
- *troubleshooting*

For simplicity's sake, we will assume Combine is installed on a server with the domain name of `combine`, though likely running at the IP `192.168.45.10`, which the Ansible/Vagrant install from [Combine-Playbook](#) defaults to. On most systems you can point that IP to a domain name like `combine` by modifying your `/etc/hosts` file on your local machine. **Note:** `combine` and `192.168.45.10` might be used interchangeably throughout.

3.1.3 SSHing into server

The most reliable way is to `ssh` in as the `combine` user (assuming server at `192.168.45.10`), password is also `combine`:

```
# username/password is combine/combine
ssh combine@192.168.45.10
```

You can also use Vagrant to `ssh` in, from the Vagrant directory on the host machine:

```
vagrant ssh
```

If using Vagrant to `ssh` in, you'll want to switch users and become `combine`, as most things are geared for that user.

3.1.4 Combine python environment

Combine runs in a [Miniconda](#) python environment, which can be activated from any filepath location by typing:

```
source activate combine
```

Note: Most commands in this QuickStart guide require you to be in this environment.

3.1.5 Starting / Stopping Combine

Gunicorn

For normal operations, Combine is run using [Supervisor](#), a python based application for running system processes. Specifically, it's running under the Python WSGI server [Gunicorn](#), under the supervisor program named `gunicorn`.

Start Combine:

```
sudo supervisorctl start gunicorn
```

Stop Combine:

```
sudo supervisorctl stop gunicorn
```

You can confirm that Combine is running by visiting <http://192.168.45.10/combine>, where you should be prompted to login with a username and password. For default/testing installations, you can use `combine / combine` for these credentials.

Django runserver

You can also run Combine via Django's built-in server.

Convenience script, from `/opt/combine`:

```
./runserver.sh
```

Or, you can run the Django command explicitly from `/opt/combine`:

```
./manage.py runserver --noreload 0.0.0.0:8000
```

You can confirm that Combine is running by visiting <http://192.168.45.10:8000/combine> (note the 8000 port number).

Livy Sessions

To run any Jobs, Combine relies on an active (idle) Apache Livy session. Livy is what makes running Spark jobs possible via the familiar request/response cycle of a Django application.

Currently, users are responsible for determining if the Livy session is ready, though there are plans to have this automatically handled.

To check and/or start a new Livy session, navigate to: <http://192.168.45.10/combine/system>. The important column is `status` which should read `idle`. If not, click `Stop` or `Remove` under the `actions` column, and once stopped, click the `start new session` link near the top. Takes anywhere from 10-20 seconds to become `idle`.

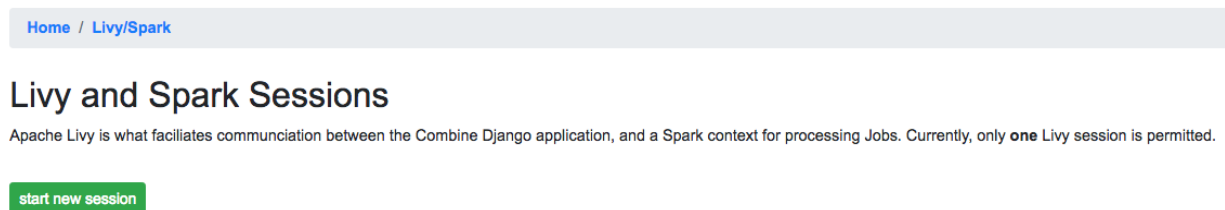


Fig. 1: Livy session page, with no active Livy session

You can check the status of the Livy session at a glance from the Combine navigation, where `Livy/Spark` next to `System` should have a green background if active.

3.1.6 Combine Data Model

Organization

The highest level of organization in Combine is an **Organization**. Organizations are intended to group and organize records at the level of an institution or organization, hence the name.

[Home](#) / [Livy/Spark](#)

Livy and Spark Sessions

Apache Livy is what facilitates communication between the Combine Django application, and a Spark context for processing Jobs. Currently, only **one** Livy session is permitted.

ID	Active	Name	Livy Session ID	Livy Session GUI	Status	Timestamp	Spark Session GUI	actions
62	True	Livy Session, sessionId 47	47	http://192.168.45.10:8998/ui/session/47	idle	April 12, 2018, 12:12 p.m.	http://192.168.45.10:4040	Stop

Fig. 2: Idle Livy session

You can create a new Organization from the Organizations page at [Organizations page](#), or by clicking “Organizations” from navigation links at the top of any page.

For this walkthrough, we can create one with the name “Amazing University”. Only the `name` field is required, others are optional.

RecordGroup

Within Organizations are **RecordGroups**. RecordGroups are a “bucket” at the level of a bunch of intellectually similar records. It is worth noting now that a single RecordGroup can contain multiple **Jobs**, whether they are failed or incomplete attempts, or across time. Suffice it to say for now that RecordGroups may contain lots of Jobs, which we will create here in a minute through harvests, transforms, etc.

For our example Organization, “Amazing University”, an example of a reasonable RecordGroup might be this fictional University’s Fedora Commons based digital repository. To create a new RecordGroup, from the [Organizations page](#), click on the Organization “Amazing University” from the table. From the following Organization page for “Amazing University” you can create a new RecordGroup. Let’s call it “Fedora Repository”; again, no other fields are required beyond `name`.

[Home](#) / [Organizations](#) / [Organization - Amazing University](#)

Organization: Amazing University

Record Groups

Record Group ID	Name	Description	Published?	Publish Set ID	Created	Actions
16	Fedora Repository	None	Not published		April 12, 2018, 12:14 p.m.	Delete

Create new Record Group

Name

Description

(optional)

Fig. 3: Demo Organization “Amazing University” and demo Record Group “Fedora Repository”

Finally, click into the newly created RecordGroup “Fedora Repository” to see the RecordGroup’s page, where we can begin to run Jobs.

Jobs

Central to Combine’s workflow philosophy are the ideas of **Jobs**. Jobs include any of the following:

- **Harvest** (OAI-PMH, static XML, and others to come)
- **Transform**
- **Merge/Duplicate**
- **Publish**
- **Analysis**

Within the context of a RecordGroup, one can think of Jobs as “stages” of a group of records, one Job serving as the *input* for the next Job run on those records. i.e.

```
OAI-PMH Harvest Job ---> XSLT Transform Job --> Publish Job
```

Record

Lastly, the most granular major entity in Combine is an individual **Record**. Records exist within a Job. When a Job is deleted, so are the Records (the same can be said for any of these hierarchies moving up). Records will be created in the course of running Jobs.

Briefly, Records are stored in MySQL, and are indexed in ElasticSearch. In MySQL, you will find the raw Record XML metadata, and other information related to the Record throughout various stages in Combine. In ElasticSearch, you find an flattened, indexed form of the Record’s *metadata*, but nothing much more. The representation of a Record in ElasticSearch is almost entirely for analysis and search, but the transactional nature of the Record through various stages and Jobs in Combine is the Record as stored in MySQL.

It is worth noting, though not dwelling on here, that groups of Records are also stored as Avro files on disk.

3.1.7 Configuration and Scenarios

Combine relies on users configuring “scenarios” that will be used for things like transformations, validations, etc. These can be viewed, modified, and tested in the Configuration page. This page includes the following main sections:

- [Field Mapper Configurations](#)
- [OAI-PMH endpoints](#)
- [Transformation Scenarios](#)
- [Validation Scenarios](#)
- [Record Identifier Transformation Scenarios](#)
- [DPLA Bulk Data Downloader](#)

For the sake of this QuickStart demo, we can bootstrap our instance of Combine with some demo configurations, creating the following:

- Transformation Scenario
 - “MODS to Service Hub profile” (XSLT transformation)
- Validation Scenarios
 - “DPLA minimum” (schematron validation)
 - “Date checker” (python validation)

To bootstrap these demo configurations for the purpose of this walkthrough, run the following command from `/opt/combine`:

```
./manage.py quickstartbootstrap
```

You can confirm these demo configurations were created by navigating to the configuration screen at <http://192.168.45.10/combine/configurations>.

3.1.8 Harvesting Records

Static XML harvest

Now we're ready to run our first Job and generate our first Records. For this QuickStart, as we have not yet configured any OAI-PMH endpoints, we can run a **static XML** harvest on some demo data included with Combine.

From the RecordGroup screen, near the bottom and under "Harvest", click "Static XML".

Run New Job

Harvest	Harvest Jobs are how Records are created and introduced to a Record Group. Currently this includes OAI-PMH harvesting, or ingesting Records from static files on disk.	<button>Harvest OAI-PMH</button> <button>Harvest Static XML</button>
Transform	Transformation Jobs change the Records within a Job some way. Currently this includes transformation of the Record's XML document with XSLT or python code snippets. Before running a Transformation, make a pre-configured Transformation Scenario exists.	<button>Transform Job</button>
Duplicate / Merge	Duplicate / Merge Jobs do not modify the Record, but can be used to combine multiple Jobs into one Job, run new or different Validation Scenarios on a pre-existing Job, or index to Elasticsearch with a different mapping.	<button>Duplicate / Merge Jobs</button>

Fig. 4: Area to initiate new Jobs from the Record Group page

You will be presented with a screen to run a harvest job of static XML files from disk:

Name

My Job Name

Optional

Notes

This job is particularly notable and/or interesting because...

Optional

The following are required parameters for this Job

X

Provide Static Content

For static harvests, optionally upload content, or provide a folder location on the filesystem where content can be found.

Note: only directories are allowed, all files in this location will be recursively found and harvested

X

Upload

Filesystem

Choose file...

Browse

Find and Parse XML Records

To locate and parse XML records from the provided file(s), additional information is needed: the root XML element for each record, and if needed, additional XML namespace declarations.

Root XML element	Because static XML harvests may be looping through many files, directories, archives, or single XML files, a root XML element is needed to locate XML documents that should be parsed and used. Unlike XPath, only the element name is needed. For example, a <code>mods:mods</code> would be sufficient for MODS records with a namespace, or <code>oai_dc</code> for OAI Dublin Core records.
Re-write XML root element declarations	Because Combine does not use XPath to parse the records, it is possible that records will be retrieved without required namespaces (often only declared at the root element). Here, you may optionally rewrite the declaration for each Record. For example, if providing the XML root <code>mods:mods</code> results in root XML nodes that look like <code><mods:mods></code> , the declaration <code>xmlns:mods="http://www.loc.gov/mods/v3"</code> could be passed and will be added to each record resulting in <code><mods:mods xmlns:mods="http://www.loc.gov/mods/v3"></code> , which is valid XML.

Root XML Element:

e.g. mods:mods, oai_dc, etc.

Re-write Root XML Declaration (if needed):

e.g. xmlns:mods="http://www.loc.gov/mods/v3"

Locate Identifier in Document

Finally, an optional XPath expression can be provided to locate a unique, meaningful identifier for each document.

Note: While not required, this is encouraged. In the absence of an XPath expression to locate an identifier, an MD5 hash of the document's contents will be created, which might help with potential duplicate records. However, if the document changes harvest-to-harvest, this identifier will not remain constant. Furthermore, this MD5 hash identifier will propagate through Combine and eventually be used for publishing, which is important to consider.

Description	Example
Use Dublin Core <code><dc:identifier></code> element.	<code>//dc:identifier</code>
Locate <code><mods:url></code> element with <code>access</code> attribute.	<code>//mods:mods/location/mods:url[@access]</code>

XPath for Record Identifier:

e.g. //dc:identifier

Fig. 5: Static Harvest Job screen

Many fields are optional – e.g. Name, Description – but we will need to tell the Harvest Job where to find the files.

First, click the tab “Filesystem”, then for the form field `Location of XML files on disk:`, enter the following, which points to a directory of 250 MODS files (this was created during bootstrapping):

```
/tmp/combine/qs/mods
```

Next, we need to provide an XPath query that locates each discrete record within the provided MODS file. Under the section “Locate Document”, for the form field `Root XML Element`, enter the following:

```
/mods:mods
```

For the time being, we can ignore the section “Locate Identifier in Document” which would allow us to find a unique identifier via XPath in the document. By default, it will assign a random identifier based on a hash of the document string.

Next, we can apply some optional parameters that are present for all jobs in Combine. This looks like the following:

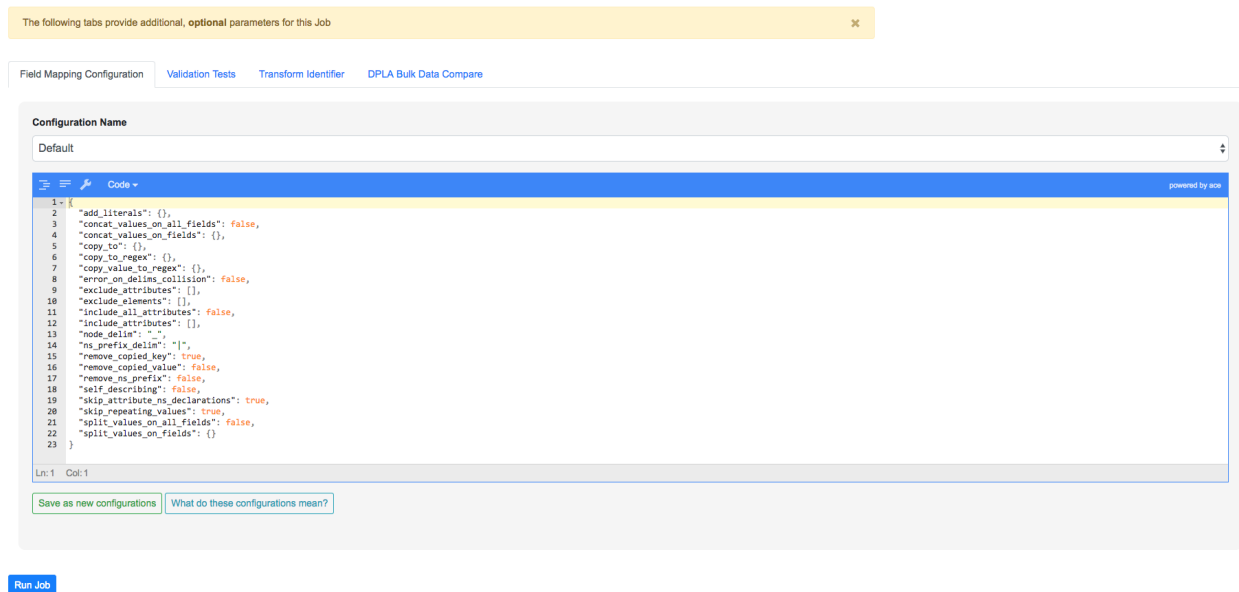


Fig. 6: Optional Job parameters

Different parameter types can be found under the various tabs, such as:

- Field Mapping Configuration
- Validation Tests
- Transform Identifier
- etc.



Most of these settings we can leave as default for now, but one optional parameter we’ll want to check and set for this initial job are Validations to perform on the records. These can be found under the “Validation Tests” tab. If you bootstrapped the demo configurations from steps above, you should see two options, *DPLA minimum* and *Date checker*; make sure both are checked.

Finally, click “Run Job” at the bottom.

This should return you to the RecordGroup page, where a new Job has appeared and is running under the Status column in the Job table. A static job of this size should not take long, refresh the page in 10-20 seconds, and hopefully,

you should see the Job status switch to `available`.

#205, HarvestStaticXMLJob @ Apr. 12, 2018, 12:26:28 PM

Show 100 entries Search:

	Job ID	Timestamp	Name	Record Group	Job Type	Status	Finished	Is Valid	Elapsed	Input	Notes	Total Record Count	Monitor	Actions
<input type="checkbox"/>	205	April 12, 2018, 12:26 p.m.	HarvestStaticXMLJob @ Apr. 12, 2018, 12:26:28 PM	Fedora Repository	HarvestJob	available	True	False	0:00:26 9.6 r/s	None	None	250	Livy / Spark	Details
				All	All	All	All	All						

Showing 1 to 1 of 1 entries Previous 1 Next

Fig. 7: Status of Static Harvest job, also showing Job failed some Validations

This table represents all Jobs run for this RecordGroup, and will grow as we run some more. You may also note that the `Is Valid` column is red and shows `False`, meaning some records have failed the Validation Scenarios optionally ran for this Job. We will return to this later.

For now, let's continue by running an XSLT Transformation on these records.

3.1.9 Transforming Records

In the previous step, we harvested 250 records from a bunch of static MODS XML documents. Now, we will transform all the Records in that Job with an XSLT Transformation Scenario.

From the RecordGroup screen, click the “Transform” link at the bottom.

For a Transform job, you are presented with other Jobs from this RecordGroup that will be used as an *input* job for this Transformation.

Again, `Job Name` and `Job Note` are both optional. What *is* required, is selecting what job will serve as the input Job for this Transformation. In Combine, most Jobs take a *previous* job as an input, essentially performing the current Job over all records from the previous job. In this way, as Records move through Jobs, you get a series of “stages” for each Record.

An input Job can be selected for this Transform Job by clicking the radio button next to the job in the table of Jobs (at this stage, we likely only have the one Harvest Job we just ran).

Search:

Select	Job ID	Name	Record Group	Job Type	Status	Finished	Is Valid	Timestamp	Input	Notes	Record Count
<input checked="" type="radio"/>	205	HarvestStaticXMLJob @ Apr. 12, 2018, 12:26:28 PM	Fedora Repository	HarvestJob	available	True	False	April 12, 2018, 12:26 p.m.		None	250
		Record Group	All	All	All	All					

Showing 1 to 1 of 1 entries

Select Transformation Scenario to use

MODS to Service Hub profile (xslt)

Fig. 8: Input Job selection screen

Next, we must select a **Transformation Scenario** to apply to the records from the input Job. We have a Transformation Scenario prepared for us from the QuickStart bootstrapping, but this is where you might optionally select different

transforms depending on your task at hand. While only one Transformation Scenario can be applied to a single Transform job, multiple Transformation Scenarios can be prepared and saved in advance for use by all users, ready for different needs.

For our purposes here, select `MODS to Service Hub profile (xslt)` from the dropdown:

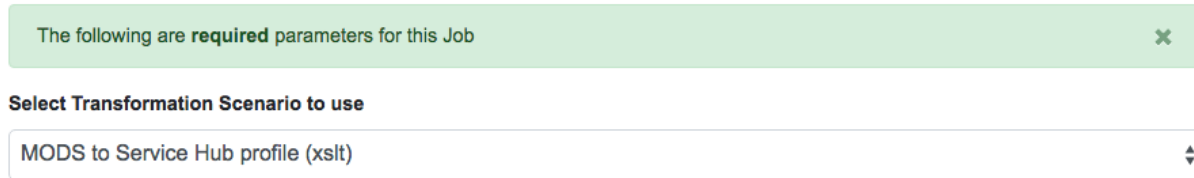


Fig. 9: Select Transformation Scenario to use

Once the input Job (radio button from table) and Transformation Scenario (dropdown) are selected, we are presented with the same optional parameters as we saw in the previous, Harvest Job. We can leave the defaults again, double checking that the two Validation Scenarios – *DPLA minimum* and *Date checker* – are both checked under the “Validation Tests” tab.

When running Jobs, we also have the ability to select subsets of Records from input Jobs. Under the tab “Record Input Filter”, you can refine the Records that will be used in the following ways:

- **Refine by Record Validity:** Select Records based on their passing/failing of Validation tests
- **Limit Number of Records:** Select a numerical subset of Records, helpful for testing
- **Refine by Mapped Fields:** Most exciting, select subsets of Records based on an ElasticSearch query run against those input Jobs mapped fields

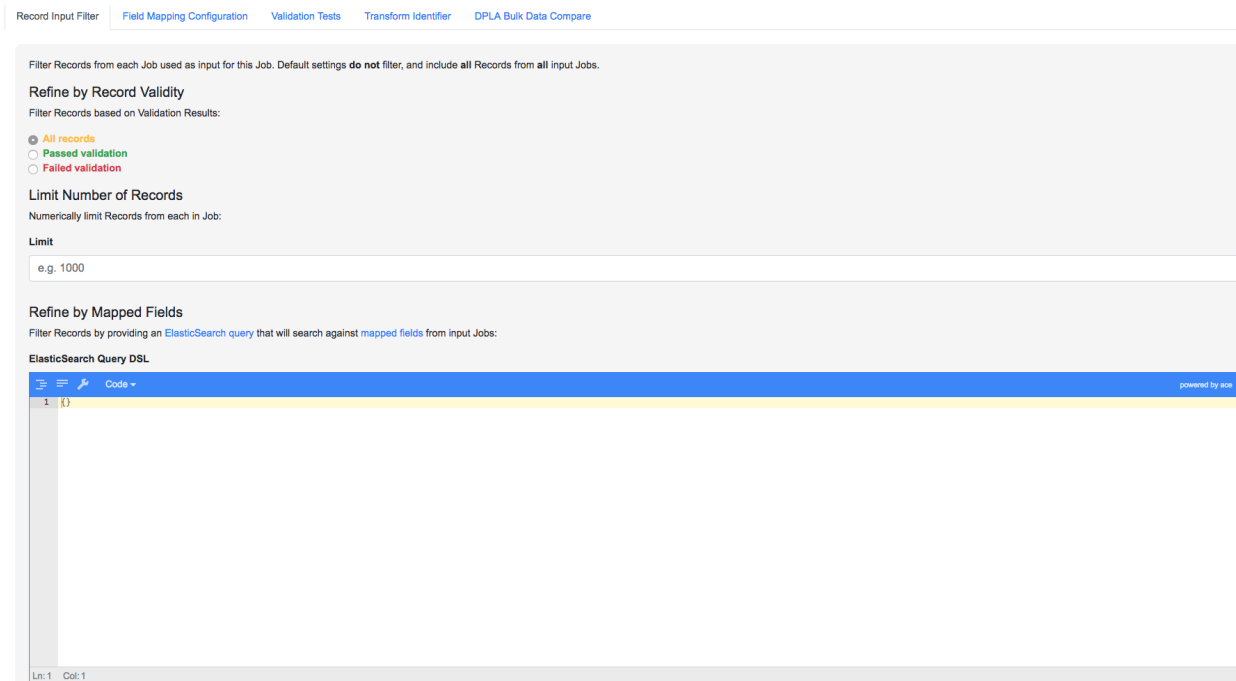


Fig. 10: Filters that can be applied to Records used as input for a Job

For the time being, we can leave these as default. Finally, click “Run Job” at the bottom.

Again, we are kicked back to the RecordGroup screen, and should hopefully see a Transform job with the status running. **Note:** The graph on this page near the top, now with two Jobs, indicates the original Harvest Job was the *input* for this new Transform Job.

#205, HarvestStaticXMLJob @ Apr. 12, 2018, 12:26:28 PM Records (250) #206, TransformJob @ Apr. 12, 2018, 12:39:25 PM

Show 100 entries Search:

	Job ID	Timestamp	Name	Record Group	Job Type	Status	Finished	Is Valid	Elapsed	Input	Notes	Total Record Count	Monitor	Actions
<input type="checkbox"/>	205	April 12, 2018, 12:26 p.m.	HarvestStaticXMLJob @ Apr. 12, 2018, 12:26:28 PM	Fedora Repository	HarvestJob	available	True	False	0:00:26 9.6 r/s	None	None	250	Livy / Spark	Details
<input type="checkbox"/>	206	April 12, 2018, 12:39 p.m.	TransformJob @ Apr. 12, 2018, 12:39:25 PM	Fedora Repository	TransformJob	running	False	True	0:00:00 None r/s	HarvestStaticXMLJob @ Apr. 12, 2018, 12:26:28 PM, Job #205, from Record Group: Fedora Repository	None	0	Livy / Spark	Details
				All	All	All	All	All						

Showing 1 to 2 of 2 entries Previous 1 Next

Fig. 11: Graph showing Transform Job with Harvest as Input, and All records sent

Transforms can take a bit longer than harvests, particularly with the additional Validation Scenarios we are running; but still a small job, might take anywhere from 15-30 seconds. Refresh the page until it shows the status as *available*.

Also of note, hopefully the *Is Valid* column is not red now, and should read *True*. We will look at validations in more detail, but because we ran the same Validation Scenarios on both Jobs, this suggests the XSLT transformation fixed whatever validation problems there were for the Records in the Harvest job.

3.1.10 Looking at Jobs and Records

Now is a good time to look at the details of the jobs we have run. Let's start by looking at the first **Harvest Job** we ran. Clicking the Job name in the table, or "details" link at the far-right will take you to a Job details page.

Note: Clicking the Job in the graph will gray out any other jobs in the table below that are not a) the job itself, or b) upstream jobs that served as inputs.

Job Details

This page provides details about a specific Job.

Major sections can be found behind the various tabs, and include:

- Records
 - a table of all records contained in this Job
- Mapped Fields
 - statistical breakdown of indexed fields, with ability to view values per field
- Input Jobs
 - what Jobs were used as inputs for this Job

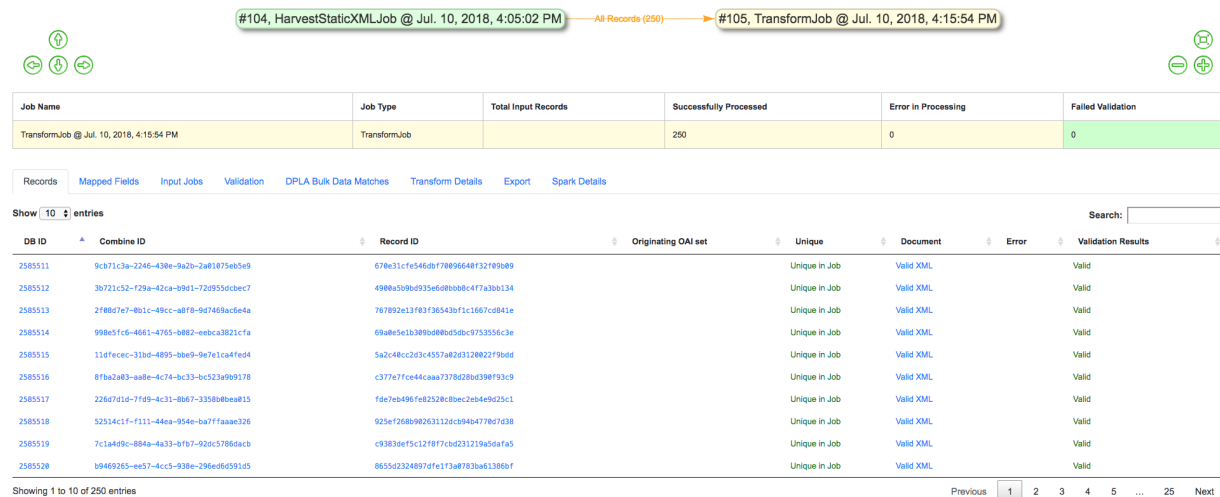
Job Details: **TransformJob @ Jul. 10, 2018, 4:15:54 PM** / Job Notes

Fig. 12: Screenshot of Job details page

- Validation
 - shows all Validations run for this Job, with reporting
- Job Type Specific Details
 - depending on the Job type, details relevant to that task (e.g. Transform Jobs will show all Records that were modified)
- DPLA Bulk Data Matches
 - if run and configured, shows matches with DPLA bulk data sets

Records

Sortable, searchable, this shows all the individual, discrete Records for this Job. This is *one*, but not the only, entry point for viewing the details about a single Record. It is also helpful for determining if the Record is unique *with respect to other Records from this Job*.

Mapped Fields

This table represents all mapped fields from the Record's original source XML record to ElasticSearch.

To this point, we have been using the default configurations for mapping, but more complex mappings can be provided when running a new Job, or when re-indexing a Job. These configurations are covered in more detail in [Field Mapping](#).

At a glance, field mapping attempts to convert XML into a key/value pairs suitable for a search platform like ElasticSearch. Combine does this via a library `xml2kvp`, which stands for "XML to Key/Value Pairs" that accepts a medley of configurations in JSON format. These JSON parameters are referred to as "Field Mapper Configurations" throughout.

For example, it might map the following XML block from a Record's MODS metadata:

```
<mods:mods>
  <mods:titleInfo>
    <mods:title>Edmund Dulac's fairy-book : </mods:title>
    <mods:subTitle>fairy tales of the allied nations</mods:subTitle>
  </mods:titleInfo>
</mods:mods>
```

to the following *two* ElasticSearch key/value pairs:

```
mods|mods_mods|titleInfo_mods|title : Edmund Dulac's fairy-book :
mods|mods_mods|titleInfo_mods|subTitle : fairy tales of the allied nations
```

An example of a field mapping configuration that could be applied would be the `remove_ns_prefix` which removes XML namespaces prefixes from the resulting fields. This would result in the following fields, removing the `mods` prefix and delimiter for each field:

```
mods_titleInfo_title : Edmund Dulac's fairy-book :
mods_titleInfo_subTitle : fairy tales of the allied nations
```

It can be dizzying at a glance, but it provides a thorough and comprehensive way to analyze the breakdown of metadata field usage across *all* Records in a Job. With, of course, the understanding that these “flattened” fields are not shaped like the raw, potentially hierarchical XML from the Record, but nonetheless crosswalk the values in one way or another.

Clicking on the mapped, ElasticSearch field name on the far-left will reveal all values for that dynamically created field, across all Records. Clicking on a count from the column `Document with Field` will return a table of Records that *have* a value for that field, `Document without` will show Records that *do not have* a value for this field.

An example of how this may be helpful: sorting the column `Documents without` in ascending order with zero at the top, you can scroll down until you see the count 11. This represents a subset of Records – 11 of them – that *do not have* the field `mods|mods_mods|subject_mods|topic`, which might itself be helpful to know. This is particularly true with fields that might represent titles, identifiers, or other required information. The far end of the column, we can see that 95% of Records have this field, and 34% of those have unique values.

mods mods_mods subject_mods topic	Select DPLA	239	11	725	249	34%	95%
-----------------------------------	-------------	-----	----	-----	-----	-----	-----

Fig. 13: Row from Indexed fields showing that 11 Records do not have this particular field

Clicking on the button “Show field analysis explanation” will reveal some information about other columns from this table.

Note: Short of an extended discussion about this mapping, and possible value, it is worth noting these indexed fields are used almost exclusively for **analysis** and **creating subsets through queries** of Records in Combine, and are not any kind of final mapping or transformation on the Record itself. The Record’s XML is always stored separately in MySQL (and on disk as Avro files), and is used for any downstream transformations or publishing. The only exception being where Combine attempts to query the DPLA API to match records, which is based on these mapped fields, but more on that later.

Validation

This table shows all the Validation Scenarios that were run for this job, including any/all failures for each scenario.

For our example Harvest, under *DPLA minimum*, we can see that there were 250 Records that failed validation. For the *Date checker* validation, all records passed. We can click “See Failures” link to get the specific Records that failed, with some information about which tests within that Validation Scenario they failed.

Records	Mapped Fields	Input Job	Validation	DPLA Bulk Data Matches	Harvest Details	Export	Spark Details
---------	---------------	-----------	------------	------------------------	-----------------	--------	---------------

Validation Scenarios

The following Validation Scenarios were run for this job:

Validation Name	Validation Type	Record Validation Failure Count	Actions
DPLA minimum	Schematron	250	See Failures Remove Validation from Job
Date checker	Python Code Snippet	0	Remove Validation from Job

[Generate Validation Results Report](#)
[Run new Validations for this Job](#)

Fig. 14: Two Validation Scenarios run for this Job

Additionally, we can click “Generate validation results report” to generate an Excel or .csv output of the validation results. From that screen, you are able to select:

- which Validation Scenarios to include in report
- any mapped fields (see below for an explanation of them) that would be helpful to include in the report as columns

More information about [Validation Scenarios](#).

Record Details

Next, we can drill down one more level and view the details of an individual Record. From the Record table tab, click on the Record ID of any individual Record. At this point, you are presented with the details of that particular Record.

Home / Organizations / Organization - Amazing University / RecordGroup - Fedora Repository / Job - HarvestStaticXMLJob @ Apr. 12, 2018, 12:26:28 PM / Record - 175c099be37b52c4b278400fb64e738d	
Record: 175c099be37b52c4b278400fb64e738d	
DB ID	1182035
Combine ID	0ace4b61-268e-4e57-a326-5d69f3a56c44
Record ID	175c099be37b52c4b278400fb64e738d
Valid?	False

Fig. 15: Top of Record details page, showing some overview information

Similar to a Job’s details, a Record details page has tabs that house the following sections:

- Record XML
- Indexed Fields
- Record stages
- Validation
- DPLA Link
- Job Type Specific

Record XML

The raw XML document for this Record. **Note:** As mentioned, regardless of how fields are mapped in Combine to Elasticsearch, the Record’s XML or “document” is always left intact, and is used for any downstream Jobs. Combine provides mapping and analysis of Records through mapping to Elasticsearch, but the Record’s XML document is stored as plain, `LONGTEXT` in MySQL for each Job.

Mapped Fields

Indexed Fields

Show 100 entries

Search:

Field Name	DPLA Mapped Field	Map DPLA Field	Field Value
combine_id		<input type="text" value="Sele"/>	0ace4b61-268e-4e57-a326-5d69f3a56c44
db_id		<input type="text" value="Sele"/>	1182035
mods_abstract		<input type="text" value="Sele"/>	"Edmund Dulac's fairy- book: fairy tales of the allied nations," was published in 1916. It contains "Snegorotchka: a Russian fairy tale," "The buried moon: an English fairy tale," "White Caroline and black Caroline: a Flemish fairy tale," "The seven conquerors of the Queen of the Mississippi: a Belgian fairy tale," "The serpent prince: an Italian fairy tale," "The hind of the wood, a French fairy tale," "Ivan and the chestnut horse: a Russian fairy tale," "The queen of the many-colored bedchamber, an Irish fairy tale," "The blue bird: a French fairy tale," "Bashitchelik, or, Real Steel: a Serbian fairy tale," "The friar and the boy: an English fairy tale," "The green serpent: a French fairy tale," "Urashima Taro: a Japanese fairy tale," and "The fire bird: a Russian fairy tale."
mods_accessCondition_@type_useAndReproduction		<input type="text" value="Sele"/>	This book is in the public domain.
mods_extension		<input type="text" value="Sele"/>	[b48448400', 'wayne:EdmundDula1916b48448400']
mods_identifier_@type_local		<input type="text" value="Sele"/>	EdmundDula1916b48448400
mods_identifier_@type_oclc		<input type="text" value="Sele"/>	881323771
mods_language_languageTerm_@authority_iso639-2b_@type_code		<input type="text" value="Sele"/>	eng
mods_language_languageTerm_@authority_iso639-2b_@type_text		<input type="text" value="Sele"/>	English
mods_location_url_@access_preview		<input type="text" value="Sele"/>	http://digital.library.wayne.edu/item/wayne:EdmundDula1916b48448400/thumbnail
mods_location_url_@usage_primary		<input type="text" value="Sele"/>	http://digital.library.wayne.edu/item/wayne:EdmundDula1916b48448400
mods_name_namePart		<input type="text" value="Sele"/>	Dulac, Edmund
mods_name_namePart_@type_date		<input type="text" value="Sele"/>	1882-1953
mods_name_role_roleTerm_@authority_marcrelator_@type_text		<input type="text" value="Sele"/>	Author

Fig. 16: Part of table showing indexed fields for Record

This table shows the individual fields in Elasticsearch that were mapped from the Record’s XML metadata. This can further reveal how this mapping works, by finding a unique value in this table, noting the Field Name, and then searching for that value in the raw XML below.

This table is mostly for informational purposes, but also provides a way to map generically mapped indexed fields from Combine, to known fields in the DPLA metadata profile. This can be done with the from the dropdowns under the DPLA Mapped Field column.

Why is this helpful? One goal of Combine is to determine how metadata will eventually map to the DPLA profile. Short of doing the mapping that DPLA does when it harvests from a Service Hub, which includes enrichments as well, we can nonetheless try and “tether” this record on a known unique field to the version that might currently exist in DPLA already.

To do this, two things need to happen:

1. register for a DPLA API key, and provide that key in `/opt/combine/combine/locasettings.py` for the variable `DPLA_API_KEY`.
2. find the URL that points to your actual item (not the thumbnail) in these mapped fields in Combine, and from the DPLA Mapped Field dropdown, select `isShownAt`. The `isShownAt` field in DPLA records contain the

URL that DPLA directs users *back* to, aka the actual item online. This is a particularly unique field to match on. If `title` or `description` are set, Combine will attempt to match on those fields as well, but `isShownAt` has proven to be much more accurate and reliable.

If all goes well, when you identify the indexed field in Combine that contains your item’s actual online URL, and map to `isShownAt` from the dropdown, the page will reload and fire a query to the DPLA API and attempt to match the record. If it finds a match, a new section will appear called “DPLA API Item match”, which contains the metadata from the DPLA API that matches this record.

This is an area still under development. Though the `isShownAt` field is usually very reliable for matching a Combine record to its live DPLA item counterpart, obviously it will not match if the URL has changed between harvests. Some kind of unique identifier might be even better, but there are problems there as well a bit outside the scope of this QuickStart guide.

Record stages

This table represents the various “stages”, aka Jobs, this Record exists in. This is good insight into how Records move through Combine. We should see two stages of this Record in this table: one for the original Harvest Job (bolded, as that is the version of the Record we are currently looking at), and one as it exists in the “downstream” Transform Job. We could optionally click the `ID` column for a downstream Record, which would take us to that *stage* of the Record, but let’s hold off on that for now.

For any stage in this table, you may view the Record Document (raw Record XML), the associated, mapped ElasticSearch document (JSON), or click into the Job details for that Record stage.

Note: Behind the scenes, a Record’s `combine_id` field is used for linking across Jobs. Formerly, the `record_id` was used, but it became evident that the ability to transform a Record’s identifier used for publishing would be important. The `combine_id` is not shown in this table, but can be viewed at the top of the Record details page. These are UUID4 in format.

Validation

This area shows all the Validation scenarios that were run for this Job, and how this specific record fared. In all likelihood, if you’ve been following this guide with the provided demo data, and you are viewing a Record from the original Harvest, you should see that it failed validation for the Validation scenario, *DPLA minimum*. It will show a row in this table for *each* rule from the Validation Scenario the Record failed, as a single Validation Scenario – schematron or python – may contain multiples rules / tests. You can click “Run Validation” to re-run and see the results of that Validation Scenario run against this Record’s XML document.

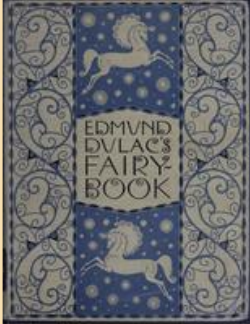
Harvest Details (Job Type Specific Details)

As we are looking at Records for a Harvest Job, clicking this tab will not provide much information. However, this is a good opportunity to think about how records are linked: we can look at the Transformation details for the same Record we are currently looking at.

To do this:

- Click the “Record Stages” tab
- Find the second row in the table, which is this same Record but as part of the Transformation Job, and click it
- From that new Record page, click the “Transform Details” tab
 - unlike the “Harvest Details”, this provides more information, including a diff of the Record’s original XML if it has changed

DPLA API Item match



format	Ebooks
subject	[{'name': 'Fairy tales'}]
rights	["This book is in the public domain."]
extent	1 online resource (1 v.)
language	[{'iso639_3': 'eng', 'name': 'English'}]
stateLocatedIn	[{'name': 'Michigan'}]
date	{'end': '1916', 'displayDate': '1916', 'begin': '1916'}
type	text
description	["Description based on print version record.", 'This metadata was created by Wayne State University Library system based on the catalog records of the print works also by the Wayne State University Library System', 'Edmund Dulac\'s fairy- book: fairy tales of the allied nations," was published in 1916. It contains \'Snegorotchka: a Russian fairy tale,\' \'The buried moon: an English fairy tale,\' \'White Caroline and black Caroline: a Flemish fairy tale,\' \'The seven conquerors of the Queen of the Mississippi: a Belgian fairy tale,\' \'The serpent prince: an Italian fairy tale,\' \'The hind of the wood, a French fairy tale,\' \'Ivan and the chestnut horse: a Russian fairy tale,\' \'The queen of the many-colored bedchamber, an Irish fairy tale,\' \'The blue bird: a French fairy tale,\' \'Bashtchelik, or, Real Steel: a Serbian fairy tale,\' \'The friar and the boy: an English fairy tale,\' \'The green serpent: a French fairy tale,\' \'Urashima Taro: a Japanese fairy tale,\' and \'The fire bird: a Russian fairy tale.']
creator	["Dulac, Edmund 1882-1953"]
@id	http://dp.la/api/items/0ea13f5f94feeb05e595cacfe4364623#sourceResource
collection	{'@id': "", 'description': "", 'title': 'Eloise Ramsey Collection of Literature for Young People', 'id': ""}
title	Edmund Dulac's fairy-book :

[DPLA item record](#)

Fig. 17: After isShownAt linked to indexed field, results of successful DPLA API query

Stages

Record DB ID	Record Record ID	Job ID	Job Name	Job Type	Record Document	Record Error	Is Valid	ElasticSearch document
1182035	175c099be37b52c4b278400fb64e738d	205	HarvestStaticXMLJob @ Apr. 12, 2018, 12:26:28 PM	HarvestStaticXMLJob	View	None	False	View
1182285	175c099be37b52c4b278400fb64e738d	206	TransformJob @ Apr. 12, 2018, 12:39:25 PM	TransformJob	View	None	True	View

Fig. 18: Showing stages of Record across Jobs

Validation Result: **Failed**

Validation Scenario	Test Failed	Actions
DPLA minimum	There must be a rights statement	Run Validation

Fig. 19: Showing results of Validation Scenarios applied to this Record

3.1.11 Duplicating and Merging Jobs

This QuickStart guide won't focus on Duplicating / Merging Jobs, but it worth knowing this is possible. If you were to click "Duplicate / Merge" link at the bottom of the RecordGroup page, you would be presented with a familiar Job creation screen, with one key difference: when selecting you input jobs, the radio buttons have been replaced by checkboxes, indicating you can select **multiple** jobs as input. Or, you can select a **single** Job as well.

The use cases are still emerging when this could be helpful, but here are a couple of examples. . .

Merging Jobs

In addition to "pulling" Jobs from one RecordGroup into another, it might also be beneficial to merge multiple Jobs into one. An example might be:

1. Harvest a single group of records via an OAI-PMH set
2. Perform a Transformation tailored to that group of records (Job)
3. Harvest *another* group of records via a different OAI-PMH set
4. Perform a Transformation tailored to *that* group of records (Job)
5. Finally, Merge these two Transform Jobs into one, suitable for publishing from this RecordGroup.

Here is a visual representation of this scenario, taken directly from the RecordGroup page:

Look for duplicates in Jobs

A more specific case might be looking for duplicates between two Jobs. In this scenario, there were two OAI endpoints with nearly the same records, but not identical. Combine allowed

1. Harvesting both
2. Merging and looking for duplicates in the Record table

3.1.12 Publishing Records

If you've made it this far, at this point we have:

- Created the Organization, "Amazing University"
- Created the RecordGroup, "Fedora Repository"

Jobs

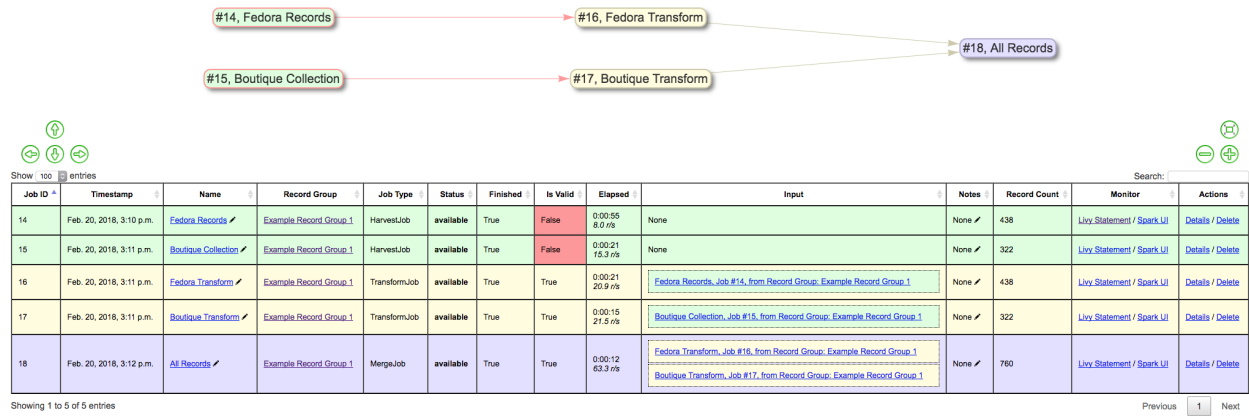


Fig. 20: Merge example



Fig. 21: Merge Job combining two Jobs of interest

DB ID	Combine ID	Record ID	Originating OAI set	Unique	Success	Document	Error	Validation Results
878393	b62ace25-62cb-48a3-b998-11a6a7a7e8c6	oai:digital.library.wayne.edu:wsudor_dpla:oai:digital.library.wayne.edu:wayne:A_PRO19811982_001	wsudor_dpla	Unique	true	Valid XML		Valid
857341	88a56697-b9c8-4fff-ace4-7e6eb58327e3	oai:digital.library.wayne.edu:wsudor_dpla:oai:digital.library.wayne.edu:wayne:NHY5_93249d	wsudor_dpla	Unique	true	Valid XML		Valid
891981	48e96abf-4911-4e3c-b71d-aed4c38cbfff	oai:digital.library.wayne.edu:wsudor_dpla:oai:digital.library.wayne.edu:wayne:A_PRO28892810_001	wsudor_dpla	Unique	true	Valid XML		Valid
859647	2839135a-6cd8-4a9e-b8c8-484567ea8c68	oai:digital.library.wayne.edu:wsudor_dpla:oai:digital.library.wayne.edu:wayne:A_PRO15781578_001	wsudor_dpla	Unique	true	Valid XML		Valid
851968	f9bed2ed-5588-4b82-8611-85d0f4ac8428	oai:digital.library.wayne.edu:wsudor_dpla:oai:digital.library.wayne.edu:wayne:vmc223_1	v2_ouroboros_oai_server	Duplicate	true	Valid XML		Valid
852224	55cae61c-2385-4fb1-bc61-87854cb8e8d3	oai:digital.library.wayne.edu:wsudor_dpla:oai:digital.library.wayne.edu:wayne:vmc78588_2	v2_ouroboros_oai_server	Duplicate	true	Valid XML		Valid
852488	88828489-9758-4db3-9fc3-576a498aabc8	oai:digital.library.wayne.edu:wsudor_dpla:oai:digital.library.wayne.edu:wayne:vmc39730	wsudor_dpla	Duplicate	true	Valid XML		Valid
852736	e1bf5eda-87be-4b2d-b435-b9ce3c4fb1d3	oai:digital.library.wayne.edu:wsudor_dpla:oai:digital.library.wayne.edu:wayne:EP02_64_9_4	wsudor_dpla	Duplicate	true	Valid XML		Valid
852992	4bee9849-be6d-4788-8aac-ccad1ce1b95c	oai:digital.library.wayne.edu:wsudor_dpla:oai:digital.library.wayne.edu:wayne:vmc63783_1	v2_ouroboros_oai_server	Duplicate	true	Valid XML		Valid
853248	1e58961f-b487-4798-9a6d-d2fd064c269	oai:digital.library.wayne.edu:wsudor_dpla:oai:digital.library.wayne.edu:wayne:vmc10863	v2_ouroboros_oai_server	Duplicate	true	Valid XML		Valid

Showing 111 to 120 of 92,756 entries

Previous 1 ... 11 12 13 ... 9276 Next

Fig. 22: Analysis of Records from Merge Job shows duplicates

- Harvested 250 Records from static XML files
- Transformed those 250 Records to meet our Service Hub profile
 - thereby also fixing validation problems revealed in Harvest
- Looked at Job and Record details

Now, we may be ready to “publish” these materials from Combine for harvesting by others (e.g. DPLA).

Overview

Publishing is done at the **RecordGroup** level, giving more weight to the idea of a RecordGroup as a meaningful, intellectual group of records. When a RecordGroup is published, it can be given a “Publish Set ID”, which translates directly to an OAI-PMH **set**. **Note:** It is possible to publish multiple, distinct RecordGroups with the same publish ID, which has the effect of allowing multiple RecordGroups to be published under the same OAI-PMH set.

Combine comes with an **OAI-PMH server baked in** that serves all published RecordGroups via the OAI-PMH HTTP protocol.

Publishing a RecordGroup

To run a Publish Job and publish a RecordGroup, navigate to the RecordGroup page, and near the top click the “Publish” button inside the top-most, small table.

[Home](#) / [Organizations](#) / [Organization - Amazing University](#) / [RecordGroup - Fedora Repository](#)

Record Group: Fedora Repository

Description	None	
Published?	This Record Group has not been published yet	<button>Publish</button>

Fig. 23: Record Group has not yet been published...

You will be presented with a new Job creation screen.

Near the top, there are some fields for entering information about an Publish set identifier. You can either select a previously used Publish set identifier from the dropdown, or create a new one. Remember, this will become the OAI set identifier used in the **outgoing** Combine OAI-PMH server.

Let’s give it a new, simple identifier: `fedora`, representing that this RecordGroup is a workspace for Jobs and Records from our Fedora repository.

Then, from the table below, select the Job (again, think as a *stage* of the same records) that will be published for this RecordGroup. Let’s select the Transformation Job that had passed all validations.

Finally, click “Publish” at the bottom.

The next step is to assign a **Publish Set ID** for this Record Group. **Note:** This will translate to an OAI set for the outbound Combine OAI-PMH server.

You have the option of creating and establishing a new one, or reusing a pre-existing one from the dropdown, effectively grouping Records from this Job under that Published set.

Create a new Publish Set ID

Select a pre-existing Publish Set ID

Fig. 24: Section to provide a new publish identifier, or select a pre-existing one

You will be returned to the RecordGroup, and should see a new Publish Job with status `running`, further extending the Job “lineage” graph at the top. Publish Jobs are usually fairly quick, as they are copy most data from the Job that served as input.

In a few seconds you should be able to refresh the page and see this Job status switch to `available`, indicating the publishing is complete.

Near the top, you can now see this Record Group is published:

[Home](#) /
 [Organizations](#) /
 [Organization - Amazing University](#) /
 [RecordGroup - Fedora Repository](#)

Record Group: Fedora Repository

Description	None	
Published?	Job #207, PublishJob @ Apr. 12, 2018, 12:58:24 PM	Unpublish
Publish Set ID	fedora	Change Publish Set ID

Fig. 25: Published Record Group

Let’s confirm and see them as published records...

Viewing published records

From any screen, click the “Published” link at the very top in the navigation links. This brings you to a new page with some familiar looking tables.

At the very top is a section “Published Sets”. These show all **RecordGroups** that have been published, with the corresponding OAI set identifier. This also provides a button to unpublish a RecordGroup (also doable from the RecordGroup page).

[Home](#) / [Published](#)

Published Records

Published Sets

All published Record Groups. One job from each Record Group may be published, taking the `publish_set_id` from that Record Group as the OAI set ID. In some cases this may be “Not set”, resulting in Records that are not aggregated under an OAI set, but will be returned via [ListRecords](#).

Record Group	OAI Set	Published Job	Record Count	Action	Action
Fedora Repository	fedora	PublishJob @ Apr. 12, 2018, 12:58:24 PM	250	Unpublish	Change Publish Set ID
		Total:	250		

Fig. 26: Currently published Record Groups, with their publish set identifier

To the right is an area that says, “Analysis.” Clicking this button will fire a new Analysis Job – which has not yet been covered, but is essentially an isolated Job that takes 1+ Jobs from any Organization, and RecordGroup, for the purpose of analysis – with all the Published Jobs automatically selected. This provides a single point of analysis for all Records published from Combine.

Below that is a table – similar to the table from a single Job details – showing all **Records** that are published, spanning all RecordGroups and OAI sets. One column of note is `Unique in Published?` which indicates whether or not this Record is unique among all published Records. **Note:** This test is determined by checking the `record_id` field for published records; if two records are essentially the same, but have different `record_ids`, this will not detect that.

Below that table, is the familiar “Indexed Fields” table. This table shows mapped, indexed fields in ElasticSearch for *all* Records across *all* RecordGroups published. Similar to a single Job, this can be useful for determining irregularities among published Records (e.g. small subset of Records that don’t have an important field).

Finally, at the very bottom are some links to the actual OAI-PMH server coming out of Combine, representing four common OAI-PMH verbs:

- [Identify](#)
 - basic identification of the Combine OAI-PMH server
- [List Identifiers](#)
 - list OAI-PMH identifiers for all published Records
- [List Records](#)
 - list full records for all published Records (primary mechanism for harvest)
- [List Sets](#)
 - list all OAI-PMH sets, a direct correlation to OAI sets identifiers for each published RecordGroup

3.1.13 Analysis Jobs

From any screen, clicking the “Analysis” link at the top in the navigation links will take you to the Analysis Jobs space. Analysis Jobs are a special kind of Job in Combine, as they are meant to operate outside the workflows of a RecordGroup.

Analysis Jobs look and feel very much like Duplicate / Merge Jobs, and that’s because they share mechanisms on the back-end. When starting a new Analysis Job, by clicking the “Run new analysis job” link at the bottom of the page, you are presented with a familiar screen to run a new Job. However, you’ll notice that you can select Jobs from any RecordGroup, and *multiple* jobs if so desired, much like Duplicate/Merge Jobs.

An example use case may be running an Analysis Job across a handful of Jobs, in different RecordGroups, to get a sense of how fields are used. Or run a battery or validation tests that may not relate directly to the workflows of a RecordGroup, but are helpful to see all the same.

Analysis Jobs are *not* shown in RecordGroups, and are not available for selection as input Jobs from any other screens; they are a bit of an island, solely for the purpose of their Analysis namesake.

3.1.14 Troubleshooting

Undoubtedly, things might go sideways! As Combine is still quite rough around some edges, here are some common gotchas you may encounter.

Run a job, status immediately flip to `available`, and has no records

The best way to diagnose why a job may have failed, from the RecordGroup screen, is to click “Livy Statement” link under the `Monitor` column. This returns the raw output from the Spark job, via Livy which dispatches jobs to Spark.

A common error is a stale Livy connection, specifically its MySQL connection, which is revealed at the end of the Livy statement output by:

```
MySQL server has gone away
```

This can be fixed by *restarting the Livy session*.

Cannot start a Livy session

Information for diagnosing can be found in the Livy logs at `/var/log/livy/livy.stderr`.

3.2 Data Model

3.2.1 Overview

Combine’s Data Model can be roughly broken down into the following hierarchy:

```
Organization --> RecordGroup --> Job --> Record
```

3.2.2 Organization

Organizations are the highest level of organization in Combine. It is loosely based on a “Data Provider” in REPOX, also the highest level of hierarchy. Organizations contain Record Groups.

Combine was designed to be flexible as to where it exists in a complicated ecosystem of metadata providers and harvesters. Organizations are meant to be helpful if a single instance of Combine is used to manage metadata from a variety of institutions or organizations.

Other than a level of hierarchy, Organizations have virtually no other affordances.

We might imagine a single instance of Combine, with two Organizations:

- Foo University
- Bar Historical Society

Foo University would contain all Record Groups that pertain to Foo University. One can imagine that Foo University has a Fedora Repository, Omeka, and might even aggregate records for a small historical society or library as well, each of which would fall under the Organization.

3.2.3 Record Group

Record Groups fall under Organizations, and are loosely based on a “Data Set” in REPOX. Record Groups contain Jobs.

Record Groups are envisioned as the right level of hierarchy for a group of records that are intellectually grouped, come from the same system, or might be managed with the same transformations and validations.

From our Foo University example above, the Fedora Repository, Omeka installs, and the records from a small historical society – all managed and mediated by Foo University – might make nice, individual, distinct Record Groups.

3.2.4 Job

Jobs are contained with a Record Group, and contain Records.

This is where the model forks from REPOX, in that a Record Group can, and likely will, contain multiple Jobs. It is reasonable to also think of a Job as a *stage* of records.

Jobs represent Records as they move through the various stages of harvesting, sub-dividing, and transforming. In a typical Record Group, you may see Jobs that represent a harvest of records, another for transforming the records, perhaps yet another transformation, and finally a Job that is “published”. In this way, Jobs also provide an approach to versioning Records.

Imagine the record `baz` that comes with the harvest from `Job1`. `Job2` is then a transformation style Job that uses `Job1` as input. `Job3` might be another transformation, and `Job4` a final publishing of the records. In each Job, the record `baz` exists, at those various stages of harvesting and transformation. Combine errs on the side of duplicating data in the name of lineage and transparency as to how and why a Record “downstream” looks the way it does.

As may be clear by this point, Jobs are used as **input** for other Jobs. `Job1` serves as the input Records for `Job2`, `Job2` for `Job3`, etc.

There are four primary types of Jobs:

- Harvests
- Transformations
- Merge / Duplicate
- Analysis

It is up to the user how to manage Jobs in Combine, but one strategy might be to leave previous harvests, transforms, and merges of Jobs within a RecordGroup for historical purposes. From an organizational standpoint, this may look like:

```
Harvest, 1/1/2017 --> Transform to Service Hub Profile
Harvest, 4/1/2017 --> Transform to Service Hub Profile
Harvest, 8/1/2017 --> Transform to Service Hub Profile
Harvest, 1/1/2018 --> Transform to Service Hub Profile (Published)
```

In this scenario, this Record Group would have **9** total Jobs, but only the last “set” of Jobs would represent the currently published Records.

Harvest Jobs

Harvest Jobs are how Records are initially created in Combine. This might be through OAI-PMH harvesting, or loading from static files.

As the creator of Records, Harvest Jobs do *not* have input Jobs.

Transformation Jobs

Transformation Jobs, unsurprisingly, transform the Records in some way! Currently, XSLT and python code snippets are supported.

Transformation Jobs allow a **single** input Job, and are limited to Jobs within the same RecordGroup.

Merge / Duplicate Jobs

Merge / Duplicate Jobs are true to their namesake: merging Records across multiple Jobs, or duplicating all Records from a single Job, into a new, single Job.

Analysis Jobs

Analysis Jobs are Merge / Duplicate Jobs in nature, but exist outside of the normal

```
Organization --> Record Group
```

hierarchy. Analysis Jobs are meant as ephemeral, disposable, one-off Jobs for analysis purposes only.

3.2.5 Record

The most granular level of hierarchy in Combine is a single Record. Records are part of Jobs.

Record’s actual XML content, and other attributes, are recorded in MongoDB, while their indexed fields are stored in ElasticSearch.

Identifiers

Additionally, Record’s have three important identifiers:

- **Database ID**
 - `id` (integer)

- This is the ObjectID in MongoDB, unique for all Records

- **Combine ID**

- `combine_id` (string)
- this is randomly generated for a Record on creation, and is what allows for linking of Records across Jobs, and is unique for all Records

- **Record ID**

- `record_id` (string)
- not necessarily unique for all Records, this is identifier is used for publishing
- in the case of OAI-PMH harvesting, this is likely populated from the OAI identifier that the Record came in with
- this can be modified with a Record Identifier Transform when run with a Job

Why the need to transform identifiers?

Imagine the following scenario:

Originally, there were multiple REPOX instances in play for a series of harvests and transforms. With each OAI “hop”, the identifier for a Record is prefixed with information about that particular REPOX instance.

Now, with a single instance of Combine replacing multiple REPOX instances and OAI “hops”, records that are harvested are missing pieces of the identifier that were previously created along the way.

Or, insert a myriad of other reasons why an identifier may drift or change.

Combine allows for the creation of [Record Identifier Transformation Scenarios](#) that allow for the modification of the `record_id`. This allows for the emulation of previous configurations or ecosystems, or optionally creating Record Identifiers – what is used for publishing – based on information from the Record’s XML record with XPath or python code snippets.

3.3 Spark and Livy

Combine was designed to provide a single point of interaction for metadata harvesting, transformation, analysis, and publishing. Another guiding factor was a desire to utilize [DPLA’s Ingestion 3](#) codebase where possible, which itself, uses [Apache Spark](#) for processing large numbers of records. The decision to use Ingestion 3 drove the architecture of Combine to use Apache Spark as the primary, background context and environment for processing Records.

This is well and good from a command line, issuing individual tasks to be performed, but how would this translate to a GUI that could be used to initiate tasks, queue them, and view the results? It became evident that an intermediary piece was needed to facilitate running Spark “jobs” from a request/response oriented front-end GUI. [Apache Livy](#) was suggested as just such a piece, and fit the bill perfectly. Livy allows for the submission of jobs to a running Spark context via JSON, and the subsequent ability to “check” on the status of those jobs.

As Spark natively allows python code as a language for submitting jobs, Django was chosen as the front-end framework for Combine, to have some parity between the language of the GUI front-end and the language of the actual code submitted to Spark for batch processing records.

This all conspires to make Combine relatively fast and efficient, but adds a level of complexity. When Jobs are run in Combine, they are submitted to this running, background Spark context via Livy. While Livy is utilized in a similar fashion at scale for large enterprise systems, it is often obfuscated from users and the front-end GUI. This is partially the case for Combine.

3.3.1 Livy Sessions

Livy creates Spark contexts that can receive jobs via what it calls “sessions”. In Combine, only one active Livy session is allowed at a time. This is partially for performance reasons, to avoid gobbling up all of the server’s resources, and partially to enforce a sequential running of Spark Jobs that avoids many of the complexities that would be introduced if Jobs – that require input from the output of one another – were finishing at different times.

Manage Livy Sessions

Navigate to the “System” link at the top-most navigation of Combine. If no Livy sessions are found or active, you will be presented with a screen that looks like this:

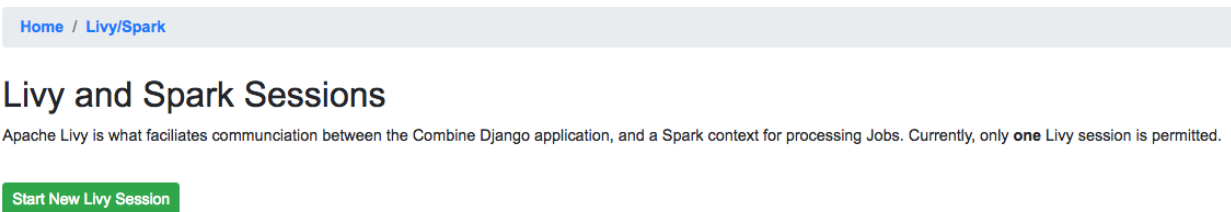


Fig. 27: Livy sessions management: No Livy sessions found

To begin a Livy session, click “Start New Livy Session”. The page will refresh and you should see a screen that shows the Livy session is starting:

Home / Livy/Spark								
Livy and Spark Sessions								
Apache Livy is what facilitates communication between the Combine Django application, and a Spark context for processing Jobs. Currently, only one Livy session is permitted.								
ID	Active	Name	Livy Session ID	Livy Session GUI	Status	Timestamp	Spark Session GUI	actions
79	True	Livy Session, sessionid 2	2	http://192.168.45.10:8998/ui/session/2	starting	May 14, 2018, 2:33 p.m.	http://192.168.45.10:4040	Stop

Fig. 28: Livy sessions management: Livy session starting

After 10-20 seconds, the page can be refreshed and it should show the Livy session as **idle**, meaning it is ready to receive jobs:

Home / Livy/Spark								
Livy and Spark Sessions								
Apache Livy is what facilitates communication between the Combine Django application, and a Spark context for processing Jobs. Currently, only one Livy session is permitted.								
ID	Active	Name	Livy Session ID	Livy Session GUI	Status	Timestamp	Spark Session GUI	actions
79	True	Livy Session, sessionid 2	2	http://192.168.45.10:8998/ui/session/2	idle	May 14, 2018, 2:33 p.m.	http://192.168.45.10:4040	Stop

Fig. 29: Livy sessions management: Livy session idle

Barring any errors with Livy, this is the only interaction with Livy that a Combine user needs to concern themselves with. If this Livy Session grows stale, or is lost, Combine will attempt to automatically restart when it’s needed. This will actually remove and begin a new session, but this should remain invisible to the casual user.

However, a more advanced user may choose to **remove** an active Livy session from Combine from this screen. When this happens, Combine cannot automatically refresh the Livy connection when needed, and all work requiring Spark will fail. To begin using Livy/Spark again, a new Livy session will need to be manually started per the instructions above.

3.4 Configuration

Combine relies heavily on front-loading configuration, so that the process of running Jobs is largely selecting pre-existing “scenarios” that have already been tested and configured.

This section will outline configuration options and associated configuration pages.

- *Field Mapping*
- *OAI-PMH Harvesting Endpoints*
- *Transformation Scenarios*
- *Validation Scenarios*
- *Record Identifier Transformation Scenarios (RITS)*
- *Built-In OAI-PMH server*
- *DPLA Bulk Data Downloads*

Note: Currently, Combine leverages Django’s built-in admin interface for editing and creating model instances – transformations, validations, and other scenarios – below. This will likely evolve into more tailored CRUDs for each, but for the time being, there is a link to the Django admin panel on the Configuration screen.

Note: What settings are not configurable via the GUI in Combine, are configurable in the file `combine/localsettings.py`.

3.4.1 Field Mapper Configurations

Field Mapping is the process of mapping values from a Record’s source document (likely XML) and to meaningful and analyzable key/value pairs that can be stored in ElasticSearch. These mapped values from a Record’s document are used in Combine for:

- analyzing distribution of XML elements and values across Records
- exporting to mapped field reports
- for single Records, querying the DPLA API to check existence
- comparing Records against DPLA bulk data downloads
- and much more!

To perform this mapping, Combine uses an internal library called `XML2kvp`, which stands for “XML to Key/Value Pairs”, to map XML to key/value JSON documents. Under the hood, `XML2kvp` uses `xmldict` to parse the Record XML into a hierarchical dictionary, and then loops through that, creating fields based on the configurations below.

I’ve mapped DC or MODS to Solr or ElasticSearch, why not do something similar?

Each mapping is unique: to support different access, preservation, or analysis purposes. A finely tuned mapping for one metadata format or institution, might be unusable for another, even for the same metadata format. Combine strives to be metadata format agnostic for harvesting, transformation, and analysis, and furthermore, performing these actions before a mapping has even been created or considered. To this end, a “generic” but customizable mapper was needed to take XML records and convert them into fields that can be used for developing an understanding about a group of Records.

While applications like Solr and ElasticSearch more recently support hierarchical documents, and would likely support a straight XML to JSON converted document (with `xmldict`, or [Object Management Group \(OMG\)’s XML to JSON conversion standard](#)), the attributes in XML give it a dimensionality beyond simple hierarchy, and can be critical to

understanding the nature and values of a particular XML element. These direct mappings would function, but would not provide the same scannable, analysis of a group of XML records.

XML2kvp provides a way to blindly map most any XML document, providing a broad overview of fields and structures, with the ability to further narrow and configure. A possible update/improvement would be the ability for users to upload mappers of their making (e.g. XSLT) that would result in a flat mapping, but that is currently not implemented.

How does it work

XML2kvp converts elements from XML to key/value pairs by converting hierarchy in the XML document to character delimiters.

Take for example the following, “unique” XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<root xmlns:internet="http://internet.com">
  <foo>
    <bar>42</bar>
    <baz>109</baz>
  </foo>
  <foo>
    <bar>42</bar>
    <baz>109</baz>
  </foo>
  <foo>
    <bar>9393943</bar>
    <baz>3489234893</baz>
  </foo>
  <tronic type='tonguetwister'>Sally sells seashells by the seashore.</tronic>
  <tronic type='tonguetwister'>Red leather, yellow leather.</tronic>
  <tronic>You may disregard</tronic>
  <goober scrog='true' tonk='false'>
    <depths>
      <plunder>Willy Wonka</plunder>
    </depths>
  </goober>
  <nested_attribs type='first'>
    <another type='second'>paydirt</another>
  </nested_attribs>
  <nested>
    <empty></empty>
  </nested>
  <internet:url url='http://example.com'>see my url</internet:url>
  <beat type="4/4">four on the floor</beat>
  <beat type="3/4">waltz</beat>
  <ordering>
    <duck>100</duck>
    <duck>101</duck>
    <goose>102</goose>
    <it>run!</it>
  </ordering>
  <ordering>
    <duck>200</duck>
    <duck>201</duck>
    <goose>202</goose>
    <it>run!</it>
  </ordering>
</root>
```

Converted with default options from XML2kvp, you would get the following key/value pairs in JSON form:

```
{'root_beat': ('four on the floor', 'waltz'),
'root_foo_bar': ('42', '9393943'),
'root_foo_baz': ('109', '3489234893'),
'root_goober_depths_plunder': 'Willy Wonka',
'root_nested_attribs_another': 'paydirt',
'root_ordering_duck': ('100', '101', '200', '201'),
'root_ordering_goose': ('102', '202'),
'root_ordering_it': 'run!',
'root_tronic': ('Sally sells seashells by the seashore.',
'Red leather, yellow leather.',
'You may disregard'),
'root_url': 'see my url'}
```

Some things to notice...

- the XML root element `<root>` is present for all fields as `root`
- the XML hierarchy `<root><foo><bar>` repeats twice in the XML, but is collapsed into a single field `root_foo_bar`
 - moreover, because `skip_repeating_values` is set to `true`, the value `42` shows up only once, if set to `false` we would see the value `('42', '42', '9393943')`
- a distinct absence of all attributes from the original XML, this is because `include_all_attributes` is set to `false` by default.

Running with `include_all_attributes` set to `true`, we see a more complex and verbose output, with `@` in various field names, indicating attributes:

```
{'root_beat_@type=3/4': 'waltz',
'root_beat_@type=4/4': 'four on the floor',
'root_foo_bar': ('42', '9393943'),
'root_foo_baz': ('109', '3489234893'),
'root_goober_@scrog=true_@tonk=false_depths_plunder': 'Willy Wonka',
'root_nested_attribs_@type=first_another_@type=second': 'paydirt',
'root_ordering_duck': ('100', '101', '200', '201'),
'root_ordering_goose': ('102', '202'),
'root_ordering_it': 'run!',
'root_tronic': 'You may disregard',
'root_tronic_@type=tonguetwister': ('Sally sells seashells by the seashore.',
'Red leather, yellow leather.'),
'root_url_@url=http://example.com': 'see my url'}
```

A more familiar example may be Dublin Core XML:

```
<oai_dc:dc xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:dc="http://
↪purl.org/dc/elements/1.1/" xmlns:oai_dc="http://www.openarchives.org/OAI/2.0/oai_dc/
↪" xmlns="http://www.openarchives.org/OAI/2.0/" xsi:schemaLocation="http://www.
↪openarchives.org/OAI/2.0/oai_dc/ http://www.openarchives.org/OAI/2.0/oai_dc.xsd">
  <dc:title>Fragments of old book</dc:title>
  <dc:creator>Unknown</dc:creator>
  <dc:date>1601</dc:date>
  <dc:description>An object of immense cultural and historical worth</
↪dc:description>
  <dc:subject>Writing--Materials and instruments</dc:subject>
  <dc:subject>Archaeology</dc:subject>
  <dc:coverage>1600-1610</dc:coverage>
```

(continues on next page)

(continued from previous page)

```
<dc:identifier>book_1234</dc:identifier>
</oai_dc:dc>
```

And with default configurations, would map to:

```
{'dc_coverage': '1600-1610',
 'dc_creator': 'Unknown',
 'dc_date': '1601',
 'dc_description': 'An object of immense cultural and historical worth',
 'dc_identifier': 'book_1234',
 'dc_subject': ('Writing--Materials and instruments', 'Archaeology'),
 'dc_title': 'Fragments of old book'}
```

Configurations

Within Combine, the configurations passed to XML2kvp are referred to as “Field Mapper Configurations”, and like many other parts of Combine, can be named, saved, and updated in the database for later, repeated use. This following table describes the configurations that can be used for field mapping.

Parameter	Type	Description
add_literals	object	Key/value pairs for literals to mixin, e.g. <code>foo:bar</code> would create field <code>foo</code> with value <code>bar</code> [Default: {}]
capture_array_attributes	Array of attributes	Attributes to capture values from and set as standalone field, e.g. if <code>[age]</code> is provided and encounters <code><foo age='42'/></code> , a field <code>foo_@age@</code> would be created (note the additional trailing <code>@</code> to indicate an attribute value) with the value <code>42</code> . [Default: [], Before: <code>copy_to, copy_to_regex</code>]
concat_value	Boolean	Boolean string to join all values from multivalued field on [Default: false]
concat_value_key	Key/value pairs	Key/value pairs for fields to concat on provided value, e.g. <code>foo_bar:-</code> if encountering <code>foo_bar:[goober, 'tronic']</code> would concatenate to <code>foo_bar:goober-tronic</code> [Default: {}]
copy_to_object	Key/value pairs	Key/value pairs to copy one field to another, optionally removing original field, based on regex match of field, e.g. <code>.*foo:bar</code> would copy create field <code>bar</code> and copy all values fields <code>goober_foo</code> and <code>tronic_foo</code> to <code>bar</code> . Note: Can also be used to remove fields by setting the target field as false, e.g. <code>.*bar:false</code> , would remove fields matching regex <code>.*bar</code> [Default: {}]
copy_to_object_key	Key/value pairs	Key/value pairs to copy one field to another, optionally removing original field, e.g. <code>foo:bar</code> would create field <code>bar</code> and copy all values when encountered for <code>foo</code> to <code>bar</code> , removing <code>foo</code> . However, the original field can be retained by setting <code>remove_copied_key</code> to <code>true</code> . Note: Can also be used to remove fields by setting the target field as false, e.g. <code>'foo':false</code> , would remove field <code>foo</code> . [Default: {}]
copy_value_key	Key/value pairs	Key/value pairs that match values based on regex and copy to new field if matching, e.g. <code>http.*:websites</code> would create new field <code>websites</code> and copy <code>http://example.com</code> and <code>https://example.org</code> to new field <code>websites</code> [Default: {}]
error_on_delimiter_collision	Boolean	Boolean to raise <code>DelimiterCollision</code> exception if delimiter strings from either <code>node_delim</code> or <code>ns_prefix_delim</code> collide with field name or field value (false by default for permissive mapping, but can be helpful if collisions are essential to detect) [Default: false]
exclude_array_attributes	Array of attributes	Attributes to skip when creating field names, e.g. <code>[baz]</code> when encountering XML <code><foo><bar baz='42' goober='1000'>tronic</baz></foo></code> would create field <code>foo_bar_@goober=1000</code> , skipping attribute <code>baz</code> [Default: []]
exclude_array_elements	Array of elements	Elements to skip when creating field names, e.g. <code>[baz]</code> when encountering field <code><foo><baz><bar>tronic</bar></baz></foo></code> would create field <code>foo_bar</code> , skipping element <code>baz</code> [Default: [], After: <code>include_all_attributes, include_attributes</code>]
include_all_attributes	Boolean	Boolean to consider and include all attributes when creating field names, e.g. if false, XML elements <code><foo><bar baz='42' goober='1000'>tronic</baz></foo></code> would result in field name <code>foo_bar</code> without attributes included. Note: the use of all attributes for creating field names has the the potential to balloon rapidly, potentially encountering ElasticSearch field limit for an index, therefore false by default. [Default: false, Before: <code>include_attributes, exclude_attributes</code>]
include_array_attributes	Array of attributes	Attributes to include when creating field names, despite setting of <code>include_all_attributes</code> , e.g. <code>[baz]</code> when encountering XML <code><foo><bar baz='42' goober='1000'>tronic</baz></foo></code> would create field <code>foo_bar_@baz=42</code> [Default: [], Before: <code>exclude_attributes</code> , After: <code>include_all_attributes</code>]
include_meta	Boolean	Boolean to include <code>xml2kvp_meta</code> field with output that contains all these configurations [Default: false]
node_delim	String	String to use as delimiter between XML elements and attributes when creating field name, e.g. <code>__</code> will convert XML <code><foo><bar>tronic</bar></foo></code> to field name <code>foo__bar</code> [Default: <code>_</code>]
ns_prefix_delim	String	String to use as delimiter between XML namespace prefixes and elements, e.g. <code> </code> for the XML <code><ns:foo><ns:bar>tronic</ns:bar></ns:foo></code> will create field name <code>ns foo_ns:bar</code> . Note: a <code> </code> pipe character is used to avoid using a colon in ElasticSearch fields, which can be problematic. [Default: <code> </code>]
remove_copied_key	Boolean	Boolean to determine if originating field will be removed from output if that field is copied to another field [Default: true]
remove_copied_value	Boolean	Boolean to determine if value will be removed from originating field if that value is copied to another field [Default: true]

Saving and Reusing

Field Mapper configurations may be saved, named, and re-used. This can be done anytime field mapper configurations are being set, e.g. when running a new Job, or re-indexing a previously run Job.

Testing

Field Mapping can also be tested against a single record, accessible from a Record's page under the "Run/Test Scenarios for this Record" tab. The following is a screenshot of this testing page:

The screenshot displays the 'Test Field Mapper' interface. At the top, there's a 'Test Field Mapper' header with a 'Instructions' link. Below it, a table lists records with columns: DB ID, Record ID, Job, Originating OAI set, Unique, Document, Error, and Validation Results. A search bar is on the right. Below the table, a message states 'Showing 1 to 1 of 1 entries (filtered from 247,379 total entries)'. The 'Field Mapper Configuration JSON' section follows, with a 'Configuration Name' dropdown set to 'Default'. Below this is a code editor showing a JSON configuration for field mapping. At the bottom, the 'Mapping Results' section shows the output of the mapping process, including the mapped record and its associated data.

Fig. 30: Testing Field Mapper Configurations

In this screenshot, you can see a single Record is used as input, a Field Mapper Configurations applied, and the resulting mapped fields at the bottom.

3.4.2 OAI Server Endpoints

Configuring OAI endpoints is the first step for harvesting from OAI endpoints.

To configure a new OAI endpoint, navigate to the Django admin screen, under the section "Core" select Oai endpoints.

This model is unique among other Combine models in that these values are sent almost untouched to the DPLA Ingestion 3 OAI harvesting codebase. More [information on these fields can be found here](#).

The following fields are all required:

- Name - Human readable name for OAI endpoint, used in dropdown menu when running harvest
- Endpoint - URL for OAI server endpoint. This should include the full URL up until, but not including, GET parameters that begin with a question mark ?.
- Verb - This pertains to the OAI-PMH verb that will be used for harvesting. Almost always, ListRecords is the required verb here. So much, this will default to ListRecords if left blank.

- `MetadataPrefix` - Another OAI-PMH term, the metadata prefix that will be used during harvesting.
- `Scope type` - Not an OAI term, this refers to what kind of harvesting should be performed. Possible values include:
 - `setList` - This will harvest the comma separated sets provided for `Scope value`.
 - `harvestAllSets` - The most performant option, this will harvest all sets from the OAI endpoint. If this is set, the `Scope value` field must be set to `true`.
 - `blacklist` - Comma separated list of OAI sets to **exclude** from harvesting.
- `Scope value` - String to be used in conjunction with `Scope type` outline above.
 - If `setList` is used, provide a comma separated string of OAI sets to harvest
 - If `harvestAllSets`, provide just the single string `true`.

Once the OAI endpoint has been added in the Django admin, from the configurations page you are presented with a table showing all configured OAI endpoints. The last column includes a link to issue a command to view all OAI sets from that endpoint.

3.4.3 Transformation Scenario

Transformation Scenarios are used for transforming the XML of Records during Transformation Jobs. Currently, there are two types of well-supported transformation supported: **XSLT** and **Python code snippets**. A third type, transforming Records based on actions performed in [Open Refine](#) exists, but is not well tested or documented at this time. These are described in more detail below.

It is worth considering, when thinking about transforming Records in Combine, that multiple transformations can be applied to same Record; “chained” together as separate Jobs. Imagine a scenario where `Transformation A` crosswalks metadata from a repository to something more aligned with a state service hub, `Transformation B` fixes some particular date formats, and `Transformation C` – a python transformation – looks for a particular identifier field and creates a new field based on that. Each of the transformations would be a separate Transformation Scenario, and would be run as separate Jobs in Combine, but in effect would be “chained” together by the user for a group of Records.

All Transformations require the following information:

- `Name` - Human readable name for Transformation Scenario
- `Payload` - This is where the actual transformation code is added (more on the different types below)
- `Transformation Type` - `xslt` for XSLT transformations, or `python` for python code snippets
- `Filepath` - *This may be ignored* (in some cases, transformation payloads were written to disk to be used, but likely deprecated moving forward)

Finally, Transformation Scenarios may be tested within Combine over a pre-existing Record. This is done by clicking the “Test Transformation Scenario” button from Configuration page. This will take you to a screen that is similarly used for testing Transformations, Validations, and Record Identifier Transformations. For Transformations, it looks like the following:

In this screenshot, a few things are happening:

- a single Record has been clicked from the sortable, searchable table, indicating it will be used for the Transformation testing
- a *pre-existing* Transformation Scenario has been selected from the dropdown menu, automatically populating the payload and transformation type inputs
 - however, a user may also add or edit the payload and transformation types live here, for testing purposes

Django administration

Home › Core › Transformations › Add transformation

Add transformation

Name:

Payload:

Transformation type: -----

Filepath:

Fig. 31: Adding Transformation Scenario in Django admin screen

Home / Configuration

Test Validation Scenario

[Instructions](#)

Show 10 entries

ID	Combine ID	Record ID	Job	Originating OAI set	Unique	Document	Error	Validation Results
1593025	ee9c2d9-bd57-4533-a7e5-3266746d63e	175c998ba37632a06278484f8d4a736d	HarvestStaticOAIJob @ May. 08, 2018, 1:03:29 PM		Unique	Valid XML		Invalid
1593026	c83b3c1-9d62-481e-a81e-9d86c18d28914	2796c336287768c557c118c494a86c1	HarvestStaticOAIJob @ May. 08, 2018, 1:03:29 PM		Unique	Valid XML		Invalid
1593027	fe7a783a-2a68-42f5-8567-cfe7937477f	a81f4c53a377943a483a2d2a24a30c	HarvestStaticOAIJob @ May. 08, 2018, 1:03:29 PM		Unique	Valid XML		Invalid
1593028	ef7a993a-83a2-4bc7-8c7c-82a958f178b	5d08633795989b5682a676271fe773	HarvestStaticOAIJob @ May. 08, 2018, 1:03:29 PM		Unique	Valid XML		Invalid
1593029	8c575d84-9a32-453b-8884-896168b8785	273d8e78c4239843cc2334717c53	HarvestStaticOAIJob @ May. 08, 2018, 1:03:29 PM		Unique	Valid XML		Invalid
1593030	86ab358-c289-4293-b6a4-b6d53a76a82	f4a0d873d54a888d11d7f5843a353	HarvestStaticOAIJob @ May. 08, 2018, 1:03:29 PM		Unique	Valid XML		Invalid
1593031	278a723c-a80b-4c36-838b-d8b8a97a6f	7d37d51c17d58823c5f187888547c	HarvestStaticOAIJob @ May. 08, 2018, 1:03:29 PM		Unique	Valid XML		Invalid
1593032	173623a9-d888-8465-a7a6d84dc9c	b8a7a6d87772a688a8d8f18d48d6dc	HarvestStaticOAIJob @ May. 08, 2018, 1:03:29 PM		Unique	Valid XML		Invalid
1593033	81748a62-a8ad-4b4d-9c42-8a658c8c7f8	c3a8a8b35a47373a73673a73673a7367	HarvestStaticOAIJob @ May. 08, 2018, 1:03:29 PM		Unique	Valid XML		Invalid
1593034	c75dc875-225f-45cd-a6df-16c99b48086	a28c7a8b2b3c8c1f46a6e4a7ce18d5	HarvestStaticOAIJob @ May. 08, 2018, 1:03:29 PM		Unique	Valid XML		Invalid

Showing 1 to 10 of 3,231 entries

Previous 1 2 3 4 5 ... 324 Next

Transformation Payload

Paste/edit your XSLT or python script in the textbox below, or select from a pre-existing Transformation Scenario to test:

Select a pre-existing Validation Scenario

MODS to Service Hub profile

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0"
  xmlns:mods="http://www.loc.gov/mods/v3"
  >
  <xsl:output method="xml" indent="yes"/>

  <xsl:template match="/">
    <mods:mods>
      <xsl:call-template name="titleInfo"/>
      <xsl:call-template name="subject"/>
      <xsl:call-template name="abstract"/>
      <xsl:call-template name="location"/>
      <xsl:call-template name="date"/>
      <xsl:call-template name="type"/>
      <xsl:call-template name="physical"/>
    </mods:mods>
  </xsl:template>
</xsl:stylesheet>
```

Select Transformation Scenario type

XSLT

Test Transformation

Transformation Results

```
<?xml version="1.0" encoding="UTF-8"?>
<mods:mods xmlns:mods="http://www.loc.gov/mods/v3">
  <mods:titleInfo>
    <mods:title>Gund Dulac's fairy-book</mods:title>
```

Fig. 32: Testing Transformation Scenario with pre-existing Record

- at the very bottom, you can see the immediate results of the Transformation as applied to the selected Record

Currently, there is no way to save changes to a Transformation Scenario, or add a new one, from this screen, but it allows for real-time testing of Transformation Scenarios.

XSLT

XSLT transformations are performed by a small XSLT processor servlet called via `pyjxslt`. Pyjxslt uses a built-in Saxon HE XSLT processor that supports XSLT 2.0.

When creating an XSLT Transformation Scenario, one important thing to consider are XSLT **includes** and **imports**. XSL stylesheets allow the inclusion of other, external stylesheets. Usually, these includes come in two flavors:

- locally on the same filesystem, e.g. `<xsl:include href="mimeType.xml"/>`
- remote, retrieved via HTTP request, e.g. `<xsl:include href="http://www.loc.gov/standards/mods/inc/mimeType.xml"/>`

In Combine, the primary XSL stylesheet provided for a Transformation Scenario is uploaded to the pyjxslt servlet to be run by Spark. This has the effect of breaking XSL `include` s that use a **local, filesystem** href s. Additionally, depending on server configurations, pyjxslt sometimes has trouble accessing **remote** XSL `include` s. But Combine provides workarounds for both scenarios.

Local Includes

For XSL stylesheets that require local, filesystem `include` s, a workaround in Combine is to create Transformation Scenarios for each XSL stylesheet that is imported by the primary stylesheet. Then, use the local filesystem path that Combine creates for that Transformation Scenario, and **update** the `<xsl:include>` in the original stylesheet with this new location on disk.

For example, let's imagine a stylesheet called `DC2MODS.xml` that has the following `<xsl:include>` s:

```
<xsl:include href="dcmiType.xml"/>
<xsl:include href="mimeType.xml"/>
```

Originally, `DC2MODS.xml` was designed to be used in the *same directory* as two files: `dcmiType.xml` and `mimeType.xml`. This is not possible in Combine, as XSL stylesheets for Transformation Scenarios are uploaded to another location to be used.

The workaround, would be to create two new special kinds of Transformation Scenarios by checking the box `use_as_include`, perhaps with fitting names like “`dcmiType`” and “`mimeType`”, that have payloads for those two stylesheets. When creating those Transformation Scenarios, saving, and then re-opening the Transformation Scenario in Django admin, you can see a `Filepath` attribute has been made which is a copy written to disk.

This `Filepath` value can then be used to replace the original `<xsl:include>` s in the primary stylesheet, in our example, `DC2MODS.xml`:

```
<xsl:include href="/home/combine/data/combine/transformations/
→a436a2d4997d449a96e008580f6dc699.xml"/> <!-- formerly dcmiType.xml -->
<xsl:include href="/home/combine/data/combine/transformations/
→00eada103f6a422db564a346ed74c0d7.xml"/> <!-- formerly mimeType.xml -->
```

Remote Includes

When the href s for XSL `includes` s are remote HTTP URLs, Combine attempts to rewrite the primary XSL stylesheet automatically by:

Change transformation

Name:	dcmiType.xsl
Payload:	<pre><?xml version="1.0" encoding="UTF-8"?> <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"> <xsl:variable name="types"> <!-- DCMi Types (normalized versions appear in lower case and with spaces) http://dublincore.org/documents/dcmi-type-vocabulary/ --> <xsl:text>Collection</xsl:text> <xsl:text>collection</xsl:text> <xsl:text>Dataset</xsl:text> <xsl:text>dataset</xsl:text> </xsl:variable> </xsl:stylesheet></pre>
Transformation type:	XSLT Stylesheet
Filepath:	/home/combine/data/combine/transformations/a436a2d4997d449a96e008580f6dc699.xsl
<input checked="" type="checkbox"/> Use as include	

Fig. 33: Filepath for saved Transformation Scenarios

- downloading the external, remote `include`s from the primary stylesheet
- saving them locally
- rewriting the `<xsl:include>` element with this local filesystem location

This has the added advantage of effectively caching the remote `include`, such that it is not retrieved each transformation.

For example, let's imagine our trusty stylesheet called `DC2MODS.xsl`, but with this time external, remote URLs for `href`s:

```
<xsl:include href="http://www.loc.gov/standards/mods/inc/dcmiType.xsl"/>
<xsl:include href="http://www.loc.gov/standards/mods/inc/mimeType.xsl"/>
```

With no action by the user, when this Transformation Scenario is saved, Combine will attempt to download these dependencies and rewrite, resulting in `include`s that look like the following:

```
<xsl:include href="/home/combine/data/combine/transformations/dcmiType.xsl"/>
<xsl:include href="/home/combine/data/combine/transformations/mimeType.xsl"/>
```

Note: If stylesheets that remote `include`s rely on external stylesheets that may change or update, the primary Transformation stylesheet – e.g. `DC2MODS.xsl` – will have to be re-entered, with the original URLs, and re-saved in Combine to update the local dependencies.

Python Code Snippet

An alternative to XSLT transformations are created Transformation Scenarios that use python code snippets to transform the Record. The key to making a successful python Transformation Scenario is code that adheres to the pattern Combine is looking for from a python Transformation. This requires a bit of explanation about how Records are transformed in Spark.

For Transformation Jobs in Combine, each Record in the input Job is fed to the Transformation Scenario. If the `transformation type` is `xslt`, the XSLT stylesheet for that Transformation Scenario is used as-is on the Record's raw XML. However, if the `transformation type` is `python`, the python code provided for the Transformation Scenario will be used.

The python code snippet may include as many imports or function definitions as needed, but will require one function that each Record will be passed to, and this function must be named `python_record_transformation`. Additionally, this function must expect one function argument, a passed instance of what is called a `PythonUDFRecord`. In Spark, "UDF" often refers to a "User Defined Function"; which is precisely what this parsed Record instance is passed to in the case of a Transformation. This is a convenience class that parses a Record in Combine for easy interaction within Transformation, Validation, and Record Identifier Transformation Scenarios. A `PythonUDFRecord` instance has the following representations of the Record:

- `record_id` - The Record Identifier of the Record
- `document` - raw, XML for the Record (what is passed to XSLT records)
- `xml` - raw XML parsed with `lxml`'s `etree`, an `ElementTree` instance
- `nsmap` - dictionary of namespaces, useful for working with `self.xml` instance

Finally, the function `python_record_transformation` must return a python **list** with the following, ordered elements: [*transformed XML as a string, any errors if they occurred as a string, True/False for successful transformation*]. For example, a valid return might be, with the middle value a blank string indicating no error:

```
[ "<xml>...</xml>", "", True ]
```

A full example of a python code snippet transformation might look like the following. In this example, a `<mods:accessCondition>` element is added or updated. Note the imports, the comments, the use of the `PythonUDFRecord` as the single argument for the function `python_record_transformation`, all fairly commonplace python code:

```
# NOTE: ability to import libraries as needed
from lxml import etree

def python_record_transformation(record):
    '''
    Python transformation to add / update <mods:accessCondition> element
    '''

    # check for <mods:accessCondition type="use and reproduction">
    # NOTE: not built-in record.xml, parsed Record document as etree instance
    # NOTE: not built-in record.nsmap that comes with record instance
    ac_ele_query = record.xml.xpath('mods:accessCondition', namespaces=record.nsmap)

    # if single <mods:accessCondition> present
    if len(ac_ele_query) == 1:

        # get single instance
        ac_ele = ac_ele_query[0]

        # confirm type attribute
        if 'type' in ac_ele.attrib.keys():

            # if present, but not 'use and reproduction', update
            if ac_ele.attrib['type'] != 'use and reproduction':
                ac_ele.attrib['type'] = 'use and reproduction'
```

(continues on next page)

```

# if <mods:accessCondition> not present at all, create
elif len(ac_ele_query) == 0:

    # build element
    rights = etree.Element('{http://www.loc.gov/mods/v3}accessCondition')
    rights.attrib['type'] = 'use and reproduction'
    rights.text = 'Here is a blanket rights statement for our institution in the_
↳absence of a record specific one.'

    # append
    record.xml.append(rights)

# finally, serialize and return as required list [document, error, success (bool)]
return [etree.tostring(record.xml), '', True]

```

In many if not most cases, XSLT will fit the bill and provide the needed transformation in Combine. But the ability to write python code for transformation opens up the door to complex and/or precise transformations if needed.

3.4.4 Validation Scenario

Validation Scenarios are by which Records in Combine are validated against. Validation Scenarios may be written in the following formats: XML Schema (XSD), Schematron, Python code snippets, and ElasticSearch DSL queries. Each Validation Scenario requires the following fields:

- Name - human readable name for Validation Scenario
- Payload - pasted Schematron or python code
- Validation type - sch for Schematron, python for python code snippets, or es_query for ElasticSearch DSL query type validations
- Filepath - *This may be ignored* (in some cases, validation payloads were written to disk to be used, but likely deprecated moving forward)
- Default run - if checked, this Validation Scenario will be automatically checked when running a new Job

When running a Job, **multiple** Validation Scenarios may be applied to the Job, each of which will run for every Record. Validation Scenarios may include multiple tests or “rules” with a single scenario. So, for example, Validation A may contain Test 1 and Test 2. If run for a Job, and Record Foo fails Test 2 for the Validation A, the results will show the failure for that Validation Scenario as a whole.

When thinking about creating Validation Scenarios, there is flexibility in how many tests to put in a single Validation Scenario, versus splitting up those tests between distinct Validation Scenarios, recalling that **multiple** Validation Scenarios may be run for a single Job. It is worth pointing out, multiple Validation Scenarios for a Job will likely degrade performance *more* than a multiple tests within a single Scenario, though this has not been testing thoroughly, just speculation based on how Records are passed to Validation Scenarios in Spark in Combine.

Like Transformation Scenarios, Validation Scenarios may also be tested in Combine. This is done by clicking the button, “Test Validation Scenario”, resulting in the following screen:

In this screenshot, we can see the following happening:

- a single Record has been clicked from the sortable, searchable table, indicating it will be used for the Validation testing
- a pre-existing Validation Scenario – DPLA minimum, a Schematron validation – has been selected, automatically populating the payload and validation type inputs

Django administration

Home > Core > Validation scenarios > Add validation scenario

Add validation scenario

Name:

Payload:

Validation type:

Filepath:

☒ Default run

Fig. 34: Adding Validation Scenario in Django admin

Fig. 35: Testing Validation Scenario

- As mentioned, two types of Validation Scenarios are currently supported, Schematron and python code snippets, and are detailed below.

XML Schemas (XSD) may be used to validate a Record's document. One limitation of XML Schema is that many python based validators will bail on the first error encountered in a document, meaning the resulting Validation failure will only show the **first** invalid XML segment encountered, though there may be many. However, knowing that a Record has failed even one part of an XML Schema, might be sufficient to look in more detail with an external validator and determine where else it is invalid, or, fix that problem through a transform or re-harvest, and continue to run the XML Schema validations.

A valid [Schematron XML](#) document may be used as the Validation Scenario payload, and will validate the Record's raw XML. Schematron validations are rule-based, and can be configured to return the validation results as XML, which

is the case in Combine. This XML is parsed, and each distinct, defined test is noted and parsed by Combine.

Below is an example of a small Schematron validation that looks for some required fields in an XML document that would help make it DPLA compliant:

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://purl.oclc.org/dsdl/schematron" xmlns:mods="http://www.loc.gov/
↳mods/v3">
  <ns prefix="mods" uri="http://www.loc.gov/mods/v3"/>
  <!-- Required top level Elements for all records record -->
  <pattern>
    <title>Required Elements for Each MODS record</title>
    <rule context="mods:mods">
      <assert test="mods:titleInfo">There must be a title element</assert>
      <assert test="count(mods:location/mods:url[@usage='primary'])=1">There must be_
↳a url pointing to the item</assert>
      <assert test="count(mods:location/mods:url[@access='preview'])=1">There must be_
↳a url pointing to a thumbnail version of the item</assert>
      <assert test="count(mods:accessCondition[@type='use and reproduction'])=1">
↳There must be a rights statement</assert>
    </rule>
  </pattern>

  <!-- Additional Requirements within Required Elements -->
  <pattern>
    <title>Subelements and Attributes used in TitleInfo</title>
    <rule context="mods:mods/mods:titleInfo">
      <assert test="*">TitleInfo must contain child title elements</assert>
    </rule>
    <rule context="mods:mods/mods:titleInfo/*">
      <assert test="normalize-space(.)">The title elements must contain text</assert>
    </rule>
  </pattern>

  <pattern>
    <title>Additional URL requirements</title>
    <rule context="mods:mods/mods:location/mods:url">
      <assert test="normalize-space(.)">The URL field must contain text</assert>
    </rule>
  </pattern>
</schema>
```

Python Code Snippet

Similar to Transformation Scenarios, python code may also be used for the Validation Scenarios payload. When a Validation is run for a Record, and a python code snippet type is detected, all defined function names that begin with `test_` will be used as separate, distinct Validation tests. This very similar to how `pytest` looks for function names prefixed with `test_`. It is not perfect, but relatively simple and effective.

These functions must expect two arguments. The first is an instance of a `PythonUDFRecord`. As detailed above, `PythonUDFRecord` instances are a parsed, convenient way to interact with Combine Records. A `PythonUDFRecord` instance has the following representations of the Record:

- `record_id` - The Record Identifier of the Record
- `document` - raw, XML for the Record (what is passed to XSLT records)
- `xml` - raw XML parsed with lxml's etree, an `ElementTree` instance

- `nsmap` - dictionary of namespaces, useful for working with `self.xml` instance

The second argument is named and must be called `test_message`. The string value for the `test_message` argument will be used for reporting if that particular test if failed; this is the human readable name of the validation test.

All validation tests, recalling the name of the function must be prefixed with `test_`, must return `True` or `False` to indicate if the Record passed the validation test.

An example of an arbitrary Validation Scenario that looks for MODS titles longer than 30 characters might look like the following:

```
# note the ability to import (just for demonstration, not actually used below)
import re

def test_title_length_30(record, test_message="check for title length > 30"):

    # using PythonUDFRecord's parsed instance of Record with .xml attribute, and_
    ↪ namespaces from .nsmap
    titleInfo_elements = record.xml.xpath('//mods:titleInfo', namespaces=record.nsmap)
    if len(titleInfo_elements) > 0:
        title = titleInfo_elements[0].text
        if len(title) > 30:
            # returning False fails the validation test
            return False
        else:
            # returning True, passes
            return True

# note ability to define other functions
def other_function():
    pass

def another_function();
    pass
```

ElasticSearch DSL query

ElasticSearch DSL query type Validations Scenarios are a bit different. Instead of validating the document for a Record, ElasticSearch DSL validations validate by performing ElasticSearch queries against mapped fields for a Job, and marking Records as valid or invalid based on whether they are matches for those queries.

These queries may be written such that Records matches are **valid**, or they may be written where matches are **invalid**.

An example structure of an ElasticSearch DSL query might look like the following:

```
[
  {
    "test_name": "field foo exists",
    "matches": "valid",
    "es_query": {
      "query": {
        "exists": {
          "field": "foo"
        }
      }
    }
  }
]
```

(continues on next page)

(continued from previous page)

```

    }
  },
  {
    "test_name": "field bar does NOT have value 'baz'",
    "matches": "invalid",
    "es_query": {
      "query": {
        "match": {
          "bar.keyword": "baz"
        }
      }
    }
  }
]

```

This example contains **two** tests in a single Validation Scenario: checking for field `foo`, and checking that field `bar` does *not* have value `baz`. Each test must contain the following properties:

- `test_name`: name that will be returned in the validation reporting for failures
- `matches`: the string `valid` if matches to the query can be consider valid, or `invalid` if query matches should be considered invalid
- `es_query`: the raw, Elasticsearch DSL query

ElasticSearch DSL queries can support complex querying (boolean, and/or, fuzzy, regex, etc.), resulting in an additional, rich and powerful way to validate Records.

3.4.5 Record Identifier Transformation Scenario

Another configurable “Scenario” in Combine is a Record Identifier Transformation Scenario or “RITS” for short. A RITS allows the transformation of a Record’s “Record Identifier”. A Record has [three identifiers in Combine](#), with the Record Identifier (`record_id`) as the only changeable, mutable of the three. The Record ID is what is used for publishing, and for all intents and purposes, the unique identifier for the Record *outside* of Combine.

Record Identifiers are created during Harvest Jobs, when a Record is first created. This Record Identifier may come from the OAI server in which the Record was harvested from, it might be derived from an identifier in the Record’s XML in the case of a static harvest, or it may be minted as a UUID4 on creation. Where the Record ID is picked up from OAI or the Record’s XML itself, it might not need transformation before publishing, and can “go out” just as it “came in.” However, there are instances where transforming the Record’s ID can be quite helpful.

Take the following scenario. A digital object’s metadata is harvested from `Repository A` with the ID `foo`, as part of OAI set `bar`, by Metadata Aggregator A. Inside Metadata Aggregator A, which has its own OAI server prefix of `baz` considers the full identifier of this record: `baz:bar:foo`. Next, Metadata Aggregator B harvests this record from Metadata Aggregator A, under the OAI set `scrog`. Metadata Aggregator B has its own OAI server prefix of `tronic`. Finally, when a terminal harvester like DPLA harvests this record from Metadata Aggregator B under the set `goober`, it might have a motley identifier, constructed through all these OAI “hops” of something like: `tronic:scrog:goober:baz:bar:foo`.

If one of these hops were replaced by an instance of Combine, one of the OAI “hops” would be removed, and the dynamically crafted identifier for that same record would change. Combine allows the ability to transform the identifier – emulating previous OAI “hops”, completely re-writing, or any other transformation – through a Record Identifier Transformation Scenario (RITS).

RITS are performed, just like Transformation Scenarios or Validation Scenarios, for every Record in the Job. RITS may be in the form of:

- **Regular Expressions** - specifically, python flavored regex
- **Python code snippet** - a snippet of code that will transform the identifier
- **XPATH expression** - given the Record's raw XML, an XPath expression may be given to extract a value to be used as the Record Identifier

All RITS have the following values:

- `Name` - Human readable name for RITS
- `Transformation type` - `regex` for Regular Expression, `python` for Python code snippet, or `xpath` for XPath expression
- `Transformation target` - the RITS payload and type may use the pre-existing Record Identifier as input, or the Record's raw, XML record
- `Regex match payload` - If using regex, the regular expression to **match**
- `Regex replace payload` - If using regex, the regular expression to **replace** that match with (allows values from groups)
- `Python payload` - python code snippet, that will be passed an instance of a `PythonUDFRecord`
- `Xpath payload` - single XPath expression as a string

Payloads that do not pertain to the `Transformation type` may be left blank (e.g. if using python code snippet, regex match and replace payloads, and xpath payloads, may be left blank).

Similar to Transformation and Validation scenarios, RITS can be tested by clicking the “Test Record Identifier Transformation Scenario” button at the bottom. You will be presented with a familiar screen of a table of Records, and the ability to select a pre-existing RITS, edit that one, and/or create a new one. Similarly, without the ability to update or save a new one, merely to test the results of one.

These different types will be outline in a bit more detail below.

Regular Expression

If transforming the Record ID with regex, two “payloads” are required for the RITS scenario: a match expression, and a replace expression. Also of note, these regex match and replace expressions are the python flavor of regex matching, performed with python's `re.sub()`.

The screenshot belows shows an example of a regex match / replace used to replace `digital.library.wayne.edu` with `goober.tronic.org`, also highlighting the ability to use groups:

A contrived example, this shows a regex expression applied to the input Record identifier of `oai:digital.library.wayne.edu:wayne:Livingto1876b22354748``.

Python Code Snippet

Python code snippets for RITS operate similarly to Transformation and Validation scenarios in that the python code snippet is given an instance of a `PythonUDFRecord` for each Record. However, it differs slightly in that if the RITS `Transformation target` is the Record ID only, the `PythonUDFRecord` will have only the `.record_id` attribute to work with.

For a python code snippet RITS, a function named `transform_identifier` is required, with a single unnamed, passed argument of a `PythonUDFRecord` instance. An example may look like the following:

Django administration

Home › Core › Record identifier transformation scenarios › Add record identifier transformation scenario

Add record identifier transformation scenario

Name:	<input type="text"/>
Transformation type:	<input type="text" value="-----"/>
Transformation target:	<input type="text" value="-----"/>
Regex match payload:	<input type="text"/>
Regex replace payload:	<input type="text"/>
Python payload:	<div></div>
Xpath payload:	<input type="text"/>

Fig. 36: Adding Record Identifier Transformation Scenario (RITS)

Home / Configuration

Test Record Identifier Transformation Scenario

[Instructions](#)

Show entries

Search:

DB ID	Combine ID	Record ID	Job	Originating OAI set	Unique	Document	Error	Validation Results
1593775	5e2b11ea-7d39-49c1-8b4f-516431871357	oai:digital.library.wayne.edu:wayne:liv ingto1876b22354748	Ramsey Books collection	wayne.collection.ramsey	Unique	Valid XML		Invalid

Showing 1 to 1 of 1 entries (filtered from 3,231 total entries)

Previous Next

Optionally, select a pre-existing Record Identifier Transformation Scenario

Results

oai:goober.tronic.org:wayne:livingto1876b22354748

Select what will be used as input for the transformation process

Select the transformation type:

Regex match pattern:

Python regular expression syntax required

Regex replace pattern:

Python regular expression syntax required

[Test Record Identifier Transformation](#)

Fig. 37: Example of RITS with Regular Expression

```
# ability to import modules as needed (just for demonstration)
import re
import time

# function named `transform_identifier`, with single passed argument of _
↳ PythonUDFRecord instance
def transform_identifier(record):

    '''
    In this example, a string replacement is performed on the record identifier,
    but this could be much more complex, using a combination of the Record's parsed
    XML and/or the Record Identifier. This example is meant to show the structure of a
    python based RITS only.
    '''

    # function must return string of new Record Identifier
    return record.record_id.replace('digital.library.wayne.edu', 'goober.tronic.org')
```

And a screenshot of this RITS in action:

XPath Expression

Finally, a single XPath expression may be used to extract a new Record Identifier from the Record's XML record. **Note:** The input must be the Record's Document, not the current Record Identifier, as the XPath must have valid XML to retrieve a value from. Below is an example screenshot:

3.4.6 Combine OAI-PMH Server

Combine comes with a built-in OAI-PMH server to serve published Records. Configurations for the OAI server, at this time, are not configured with Django's admin, but may be found in `combine/localsettings.py`. These

Home / Configuration

Test Record Identifier Transformation Scenario

Instructions

Show 10 entries Search: 1593775

DB ID	Combine ID	Record ID	Job	Originating OAI set	Unique	Document	Error	Validation Results
1593775	5e2b11ee-7d39-49c1-88df-516431871357	oai:digital.library.wayne.edu:wayne:Liv ingto1876b22354748	Ramsey Books collection	wayne.collectionramsey	Unique	Valid XML		Invalid

Showing 1 to 1 of 1 entries (filtered from 3,231 total entries) Previous 1 Next

Optionally, select a pre-existing Record Identifier Transformation Scenario

python test

Select what will be used as input for the transformation process

Record's Identifier

Select the transformation type:

Python Code Snippet

Python code for transformation:

A function named `transform_identifier(record)` is required, with the single argument `record` containing the properties outlined above

```
def transform_identifier(record):
    return
    record.record_id.replace('digital.library.wayne.edu', 'goober.tronic.org')
```

Test Record Identifier Transformation

Results

oai:goober.tronic.org:wayne:LivIngto1876b22354748

Fig. 38: Example of RITS with Python code snippet

Home / Configuration

Test Record Identifier Transformation Scenario

Instructions

Show 10 entries Search: 1593775

DB ID	Combine ID	Record ID	Job	Originating OAI set	Unique	Document	Error	Validation Results
1593775	5e2b11ee-7d39-49c1-88df-516431871357	oai:digital.library.wayne.edu:wayne:Liv ingto1876b22354748	Ramsey Books collection	wayne.collectionramsey	Unique	Valid XML		Invalid

Showing 1 to 1 of 1 entries (filtered from 3,231 total entries) Previous 1 Next

Optionally, select a pre-existing Record Identifier Transformation Scenario

Select a pre-existing scenario...

Select what will be used as input for the transformation process

Record's XML document

Select the transformation type:

XPath Expression

XPath expression:

Only works with "Record's XML document" as input

```
//mods:mods/mods:identifier[@type="local"]
```

Test Record Identifier Transformation

Results

LivIngto1876b22354748

Fig. 39: Example of RITS with XPath expression

settings include:

- `OAI_RESPONSE_SIZE` - How many records to return per OAI paged response
- `COMBINE_OAI_IDENTIFIER` - It is common for OAI servers (producers) to prefix Record identifiers on the way out with an identifier unique to the server. This setting can also be configured to mirror the identifier used in other/previous OAI servers to mimic downstream identifiers

3.4.7 DPLA Bulk Data Downloads (DBDD)

One of the more experimental features of Combine is to compare the Records from a Job (or, of course, multiple Jobs if they are [Merged into one](#)) against a [bulk data download from DPLA](#).

To use this function, S3 credentials must be added to the `combine/localsettings.py` settings file that allow for downloading of bulk data downloads from S3. Once added, and Combine restarted, it is possible to download previous bulk data dumps. This can be done from the configuration page by clicking on “Download and Index Bulk Data”, then selecting a bulk data download from the long dropdown. When the button is clicked, this data set will be downloaded and indexed locally in ElasticSearch, all as a background task. This will be reflected in the table on the Configuration page as complete when the row reads “Downloaded and Indexed”:

DPLA Bulk Data Downloads

Manage local, [bulk data downloads from DPLA](#).

ID	S3 Bucket Key	Status	Uploaded to S3	Downloaded to Combine	ElasticSearch index	Filepath
16	2018/04/michigan_json.gz	Downloaded and Indexed	April 1, 2018, 12:23 p.m.	April 16, 2018, 5:53 p.m.	f00755b7dabfe04114902fcfd28cdc1	/home/combine/data/combine/bulk/2018_04_michigan_json.gz

[Download and Index Bulk Data](#)

Fig. 40: Downloaded and Indexed DPLA Bulk Data Download (DBDD)

Comparison can be triggered from any [Job’s optional parameters](#) under the tab [DPLA Bulk Data Compare](#). Comparison is performed by attempting to match a Record’s Record Identifier to the `_id` field in the DPLA Item document.

Because this comparison is using the Record Identifier for matching, this is a great example of where a Record Identifier Transformation Scenario (RITS) can be a powerful tool to emulate or recreate a known or previous identifier pattern. So much so, it’s conceivable that passing a RITS along with the DPLA Bulk Data Compare – just to temporarily transform the Record Identifier for comparison’s sake, but not in the Combine Record itself – might make sense.

3.5 Workflows and Viewing Results

This section will describe different parts of workflows for running, viewing detailed results, and exporting Jobs and Records.

Sub-sections include:

- [Record Versioning](#)
- [Running Jobs](#)
- [Viewing Job Details](#)
- [Viewing Record Details](#)
- [Managing Jobs](#)

3.5.1 Record Versioning

In an effort to preserve various stages of a Record through harvest, possible multiple transformation, merges and sub-dividing, Combine takes the approach of copying the Record each time.

As outlined in the [Data Model](#), Records are represented in both MongoDB and ElasticSearch. Each time a Job is run, and a Record is duplicated, it gets a new document in Mongo, with the full XML of the Record duplicated. Records are associated with each other across Jobs by their [Combine ID](#).

This approach has pros and cons:

- Pros
 - simple data model, each version of a Record is stored separately
 - each Record stage can be indexed and analyzed separately
 - Jobs containing Records can be deleted without effecting up/downstream Records (they will vanish from the lineage)
- Cons
 - duplication of data is potentially unnecessary if Record information has not changed

3.5.2 Running Jobs

Note: For all Jobs in Combine, confirm that an [active Livy session is up and running](#) before proceeding.

All Jobs are tied to, and initiated from, a [Record Group](#). From the Record Group page, at the bottom you will find buttons for starting new jobs:

Run New Job		
Harvest	Harvests Jobs are how Records are created and introduced to a Record Group. Currently this includes OAI-PMH harvesting, or ingesting Records from static files on disk.	Harvest OAI-PMH Harvest Static XML
Transform	Transformation Jobs change the Records within a Job some way. Currently this includes transformation of the Record's XML document with XSLT or python code snippets. Before running a Transformation, make a pre-configured Transformation Scenario exists.	Transform Job
Duplicate / Merge	Duplicate / Merge Jobs do not modify the Record, but can be used to combine multiple Jobs into one Job, run new or different Validation Scenarios on a pre-existing Job, or index to ElasticSearch with a different mapping.	Duplicate / Merge Jobs

Fig. 41: Buttons on a Record Group to begin a Job

Clicking any of these Job types will initiate a new Job, and present you with the options outlined below.

Optional Parameters

When running any type of Job in Combine, you are presented with a section near the bottom for **Optional Parameters** for the job:

These options are split across various tabs, and include:

- *Record Input Filters*
- *Field Mapping Configuration*
- *Validation Tests*
- *Transform Identifier*
- *DPLA Bulk Data Compare*

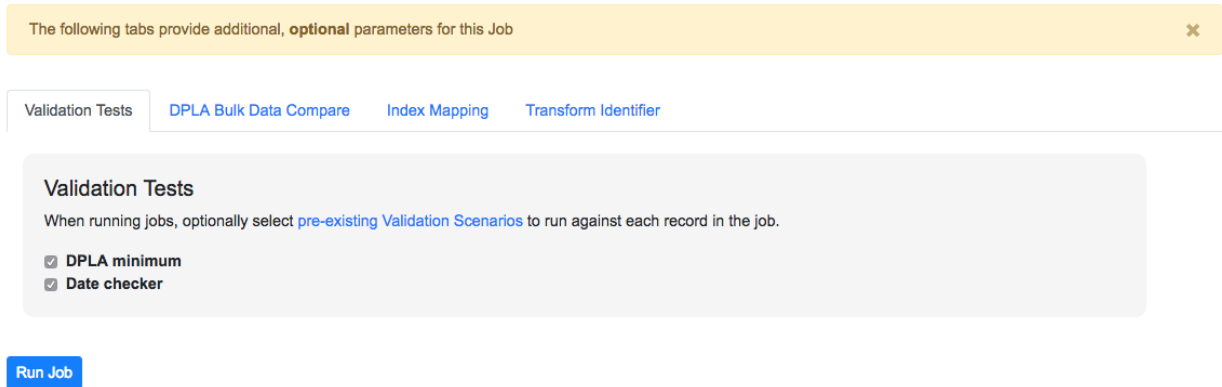


Fig. 42: Optional Parameters for all Jobs

For the most part, a user is required to pre-configure these in the [Configurations section](#), and then select which optional parameters to apply during runtime for Jobs.

Record Input Filters

When running a new Transform or Duplicate/Merge Job, which both rely on other Jobs as Input Jobs, filters can be applied to filter incoming Records. These filters are settable via the “Record Input Filter” tab.

There are two ways in which filters can be applied:

- “Globally”, where all filters are applied to all Jobs
- “Job Specific”, where a set of filters can be applied to individual Jobs, overriding any “Global” filters

Setting filters for individual Jobs is performed by clicking the filter icon next to a Job’s checklist in the Input Job selection table:

This will bring up a modal window where filters can be set for that Job, and that Job only. When the modal window is saved, and filters applied to that Job, the filter icon will turn orange indicating that Job has unique filters applied:

When filters are applied to specific Jobs, this will be reflected in the Job lineage graph:

and the Input Jobs tab for the Job as well:

Currently, the following input Record filters are supported:

- Filter by Record Validity
- Limit Number of Records
- Filter Duplicates
- Filter by Mapped Fields

Filter by Record Validity

Users can select if **all**, **valid**, or **invalid** Records will be included.

Below is an example of how those valves can be applied and utilized with Merge Jobs to select only only valid or invalid records:

Keep in mind, if multiple Validation Scenarios were run for a particular Job, it only requires failing one test, within one Validation Scenario, for the Record to be considered “invalid” as a whole.

Limit Number of Records



Fig. 43: Click the filter button to set filters for a specific Job

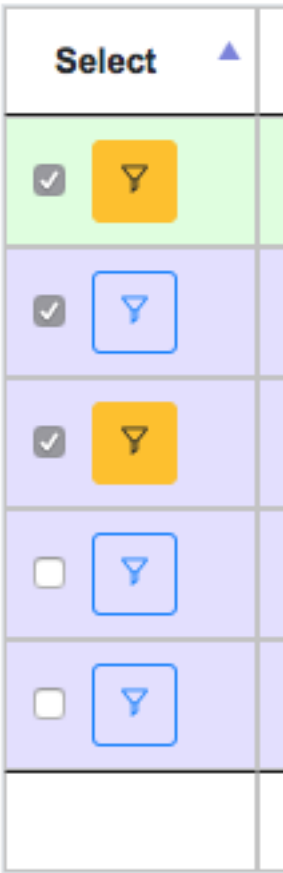


Fig. 44: Orange filter buttons indicate filters have been set for a specific Job



Fig. 45: Job lineage showing Job specific filters applied

Input Job ID	Input Job Name	Job Type	Job Specific Filters Applied?	Validity	De-Dupe Records	Mapped Field Query Filtered	Numerical Limit	Total Passed Records
570	MergeJob @ Sep. 27, 2018, 6:46:57 PM	MergeJob	True	Failed Validation	False	None	None	64
571	MergeJob @ Sep. 27, 2018, 6:47:03 PM	MergeJob	True	Ignored Validation	True	{"query":{"match":{"mods_subject_topic":"Pamphlets"}}}	None	8
Total:								72

Fig. 46: Job lineage showing Job specific filters applied

Record Input Validity

From the options below, select what kind of records from the selected input jobs will be used: **all** records, those that **passed** validation, or those that **failed**.

- ☒ All records (default)
- ☐ Passed validation
- ☐ Failed validation

Fig. 47: Selecting Record Input Validity Valve for Job



Fig. 48: Example of shunting Records based on validity, and eventually merging all valid Records

Arguably the simplest filter, users can provide a number to limit **total** number of Records that will be used as input. This numerical filter is applied after other filters have been applied, and the Records from each Input Job have been mixed. Given Input Jobs A, B, and C, all with 1,000 Records, given a numerical limit of 50, it's quite possible that all 50 will come from Job A, and 0 from B and C.

This filter is likely most helpful for testing and sampling.

Filter Duplicates

Optionally, remove duplicate Records based on matching `record_id` values. As these are used for publishing, this can be a way to ensure that Records are not published with duplicate `record_id`.

Filter by Mapped Fields

Users can provide an Elasticsearch DSL query, as JSON, to refine the records that will be used for this Job.

Take, for example, an input Job of 10,000 Records that has a field `foo_bar`, and 500 of those Records have the value `baz` for this field. If the following query is entered here, only the 500 Records that are returned from this query will be used for the Job:

```
{
  "query": {
    "match": {
      "foo_bar": "baz"
    }
  }
}
```

This ability hints at the potential for taking the time to map fields in interesting and helpful ways, such that you can use those mapped fields to refine later Jobs by. Elasticsearch queries can be quite powerful and complex, and in theory, this filter will support any query used.

Field Mapping Configuration

Combine maps a Record's original document – likely XML – to key/value pairs suitable for Elasticsearch with a library called XML2kvp. When running a new Job, users can provide parameters to the XML2kvp parser in the form of JSON.

Here's an example of the default configurations:

```
{
  "add_literals": {},
  "concat_values_on_all_fields": false,
  "concat_values_on_fields": {},
  "copy_to": {},
  "copy_to_regex": {},
  "copy_value_to_regex": {},
  "error_on_delims_collision": false,
  "exclude_attributes": [],
  "exclude_elements": [],
  "include_all_attributes": false,
  "include_attributes": [],
  "node_delim": "-",
  "ns_prefix_delim": "|",
  "remove_copied_key": true,
  "remove_copied_value": false,
  "remove_ns_prefix": false,
  "self_describing": false,
  "skip_attribute_ns_declarations": true,
  "skip_repeating_values": true,
  "split_values_on_all_fields": false,
  "split_values_on_fields": {}
}
```

Clicking the button “What do these configurations mean?” will provide information about each parameter, pulled from the XML2kvp JSON schema.

The default is a safe bet to run Jobs, but configurations can be **saved**, **retrieved**, **updated**, and **deleted** from this screen as well.

Additional, high level discussion about mapping and indexing metadata [can also be found here](#).

Validation Tests

One of the most commonly used optional parameters would be what Validation Scenarios to apply for this Job. Validation Scenarios are [pre-configured validations](#) that will run for *each* Record in the Job. When viewing a Job's or Record's details, the result of each validation run will be shown.

The Validation Tests selection looks like this for a Job, with checkboxes for each pre-configured Validation Scenarios (additionally, checked if the Validation Scenario is marked to run by default):

Transform Identifier

When running a Job, users can optionally select a [Record Identifier Transformation Scenario \(RITS\)](#) that will modify the Record Identifier for each Record in the Job.

Validation Tests

When running jobs, optionally select [pre-existing Validation Scenarios](#) to run against each record in the job.

☒ DPLA minimum
☐ Date checker
☐ title > 30

Fig. 49: Selecting Validations Tests for Job

Transform Identifier

For Records in this Job, optionally transform the Record's Identifier (`record_id`) -- used for publishing and uniqueness checks -- with a [pre-existing Record Identifier Transformation Scenario \(RITS\)](#).

Select a Record Identifier Transformation Scenario...

Fig. 50: Selecting Record Identifier Transformation Scenario (RITS) for Job

DPLA Bulk Data Compare

One somewhat experimental feature is the ability to compare the Record's from a Job against a downloaded and indexed bulk data dump from DPLA. These DPLA bulk data downloads can be [managed in Configurations here](#).

When running a Job, a user may optionally select what bulk data download to compare against:

DPLA Bulk Data Compare

If bulk DPLA data dumps have been [downloaded and indexed](#), check the existence of Record identifiers from this Job against those in the selected data dump to see if they exist. If they do, it can be assumed that Record existed in DPLA at the time of the data dump.

2018/04/michigan.json.gz

Fig. 51: Selecting DPLA Bulk Data Download comparison for Job

3.5.3 Viewing Job Details

One of the most detail rich screens are the results and details from a Job run. This section outlines the major areas. This is often referred to as the “Job Details” page.

At the very top of an Job Details page, a user is presented with a “lineage” of input Jobs that relate to this Job:

Also in this area is a button “Job Notes” which will reveal a panel for reading / writing notes for this Job. These notes will also show up in the Record Group's Jobs table.

Below that are tabs that organize the various parts of the Job Details page:

Job Details: **MergeJob @ May. 10, 2018, 1:19:51 PM** [Job Notes](#)

Fig. 52: Lineage of input Jobs for a Job

- *Records*
- *Mapped Fields*
- *Re-Run*
- *Publish*
- *Input Jobs*
- *Validation*
- *DPLA Bulk Data Matches*
- *Job Type Details - Jobs*
- *Exporting*
- *Spark Details*

Records

Show entries

DB ID	Combine ID	Record ID	Originating OAI set	Unique	Document	Error	Validation Results
1594974	5e2b11ea-7d39-49c1-880f-516431871357	oai:digital.library.wayne.edu:wayne:Livingto1876b22354748	wayne:collectionramsey	Unique	Valid XML		Invalid
1594975	2251821a-a358-45c6-a109-8ee450808864	oai:digital.library.wayne.edu:wayne:RENCEN18s	wayne:collectionrencen	Unique	Valid XML		Invalid
1594976	c99e4bda-fa8a-4366-bc85-77e2ca7698db	oai:digital.library.wayne.edu:wayne:Hawatha1918621559680	wayne:collectionramsey	Unique	Valid XML		Invalid
1594977	e4187473-6ee8-43da-b137-6b805fc69976	oai:digital.library.wayne.edu:wayne:LittleLu1825621567780	wayne:collectionramsey	Unique	Valid XML		Invalid
1594978	0baea586-fa42-46a7-9e06-488a3783c4de	oai:digital.library.wayne.edu:wayne:RENCEN80b	wayne:collectionrencen	Unique	Valid XML		Invalid
1594979	416d9991-6e08-4b08-acf0-1f831ealc65c	oai:digital.library.wayne.edu:wayne:RENCEN16d	wayne:collectionrencen	Unique	Valid XML		Invalid
1594980	8d8e893c-9e27-4d7d-85e9-c454cf6b84eb	oai:digital.library.wayne.edu:wayne:RENCEN29d	wayne:collectionrencen	Unique	Valid XML		Invalid
1594981	059c8265-d275-44e3-b897-bbb4cd95be1a	oai:digital.library.wayne.edu:wayne:Snythergen1923b48015817	wayne:collectionramsey	Unique	Valid XML		Invalid
1594982	27859727-4fae-46ad-a202-4ff4dac7beeb	oai:digital.library.wayne.edu:wayne:Adventur1868b59881974	wayne:collectionramsey	Unique	Valid XML		Invalid
1594983	6a6278fa-c385-44cc-9993-45d884af8296	oai:digital.library.wayne.edu:wayne:RENCEN85c	wayne:collectionrencen	Unique	Valid XML		Invalid

Showing 1 to 10 of 761 entries

Previous 2 3 4 5 ... 77 Next

Fig. 53: Table of all Records from a Job

This table shows all Records for this Job. It is sortable and searchable (though limited to what fields), and contains the following fields:

- **DB ID** - Record's ObjectID in MongoDB
- **Combine ID** - identifier assigned to Record on creation, sticks with Record through all stages and Jobs
- **Record ID** - Record identifier that is acquired, or created, on Record creation, and is used for publishing downstream. This may be modified across Jobs, unlike the **Combine ID**.
- **Originating OAI set** - what OAI set this record was harvested as part of
- **Unique** - True/False if the **Record ID** is unique in this Job

- **Document** - link to the Record's raw, XML document, blank if error
- **Error** - explanation for error, if any, otherwise blank
- **Validation Results** - True/False if the Record passed *all* Validation Tests, True if none run for this Job

In many ways, this is the most direct and primary route to access Records from a Job.

Mapped Fields

This tab provides a table of all indexed fields for this job, the nature of which is covered in more detail [here](#):

Total Records for Job	Successfully Indexed	Actions					
250	250	Show Field Mapping Configuration Used Browse Elasticsearch Index View Mapping and Indexing Errors Re-Map and Re-Index Fields					

What do these numbers mean?

Show 10 entries

Field Name	DPLA Mapped Field	Map DPLA Field	Documents with Field (of total 250)	Documents without	Count of Total Values for Field	Count of Distinct Values for Field	Percentage of Field Values that are Unique	Percentage of Documents with Field
nods_abstract		Select DPLA f	111	139	98	96	98%	44%
nods_accessCondition		Select DPLA f	250	0	250	1	0%	100%
nods_language_languageTerm		Select DPLA f	250	0	250	10	4%	100%
nods_location_url_access=review		Select DPLA f	250	0	250	249	100%	100%
nods_location_url_usage=primary		Select DPLA f	250	0	250	255	102%	100%
nods_name_namePart		Select DPLA f	221	29	316	236	75%	88%
nods_name_role_roleTerm		Select DPLA f	207	43	210	2	1%	83%
nods_note		Select DPLA f	197	53	268	50	19%	79%
nods_originInfo_dateIssued		Select DPLA f	249	1	249	107	43%	100%
nods_physicalDescription_extent		Select DPLA f	250	0	250	190	76%	100%

Showing 1 to 10 of 27 entries

Previous 1 2 3 Next

Fig. 54: Indexed field analysis for a Job, across all

Re-Run

Jobs can be re-run “in place” such that all current parameters, applied scenarios, and linkages to other jobs are maintained. All “downstream” Jobs – Jobs that inherit Records from this Job – are also automatically re-run.

One way to think about re-running Jobs would be to think of a group of Jobs that that inherit Records from one another as a “pipeline”.

Jobs may also be re-run, as well as in bulk with other Jobs, from a Record Group page.

More information can be found here: [Re-Running Jobs documentation](#).

Publish

This tab provides the means of publishing a single Job and its Records. This is covered in more detail in the [Publishing section](#).

Input Jobs

This table shows all Jobs that were used as *input* Jobs for this Job.

Records Field Analysis Input Jobs Validation DPLA Bulk Data Matches

Input Jobs

The following Jobs were used as input for this MergeJob.

Input Job Name	Job Type	Record Input Type	Record Count
Ramsey Books collection	HarvestOAIJob	All Records	438
Ransom Collection	HarvestOAIJob	All Records	323
		Total:	761

Fig. 55: Table of Input Jobs used for this Job

Validation

This tab shows the results of all Validation tests run for this Job:

Records Field Analysis Input Jobs Validation DPLA Bulk Data Matches

Validation Scenarios

The following Validation Scenarios were run for this job:

Validation Name	Validation Type	Record Validation Failure Count	Actions
DPLA minimum	Schematron	761	See Failures

Run validation results report

Fig. 56: Results of all Validation Tests run for this Job

For each Validation Scenario run, the table shows the name, type, count of records that failed, and a link to see the failures in more detail.

More information about [Validation Results](#) can be found [here](#).

DPLA Bulk Data Matches

If a DPLA bulk data download was selected to compare against for this Job, the results will be shown in this tab.

The following screenshot gives a sense of what this looks like for a Job containing about 250k records, that was compared against a DPLA bulk data download of comparable size:

This feature is still somewhat exploratory, but Combine provides an ideal environment and “moment in time” within the greater metadata aggregation ecosystem for this kind of comparison.

In this example, we are seeing that 185k Records were found in the DPLA data dump, and that 38k Records appear to be new. Without an example at hand, it is difficult to show, but it’s conceivable that by leaving Jobs in Combine, and then comparing against a later DPLA data dump, one would have the ability to confirm that all records do indeed show up in the DPLA data.

Spark Details

This tab provides helpful diagnostic information about the Job as run in in the background in Spark.

Spark Jobs/Tasks Run

Shows the actual tasks and stages as run by Spark. Due to how Spark runs, the names of these tasks may not be familiar or immediately obvious, but provide a window into the Job as it runs. This section also shows additional tasks that have been run for this Job such as re-indexing, or new validations.

Livy Statement Information

[Records](#)
[Field Analysis](#)
[Input Jobs](#)
[Validation](#)
[DPLA Bulk Data Matches](#)

The following tables shows matches and misses based on the Record's Identifier as run against the selected DPLA Bulk Data dump.

DPLA Bulk Data Download	Total Records in Job	Matches	Misses
2018/04/michigan.json.gz	224169	185430	38739

Matches

Show 10 entries Search:

DB ID	Record ID
2825583	mi--oai:michigan.yhsc1:0ai:quod.lib.umich.edu:IC-YHSIC1-X-99%SDYHS00099.TIF
2787099	mi--oai:michigan.yhsc1:0ai:quod.lib.umich.edu:IC-YHSIC1-X-98%SDYHS00098.TIF
2813217	mi--oai:michigan.yhsc1:0ai:quod.lib.umich.edu:IC-YHSIC1-X-97%SDYHS00097.TIF
2665229	mi--oai:michigan.yhsc1:0ai:quod.lib.umich.edu:IC-YHSIC1-X-96%SDYHS00096.TIF
2711235	mi--oai:michigan.yhsc1:0ai:quod.lib.umich.edu:IC-YHSIC1-X-95%SDYHS00095.TIF
2730170	mi--oai:michigan.yhsc1:0ai:quod.lib.umich.edu:IC-YHSIC1-X-94%SDYHS00094.TIF
2840082	mi--oai:michigan.yhsc1:0ai:quod.lib.umich.edu:IC-YHSIC1-X-93%SDYHS00093.TIF
2791636	mi--oai:michigan.yhsc1:0ai:quod.lib.umich.edu:IC-YHSIC1-X-92%SDYHS00092.TIF
2881659	mi--oai:michigan.yhsc1:0ai:quod.lib.umich.edu:IC-YHSIC1-X-91%SDYHS00091.TIF
2783675	mi--oai:michigan.yhsc1:0ai:quod.lib.umich.edu:IC-YHSIC1-X-90%SDYHS00090.TIF

Showing 1 to 10 of 185,430 entries Previous 1 2 3 4 5 ... 18543 Next

Misses

Show 10 entries Search:

DB ID	Record ID
2736894	mi--oai:michigan.wsuai:0ai:digital.library.wayne.edu:wsudor_dpla:0ai:digital.library.wayne.edu:wayne:wpa_909-B
2827809	mi--oai:michigan.wsuai:0ai:digital.library.wayne.edu:wsudor_dpla:0ai:digital.library.wayne.edu:wayne:wpa_675
2766958	mi--oai:michigan.wsuai:0ai:digital.library.wayne.edu:wsudor_dpla:0ai:digital.library.wayne.edu:wayne:wpa_674-C
2758151	mi--oai:michigan.wsuai:0ai:digital.library.wayne.edu:wsudor_dpla:0ai:digital.library.wayne.edu:wayne:wpa_31-A
2705684	mi--oai:michigan.wsuai:0ai:digital.library.wayne.edu:wsudor_dpla:0ai:digital.library.wayne.edu:wayne:wpa_164-B

Fig. 57: Results of DPLA Bulk Data Download comparison

This section shows the raw JSON output from the Job as submitted to Apache Livy.

Job Type Details - Jobs

For each Job type – Harvest, Transform, Merge/Duplicate, and Analysis – the Job details screen provides a tab with information specific to that Job type.

All Jobs contain a section called **Job Runtime Details** that show all parameters used for the Job:

OAI Harvest Jobs

Shows what OAI endpoint was used for Harvest.

Static Harvest Jobs

No additional information at this time for Static Harvest Jobs.

Transform Jobs

The “Transform Details” tab shows Records that were transformed during the Job in some way. For some Transformation Scenarios, it might be assumed that all Records will be transformed, but others, may only target a few Records. This allows for viewing what Records were altered.

Clicking into a Record, and then clicking the “Transform Details” tab at the Record level, will show detailed changes for that Record (see below for more information).

Merge/Duplicate Jobs

No additional information at this time for Merge/Duplicate Jobs.

Analysis Jobs

Combine Documentation

Records	Mapped Fields	Publish	Input Jobs	Validation	DPLA Bulk Data Matches	Merge / Duplicate Details	Export	Spark Details
Livy Statement	Livy Session Logs	Spark Application GUI	Spark Cluster GUI					

Spark Jobs/Tasks Run

ID	Description	Name	Status	Completed / Total Tasks	Submitted	Completed	Duration
25	Indexing to ElasticSearch, Job #248	saveAtoNewAPIHadoopFile at PythonRDD.scala:834	SUCCEEDED	8/8	2018-08-01T14:35:45.988GMT	2018-08-01T14:35:47.999GMT	0:00:02
24	Indexing to ElasticSearch, Job #248	take at SerDeUtil.scala:233	SUCCEEDED	1/1	2018-08-01T14:35:45.887GMT	2018-08-01T14:35:45.950GMT	0:00:00
23	Indexing to ElasticSearch, Job #248	runJob at PythonRDD.scala:441	SUCCEEDED	3/3	2018-08-01T14:35:45.367GMT	2018-08-01T14:35:45.551GMT	0:00:00
22	Indexing to ElasticSearch, Job #248	runJob at PythonRDD.scala:441	SUCCEEDED	4/4	2018-08-01T14:35:45.163GMT	2018-08-01T14:35:45.354GMT	0:00:00
21	Indexing to ElasticSearch, Job #248	runJob at PythonRDD.scala:441	SUCCEEDED	1/1	2018-08-01T14:35:45.013GMT	2018-08-01T14:35:45.147GMT	0:00:00
20	Running Merge/Duplicate Job, Job #248	jdbc at NativeMethodAccessorImpl.java:0	SUCCEEDED	408/408	2018-08-01T14:35:33.849GMT	2018-08-01T14:35:44.953GMT	0:00:11

Livv Statement Information

```

code : from jobs import MergeSpark as MergeSpark(spark, input_jobs_ids=["246, 247"], job_id="248", fm_config_json="{\"add_literals\":{\"capture_attribute_values\":{\"concat_values_on_all_fields\":false,\"concat_values_on_fields\":{\"copy_to\":{\"copy_to_regexp\":{\"copy_value_to_regexp\":{\"error_on_delim_collision\":false,\"exclude_attributes\":{\"exclude_elements\":{\"include_all_attributes\":false,\"include_attributes\":{\"nodes_delim\":\"\\n\",\"ns_prefix_delim\":\"\\n\"},\"remove_copied_key\":true,\"remove_copied_value\":false,\"remove_ns_prefix\":true,\"self_describing\":false,\"skip_attribute_ns_declarations\":true,\"skip_repeating_values\":true,\"split_values_on_all_fields\":false,\"split_values_on_fields\":{\"}}\"}, validation_scenarios\":{\"}}, rits=None, input_filters={\"input_numerical_value\": None, \"filter_dupe_record_ids\": false, \"input_validity_value\": \"all\", \"input_es_query_value\": None}, dbds=None).spark_function()

id : 3

▼ output : {3}

▼ data : {1}

text/plain : /usr/local/anaconda/envs/combine/lib/python3.5/site-packages/django/db/models/fields/__init__.py:1451: RuntimeWarning: DateTimeField JobTrack.finish_timestamp received a naive datetime (2018-08-01 14:35:48.456427) while time zone support is active.\n RuntimeWarning)

execution_count : 3

status : ok

progress : 1

state : available

```

Fig. 58: Details about the Job as run in Apache Spark

Job Runtime Details

```

1 2
3 4
5 6
7 8
9 10
11 12
13 14
15 16
17 18
19 20
21 22
23 24
25 26
27 28
29 30
31 32
33 34
35 36
37 38
39 40
41 42
43 44
45 46
47 48
49 50
51 52
53 54
55 56
57 58
59 60
61 62
63 64
65 66
67 68
69 70
71 72
73 74
75 76
77 78
79 80
81 82
83 84
85 86
87 88
89 90
91 92
93 94
95 96
97 98
99 100
101 102
103 104
105 106
107 108
109 110
111 112
113 114
115 116
117 118
119 120
121 122
123 124
125 126
127 128
129 130
131 132
133 134
135 136
137 138
139 140
141 142
143 144
145 146
147 148
149 150
151 152
153 154
155 156
157 158
159 160
161 162
163 164
165 166
167 168
169 170
171 172
173 174
175 176
177 178
179 180
181 182
183 184
185 186
187 188
189 190
191 192
193 194
195 196
197 198
199 200
201 202
203 204
205 206
207 208
209 210
211 212
213 214
215 216
217 218
219 220
221 222
223 224
225 226
227 228
229 230
231 232
233 234
235 236
237 238
239 240
241 242
243 244
245 246
247 248
249 250
251 252
253 254
255 256
257 258
259 260
261 262
263 264
265 266
267 268
269 270
271 272
273 274
275 276
277 278
279 280
281 282
283 284
285 286
287 288
289 290
291 292
293 294
295 296
297 298
299 300
301 302
303 304
305 306
307 308
309 310
311 312
313 314
315 316
317 318
319 320
321 322
323 324
325 326
327 328
329 330
331 332
333 334
335 336
337 338
339 340
341 342
343 344
345 346
347 348
349 350
351 352
353 354
355 356
357 358
359 360
361 362
363 364
365 366
367 368
369 370
371 372
373 374
375 376
377 378
379 380
381 382
383 384
385 386
387 388
389 390
391 392
393 394
395 396
397 398
399 400
401 402
403 404
405 406
407 408
409 410
411 412
413 414
415 416
417 418
419 420
421 422
423 424
425 426
427 428
429 430
431 432
433 434
435 436
437 438
439 440
441 442
443 444
445 446
447 448
449 450
451 452
453 454
455 456
457 458
459 460
461 462
463 464
465 466
467 468
469 470
471 472
473 474
475 476
477 478
479 480
481 482
483 484
485 486
487 488
489 490
491 492
493 494
495 496
497 498
499 500
501 502
503 504
505 506
507 508
509 510
511 512
513 514
515 516
517 518
519 520
521 522
523 524
525 526
527 528
529 530
531 532
533 534
535 536
537 538
539 540
541 542
543 544
545 546
547 548
549 550
551 552
553 554
555 556
557 558
559 560
561 562
563 564
565 566
567 568
569 570
571 572
573 574
575 576
577 578
579 580
581 582
583 584
585 586
587 588
589 590
591 592
593 594
595 596
597 598
599 600
601 602
603 604
605 606
607 608
609 610
611 612
613 614
615 616
617 618
619 620
621 622
623 624
625 626
627 628
629 630
631 632
633 634
635 636
637 638
639 640
641 642
643 644
645 646
647 648
649 650
651 652
653 654
655 656
657 658
659 660
661 662
663 664
665 666
667 668
669 670
671 672
673 674
675 676
677 678
679 680
681 682
683 684
685 686
687 688
689 690
691 692
693 694
695 696
697 698
699 699

```

Fig. 59: Parameters used to initiate and run Job that can be useful for diagnostic purposes

RecordsField AnalysisInput JobsValidationDPLA Bulk Data MatchesTransform Details

See below for a table of all records that were altered in some way during this Transform Job. Click into a Record and then click the tab "Transform Details" to view detailed changes for that Record.

Show105 entries

Search:

DB ID	Record ID
1758764	oai:digital.library.wayne.edu:wayne:visitfrom1864b48447122
1758483	oai:digital.library.wayne.edu:wayne:Undine1990-4838933
1758499	oai:digital.library.wayne.edu:wayne:Talesfor1828b28860742
1758461	oai:digital.library.wayne.edu:wayne:Talesatt1874b21392888
1758383	oai:digital.library.wayne.edu:wayne:Sunshine1808b21378846
1758576	oai:digital.library.wayne.edu:wayne:Strawder1899b21515943
1758512	oai:digital.library.wayne.edu:wayne:Stranger1865b21378776
1758585	oai:digital.library.wayne.edu:wayne:Stories1868b21568638
1758645	oai:digital.library.wayne.edu:wayne:Stories1808b22356883
1758595	oai:digital.library.wayne.edu:wayne:springtide1938b4838933x

Showing 1 to 10 of 246 entries

Previous12345...25Next

Fig. 60: Table showing transformed Records for a Job

No additional information at this time for Analysis Jobs.

Export

Records from Jobs may be exported in a variety of ways, [read more about exporting here](#).

Record Documents

Exporting a Job as Documents takes the stored XML documents for each Record, distributes them across a user-defined number of files, exports as XML documents, and compiles them in an archive for easy downloading.

Fig. 61: Exporting Mapped Fields for a Job

For example, 1000 records where a user selects 250 per file, for Job #42, would result in the following structure:

```
- archive.zip|tar
  - j42/ # folder for Job
    - part00000.xml # each XML file contains 250 records grouped under a root XML
    ↪element <documents>
      - part00001.xml
      - part00002.xml
      - part00003.xml
```

The following screenshot shows the actual result of a Job with 1,070 Records, exporting 50 per file, with a zip file and the resulting, unzipped structure:

Why export like this? Very large XML files can be problematic to work with, particularly for XML parsers that attempt to load the entire document into memory (which is most of them). Combine is naturally pre-disposed to think in terms of the parts and partitions with the Spark back-end, which makes for convenient writing of all Records from Job in smaller chunks. The size of the “chunk” can be set by specifying the XML Records per file input in the export form. Finally, .zip or .tar files for the resulting export are both supported.

▼ j408	--	Folder
part-00000.xml	203 KB	XML Document
part-00001.xml	203 KB	XML Document
part-00002.xml	166 KB	XML Document
part-00003.xml	201 KB	XML Document
part-00004.xml	154 KB	XML Document
part-00005.xml	131 KB	XML Document
part-00006.xml	128 KB	XML Document
part-00007.xml	157 KB	XML Document
part-00008.xml	172 KB	XML Document
part-00009.xml	170 KB	XML Document
part-00010.xml	214 KB	XML Document
part-00011.xml	201 KB	XML Document
part-00012.xml	208 KB	XML Document
part-00013.xml	249 KB	XML Document
part-00014.xml	213 KB	XML Document
part-00015.xml	262 KB	XML Document
part-00016.xml	298 KB	XML Document
part-00017.xml	283 KB	XML Document
part-00018.xml	241 KB	XML Document
part-00019.xml	256 KB	XML Document
part-00020.xml	249 KB	XML Document
part-00021.xml	213 KB	XML Document
j_408_documents.zip	4.6 MB	ZIP archive

Fig. 62: Example structure of an exported Job as XML Documents

When a Job is exported as Documents, this will send users to the [Background Tasks](#) screen where the task can be monitored and viewed.

3.5.4 Viewing Record Details

At the most granular level of [Combine's data model](#) is the Record. This section will outline the various areas of the Record details page.

The table at the top of a Record details page provides identifier information:

Record: [oai:digital.library.wayne.edu:wayne:Livingto1876b22354748](#)

DB ID	1594974
Combine ID	5e2b11ea-7d39-49c1-88df-516431871357
Record ID	oai:digital.library.wayne.edu:wayne:Livingto1876b22354748
Valid?	False

Fig. 63: Top of Record details page

Similar to a Job details page, the following tabs breakdown other major sections of this Record details.

- [Record XML](#)
- [Indexed Fields](#)
- [Record Stages](#)
- [Record Validation](#)
- [DPLA Link](#)

- *Job Type Details - Records*

Record XML

This tab provides a glimpse at the raw, XML for a Record:

The screenshot shows the 'Record XML' tab selected in the top navigation bar. Below the navigation bar, the 'Record Document' section is active. It contains a preview of the raw XML document for a record. The XML content is displayed in a monospace font, showing various metadata elements such as title, author, and subject. The XML is a MARC21 record for a book titled 'Living too fast' by Oliver. The XML is encoded in UTF-8 and uses the MARC21 schema.

Fig. 64: Record's document

Note also two buttons for this tab:

- View Document in New Tab This will show the raw XML in a new browser tab
- Search for Matching Documents: This will search all Records in Combine for other Records with an *identical* XML document

Indexed Fields

This tab provides a table of all indexed fields in ElasticSearch for this Record:

Notice in this table the columns DPLA Mapped Field and Map DPLA Field. Both of these columns pertain to a functionality in Combine that attempts to “link” a Record with the same record in the live DPLA site. It performs this action by querying the DPLA API (DPLA API credentials must be set in `localsettings.py`) based on mapped indexed fields. Though this area has potential for expansion, currently the most reliable and effective DPLA field to try and map is the `isShownAt` field.

The `isShownAt` field is the URL that all DPLA items require to send visitors back to the originating organization or institution's page for that item. As such, it is also unique to each Record, and provides a handy way to “link” Records in Combine to items in DPLA. The difficult part is often figuring out which indexed field in Combine contains the URL.

Note: When this is applied to a single Record, that mapping is then applied to the Job as a whole. Viewing another Record from this Job will reflect the same mappings. These mappings can also be applied at the Job or Record level.

In the example above, the indexed field `mods_location_url_usage_primary` has been mapped to the DPLA field `isShownAt` which provides a reliable linkage at the Record level.

Combine Documentation

[Record XML](#) [Indexed Fields](#) [Record Stages](#) [Validation](#) [DPLA Link](#) [Job Type Specific](#)

Indexed Fields

[View Elasticsearch document](#)

Show 100 entries

Search:

Field Name	DPLA Mapped Field	Map DPLA Field	Field Value
combine_id		Select DPLA fi	5e2b11ea-7d39-49c1-80df-516431871357
db_id		Select DPLA fi	1594974
mods_accessCondition_@type_useAndReproduction		Select DPLA fi	This book is in the public domain.
mods_extension		Select DPLA fi	[b22354748, b51075969, wayneLivingto1876b22354748]
mods_identifier_@type_local		Select DPLA fi	Livingto1876b22354748
mods_language_languageTerm_@authority_iso639-2b_@type_code		Select DPLA fi	eng
mods_location_url_@access_preview		Select DPLA fi	http://digital.library.wayne.edu/item/wayneLivingto1876b22354748/thumbnail
mods_location_url_@usage_primary	isShownAt	Select DPLA fi	http://digital.library.wayne.edu/item/wayneLivingto1876b22354748
mods_name_namePart		Select DPLA fi	Optic, Oliver
mods_name_namePart_@type_date		Select DPLA fi	1822-1897
mods_name_role_roleTerm_@authority_marccrelator_@type_text		Select DPLA fi	creator
mods_note_@type_digitalization		Select DPLA fi	The electronic version of this item was provided by the Wayne State University Library System and is freely accessible through the Wayne State University Libraries Digital Collections.
mods_originInfo_dateIssued_@encoding_w3cdtf_@keyDate_yes		Select DPLA fi	1876
mods_originInfo_issuance		Select DPLA fi	monographic
mods_originInfo_place_placeTerm_@authority_marccountry_@type_code		Select DPLA fi	mau
mods_originInfo_place_placeTerm_@type_text		Select DPLA fi	Boston
mods_originInfo_publisher		Select DPLA fi	Lee and Shepard

Fig. 65: Indexed fields for a Record

Record Stages

This table show the various “stages” of a Record, which is effectively what Jobs the Record also exists in:

[Record XML](#) [Indexed Fields](#) [Record Stages](#) [Validation](#) [DPLA Link](#) [Job Type Specific](#)

Stages

The following table shows this Record through various “stages”, or Jobs, in Combine. Each stage shares the following Combine ID: 5e2b11ea-7d39-49c1-80df-516431871357, which links them across Jobs.

DB ID	Record ID	Job ID	Job Name	Job Type	Document XML	Error	Is Valid	ElasticSearch document
1593775	oai:digital.library.wayne.edu:wayneLivingto1876b22354748	254	Ramsey Books collection	HarvestOAIJob	View	None	False	View
1594974	oai:digital.library.wayne.edu:wayneLivingto1876b22354748	257	MergeJob @ May, 10, 2018, 1:19:51 PM	MergeJob	View	None	False	View

Fig. 66: Record stages across other Jobs

Records are connected by their Combine ID (`combine_id`). From this table, it is possible to jump to other, earlier “upstream” or later “downstream”, versions of the same Record.

Record Validation

This tab shows all Validation Tests that were run for this Job, and how this Record fared:

More information about [Validation Results](#) can be found here.

Record XML Indexed Fields Record Stages Validation **DPLA Link** Job Type Specific

Validation Result: Failed

The following Validation Scenarios were run for this Job.

Validation Scenario	Test Failed	Actions
DPLA minimum	There must be a rights statement	Run Validation

[Test Validation Scenario on this Record](#)

Fig. 67: Record's Validation Results tab


DPLA Link

When a mapping has been made to the DPLA `isShownAt` field from the Indexed Fields tab (or at the Job level), and if a DPLA API query is successful, a result will be shown here:

Record XML Indexed Fields Record Stages Validation **DPLA Link** Job Type Specific

DPLA API Item match

A [matching DPLA Item](#) was found! Below is information from the DPLA API response.



@id	http://dp.la/api/items/99a090b3be9adb9649ae26549f073e#sourceResource
rights	["This book is in the public domain."]
creator	["Optic, Oliver 1822-1897"]
title	Living too fast
language	[{"name": "English", "iso639_3": "eng"}]
stateLocatedIn	[{"name": "Michigan"}]
collection	{"id": "", "title": "Eloise Ramsey Collection of Literature for Young People", "description": "", "@id": ""}
subject	[]
date	{"begin": "1876", "end": "1876", "displayDate": "1876"}
type	["text"]
extent	351 p., [4] leaves of plates : ill. ; 20 cm.

Fig. 68: Indexed fields for a Record

Results from the DPLA API are parsed and presented here, with the full API JSON response at the bottom (not pictured here). This can be useful for:

- confirming existence of a Record in DPLA
- easily retrieving detailed DPLA API metadata about the item
- confirming that changes and transformations are propagating as expected

Job Type Details - Records

For each Job type – Harvest, Transform, Merge/Duplicate, and Analysis – the Record details screen provides a tab with information specific to that Job type.

Harvest Jobs

No additional information at this time for Harvest Jobs.

Transform Jobs

This tab will show Transformation details specific to this Record.

The first section shows the Transformation Scenario used, including the transformation itself, and the “input” or “upstream” Record that was used for the transformation:

The following are details specific to the Transformation that created this record.

Transformation Scenario Used

ID	Name	Type	Transformation Payload	Input Document
23	Juvenile -> youth	Python Code Snippet	View	View (Record #1757450)

[Re-run Transformation on Input Record](#)

Fig. 69: Information about Input Record and Transformation Scenario used for this Record

Clicking the “Re-run Transformation on Input Record” button will send you to the Transformation Scenario preview page, with the Transformation Scenario and Input Record automatically selected.

Further down, is a detailed diff between the **input** and **output** document for this Record. In this minimal example, you can observe that `Juvenile` was changed to `Youth` in the Transformation, resulting in only a couple of isolated changes:

For transformations where the Record is largely re-written, the changes will be lengthier and more complex:

Users may also click the button “View Side-by-Side Changes” for a GitHub-esque, side-by-side diff of the Input Record and the Current Record (made possible by the `sxsdiff` library):

Merge/Duplicate Jobs

No additional information at this time for Merge/Duplicate Jobs.

Analysis Jobs

No additional information at this time for Analysis Jobs.

3.5.5 Managing Jobs

Once you work through initiating the Job, configuring the optional parameters outlined below, and running it, you will be returned to the Record Group screen and presented with the following job lineage “graph” and a table showing all Jobs for the Record Group:

The graph at the top shows all Jobs for this Record Group, and their relationships to one another. The edges between nodes show how many Records were used as input for the target Job, what – if any – filters were applied. This graph is zoomable and clickable. This graph is designed to provide some insight and context at a glance, but the table below is designed to be more functional.

The table shows all Jobs, with optional filters and a search box in the upper-right. The columns include:

- `Job ID` - Numerical Job ID in Combine
- `Timestamp` - When the Job was started
- `Name` - Clickable name for Job that leads to Job details, optionally given one by user, or a default is generated. This is editable anytime.
- `Organization` - Clickable link to the Organization this Job falls under
- `Record Group` - Clickable link to the Record Group this Job falls under (as this table is reused throughout Combine, it can sometimes contain Jobs from other Record Groups)
- `Job Type` - Harvest, Transform, Merge, or Analysis
- `Livy Status` - This is the status of the Job in Livy

Record Changes

```

-----

+++

@@ -154,7 +154,7 @@
    <mods:topic>Santa Claus</mods:topic>

-    <mods:genre> Juvenile poetry</mods:genre>
+    <mods:genre> Youth poetry</mods:genre>

    </mods:subject>
@@ -166,7 +166,7 @@
    <mods:topic>Christmas</mods:topic>

-    <mods:genre> Juvenile poetry</mods:genre>
+    <mods:genre> Youth poetry</mods:genre>

    </mods:subject>

```

Fig. 70: Record transformation diff, small change

Chapter 3. Table of Contents

Fig. 71: Snippet of Record transformation diff, many changes

Fig. 72: Side-by-side diff, minimal changes

```

1 <mods:mods xmlns:mods="http://www.loc.gov/mods/v3">
2
3   <mods:subject>
4
5   <mods:hierarchicalGeographic>
6
7   <mods:country>United States</mods:country>
8
9   <mods:state>California</mods:state>
10
11 </mods:hierarchicalGeographic>
12
13 </mods:subject>
14
15 <mods:titleInfo type="alternative">
16
17 <mods:title>Trailed to the end</mods:title>
18
19 </mods:titleInfo>
20
21 <mods:titleInfo>
22
23
24
25
26
27
28
29
30
31

```

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <mods:mods xmlns:mods="http://www.loc.gov/mods/v3">
3   <mods:titleInfo>
4     <mods:title>Buffalo Bill in Death Valley, or, Trailed to the end</mods:title>
5   </mods:titleInfo>
6   <mods:subject>
7     <mods:hierarchicalGeographic>
8       <mods:country>United States</mods:country>
9       <mods:state>California</mods:state>
10     </mods:hierarchicalGeographic>
11   </mods:subject>
12   <mods:topic>Dime novels</mods:topic>
13   </mods:subject>
14   <mods:topic>Western stories</mods:topic>
15   </mods:subject>
16   <mods:subject>
17     <mods:name type="personal" authority="naf">
18       <mods:namePart>Buffalo Bill</mods:namePart>
19       <mods:namePart type="date">1846-1917</mods:namePart>
20     </mods:subject>
21   </mods:subject>
22
23
24
25
26
27
28
29
30
31

```

Fig. 73: Side-by-side diff, many changes

Record Group: Record Group Example

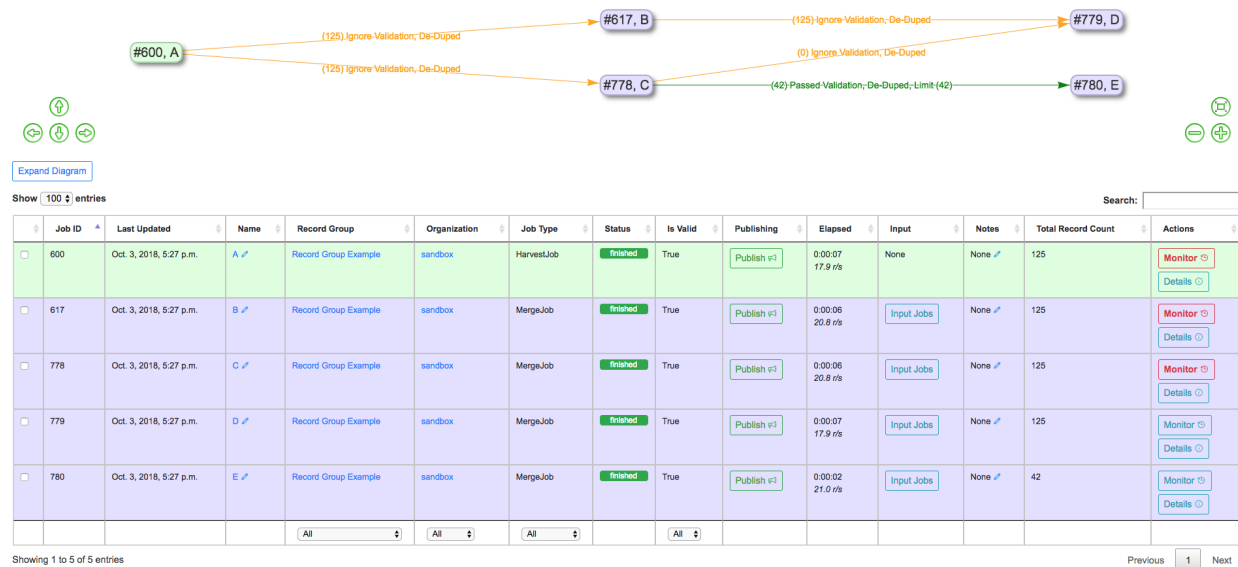


Fig. 74: Job “lineage” graph at the top, table with Jobs at the bottom

- gone - Livy has been restarted or stopped, and no information about this Job is available
- available - Livy reports the Job as complete and available
- waiting - The Job is queued behind others in Livy
- running - The Job is currently running in Livy
- Finished - Though Livy does the majority of the Job processing, this indicates the Job is finished in the context of Combine
- Is Valid - True/False, True if no validations were run or *all* Records passed validation, False if any Records failed any validations
- Publishing - Buttons for [Publishing](#) or [Unpublishing](#) a Job
- Elapsed - How long the Job has been running, or took
- Input - All input Jobs used for this Job
- Notes - Optional notes field that can be filled out by User here, or in Job Details
- Total Record Count - Total number of successfully processed Records
- Actions - Buttons for Job details, or monitoring status of Job in Spark (see [Spark and Livy documentation](#) for more information)

This graph and table represents Jobs already run, or running. This is also where Jobs can be moved, stopped, deleted, rerun, even cloned. This is performed by using the bank of buttons under “Job Management”:

Job Management		
Move	Moves selected Jobs to another Record Group within this Organization	<div>Select Record Group...</div> <div>Move Selected Jobs →</div> <div><input type="checkbox"/> Include Downstream?</div>
Stop Jobs	Stop Job(s)	<div>Stop Selected Jobs ⌵</div> <div><input type="checkbox"/> Include Downstream?</div>
Delete Jobs	Delete Job(s)	<div>Delete Selected Jobs ✕</div> <div><input type="checkbox"/> Include Downstream?</div>
Re-Run	Re-run Job	<div>Re-Run Selected Jobs ↺</div> <div><input checked="" type="checkbox"/> Include Downstream?</div>
Clone	Clone Job	<div>Clone Selected Jobs 📄</div> <div><input checked="" type="checkbox"/> Include Downstream? <input checked="" type="checkbox"/> Re-Run on Clone?</div>

Fig. 75: Buttons used to manage running and finished Jobs

All management options contain a slider titled “Include Downstream” that defaults to **on** or **off**, depending on the task. When **on** for a particular task, this will analyze the lineage of all selected Jobs and determine which are downstream and include them in the action being performed (e.g. moving, deleting, rerunning, etc.)

The idea of “downstream” Jobs, and some of the actions like **Re-Running** and **Cloning** introduce another dimension to Jobs and Records in Combine, that of **Pipelines**.

Pipelines

What is meant by “downstream” Jobs? Take the interconnected Jobs below:

In this example, the OAI Harvest Job A is the “root” or “origin” Job of this lineage. This is where Records were first harvested and created in Combine (this might also be static harvests, or other forms of importing Records yet to come). All other Jobs in the lineage – B, C, D, and E – are considered “downstream”. From the point of view of A, there is a

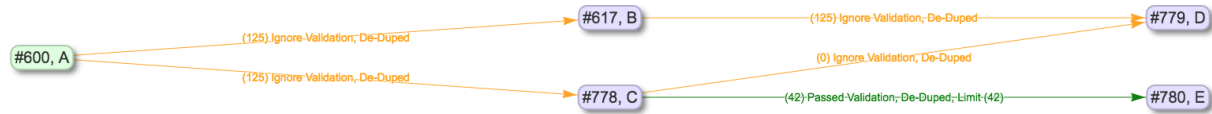


Fig. 76: Five interconnected Jobs

single pipeline. If a user were to reharvest A, potentially adding, removing, or modifying Records in that Job, this has implications for all other Jobs that either got Records from A, or got Records from Jobs that got Records from A, and so forth. In that sense, Jobs are “downstream” if changes to an “upstream” Job would potentially change their own Records.

Moving to B, only one Job is downstream, D. Looking at C, there are two downstreams Jobs, D and E. Looking again at the Record Group lineage, we can see then that D has two upstream Jobs, B and C. This can be confirmed by looking at the “Input Jobs” tab for D:

Input Jobs

The following Jobs were used as input for this MergeJob.

Input Job ID	Input Job Name	Job Type	Job Specific Filters Applied?	Validity	De-Dupe Records	Mapped Field Query Filtered	Numerical Limit	Total Passed Records
617	B	MergeJob	False	Ignored Validation	True	None	None	125
778	C	MergeJob	False	Ignored Validation	True	None	None	0
Total:								125

Fig. 77: Input Jobs for Job D, showing Jobs B and C

Why are there zero Records coming from C as an Input Job? Looking more closely at this contrived example, and the input filters applied to Jobs B and C, we see that “De-Dupe Records” is true for both. We can infer that Jobs B and C provided Records with the same `record_id`, and as a result, were all de-duped – skipped – from C during the Merge.

Another view of the lineage for D, from it’s perspective, can be seen at the top of the Job details page for D, confirming all this:

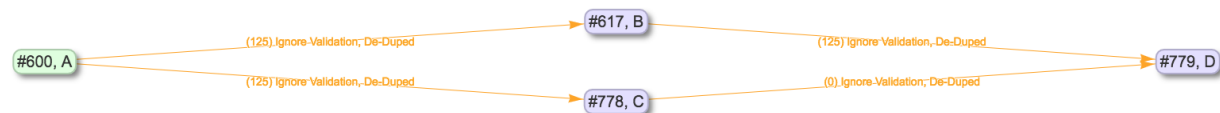


Fig. 78: Upstream lineage for Job D

Getting back to the idea of pipelines and Job management, what would happen if we select A and click the “Re-Run Selected Jobs” button, with “Include Downstream” turned on? Jobs A-E would be slated for re-running, queuing in order to ensure that each Jobs is getting updated Records from each upstream Job:

We can see that status changed for each Job (potentially after a page refresh), and the Jobs will re-run in order.

We also have the ability to **clone** Jobs, including or ignoring downstream Jobs. The following is an example of cloning C, *not* including downstream Jobs:

Under the hood, all validations, input filters, and parameters that were set for C are copied to the new Job C (CLONED), but because downstream Jobs were not included, D and E were not cloned. But if we were to select downstream Jobs from C when cloning, we’d see something that looks like this:

Woah there! Why the line from B to the newly created cloned Job D (CLONE)? D was downstream from C during the clone, so was cloned as well, but still required input from B, which was not cloned. We can imagine that B might

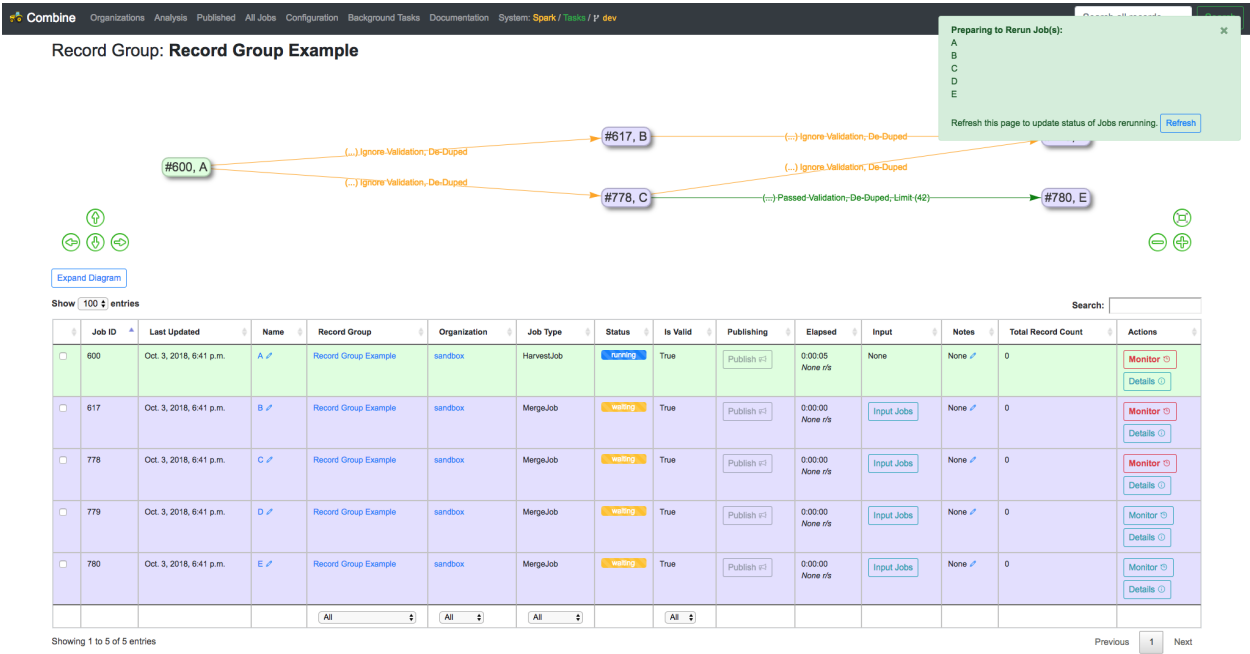


Fig. 79: Re-Running Job A, including downstream Jobs

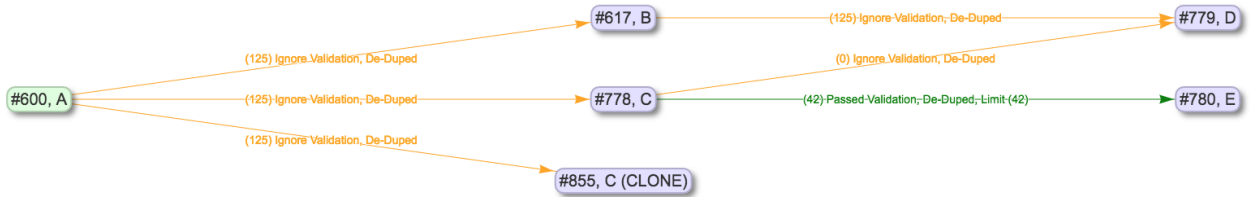


Fig. 80: Cloning Job C

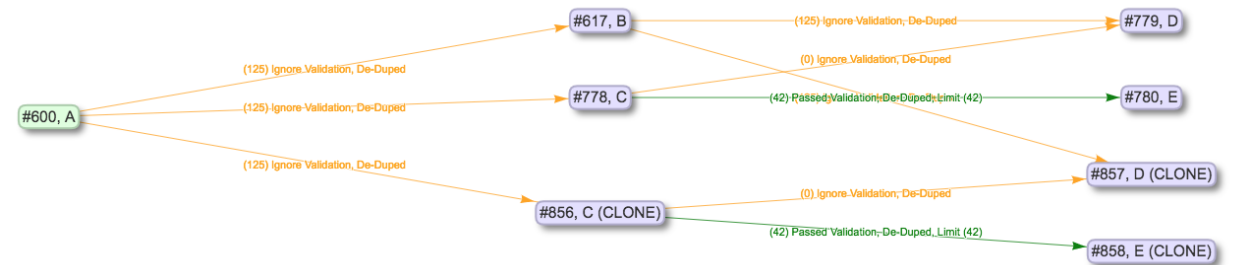


Fig. 81: Cloning Job C, including downstream Jobs

be a group of Records that rarely change, but are required in our pursuits, and so that connection is persisted.

As one final example of cloning, to get a sense about Input Jobs for Jobs that are cloned, versus those that are not, we can look at the example of cloning A, including all its downstream Jobs:

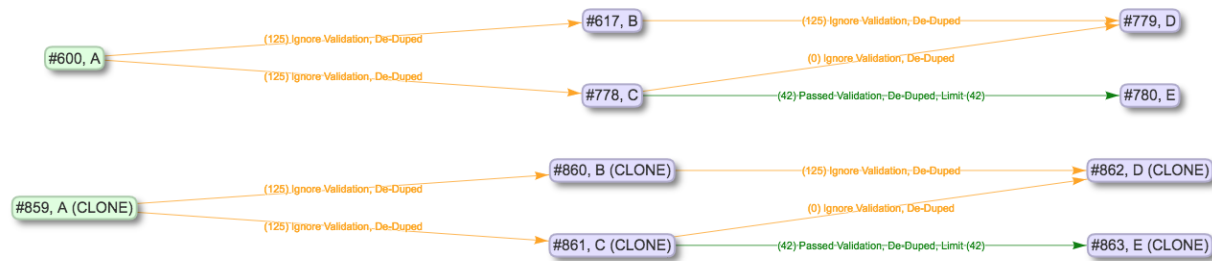


Fig. 82: Cloning Job A, including downstream Jobs

Because A has every job in this view as downstream, cloning A essentially clones the entire “pipeline” and creates a standalone copy. This could be useful for cloning a pipeline to test re-running the entire thing, where it is not desirable to risk the integrity of the pipeline before knowing if it will be successful.

Finally, we can see that the “Include Downstream” applied to other tasks as well, e.g. deleting, where we have selected to delete A (CLONE) and all downstream Jobs:

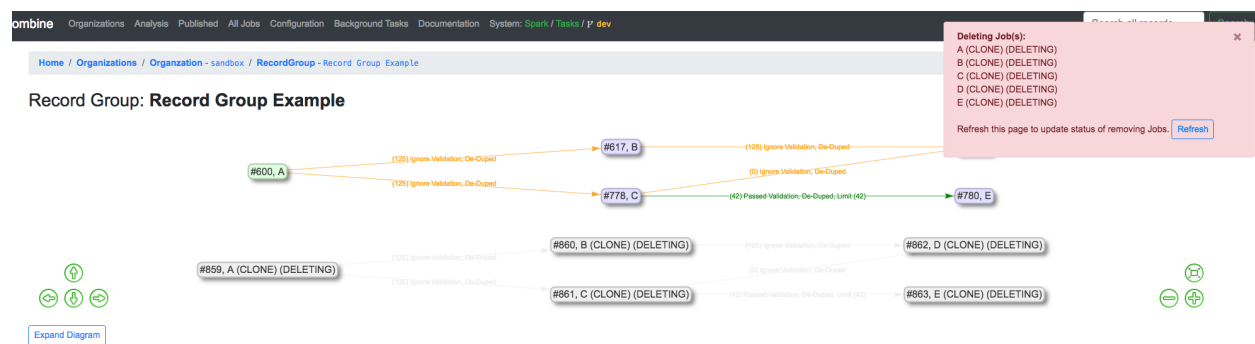


Fig. 83: Deleting Job A (CLONE), and all downstream Jobs

“Pipelines” are not a formal structure in Combine, but can be a particularly helpful way to think about a “family” or “lineage” of connected Jobs. The ability to re-run and clone Jobs came later in the data model, but with the addition of granular control of input filters for Input Jobs, can prove to be extremely helpful for setting up complicated pipelines of interconnected Jobs that can be reused.

3.6 Harvesting Records

Harvesting is how Records are first introduced to Combine. Like all Jobs, Harvest Jobs are run from the the Record Group overview page.

The following will outline specifics for running Harvest Jobs, with more [general information about running Jobs here](#).

3.6.1 OAI-PMH Harvesting

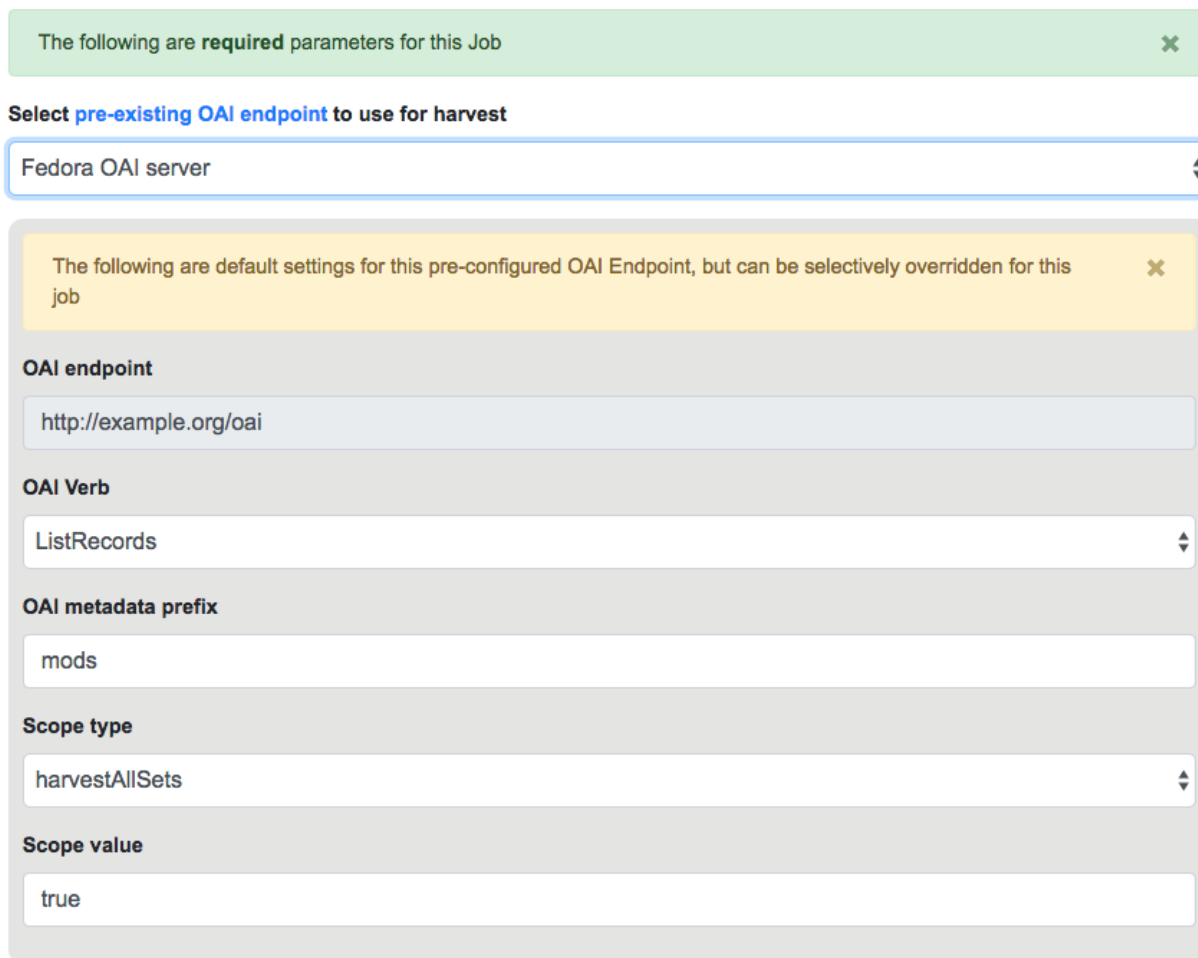
OAI-PMH harvesting in Combine utilizes the Apache Spark OAI harvester from [DPLA’s Ingestion 3 engine](#).

Before running an OAI harvest, you must first [configure an OAI Endpoint](#) in Combine that will be used for harvesting from. This only needs to be done once, and can then be reused for future harvests.

From the Record Group page, click the “Harvest OAI-PMH” button at the bottom.

Like all Jobs, you may optionally give the Job a name or add notes.

Below that, indicated by a green alert, are the required parameters for an OAI Job. First, is to select your pre-configured OAI endpoint. In the screenshot below, an example OAI endpoint has been selected:



The following are **required** parameters for this Job

Select **pre-existing OAI endpoint** to use for harvest

Fedora OAI server

The following are default settings for this pre-configured OAI Endpoint, but can be selectively overridden for this job

OAI endpoint

http://example.org/oai

OAI Verb

ListRecords

OAI metadata prefix

mods

Scope type

harvestAllSets

Scope value

true

Fig. 84: Selecting OAI endpoint and configuring parameters

Default values for harvesting are automatically populated from your configured endpoint, but can be overridden at this time, for this harvest only. Changes are not saved for future harvests.

Once configurations are set, click “Run Job” at the bottom to harvest.

Identifiers for OAI-PMH harvesting

As an Harvest type Job, OAI harvests are responsible for creating a Record Identifier (`record_id`) for each Record. The `record_id` is pulled from the `record/header/identifier` field for each Record harvested.

As you continue on your metadata harvesting, transforming, and publishing journey, and you are thinking about how identifiers came to be, or might be changed, this is a good place to start from to see what the originating identifier was.

3.6.2 Static File Harvest

It is also possible to harvest Records from static sources, e.g. XML uploads. Combine uses [Databricks Spark-XML](#) to parse XML records from uploaded content. This utilizes the powerful [globbing capabilities](#) of Hadoop for locating XML files. Users may also provide a location on disk as opposed to uploading a file, but this is probably less commonly used, and the documentation will focus on uploads.

The following are **required** parameters for this Job ✕

Provide Static Content

For static harvests, optionally upload content, or provide a location on the filesystem where the content is.

Upload

Filesystem

Choose file...

Browse

Fig. 85: Upload file, or provide location on disk for Static harvest

Using the Spark-XML library provides an efficient and powerful way of locating and parsing XML records, but it does so in a way that might be unfamiliar at first. Instead of providing XPath expressions for locating Records, only the **XML Record's root element is required**, and the Records are located as raw strings.

For example, a MODS record that looks like the following:

```
<mods:mods>
  <mods:titleInfo>
    <mods:title>Amazing Record of Incalculable Worth</mods:title>
  </mods:titleInfo>
  ...
  ...
</mods:mods>
```

Would need only the following Root XML element string to be found: `mods:mods`. No angle brackets, no XPath expressions, just the element name!

However, a close inspect reveals this MODS example record does not have the required namespace declaration, `xmlns:mods="http://www.loc.gov/mods/v3"`. It's possible this was declared in a different part of the XML Record. Because Spark-XML locates XML records more as strings, as opposed to parsed documents, Combine also allows users to include an **XML root element declaration** that will be used for each Record found. For this example, the following could be provided:

```
xmlns:mods="http://www.loc.gov/mods/v3"
```

Which would result in the following, final, valid XML Record in Combine:

```
<mods:mods xmlns:mods="http://www.loc.gov/mods/v3">
  <mods:titleInfo>
    <mods:title>Amazing Record of Incalculable Worth</mods:title>
  </mods:titleInfo>
  ...
  ...
</mods:mods>
```

Find and Parse XML Records

To locate and parse XML records from the provided file(s), additional information is needed: the root XML element for each record, and if needed, additional XML namespace declarations.

Root XML element	Because static XML harvests may be looping through many files, directories, archives, or single XML files, a root XML element is needed to locate XML documents that should be parsed and used. Unlike XPath, only the element name is needed. For example, a <code>mods:mods</code> would be sufficient for MODS records with a namespace, or <code>oai_dc</code> for OAI Dublin Core records.
Re-write XML root element declarations	Because Combine does not use XPath to parse the records, it is possible that records will be retrieved without required namespaces (often only declared at the root element). Here, you may optionally rewrite the declaration for each Record. For example, if providing the XML root <code>mods:mods</code> results in root XML nodes that look like <code><mods:mods></code> , the declaration <code>xmlns:mods="http://www.loc.gov/mods/v3"</code> could be passed and will be added to each record resulting in <code><mods:mods xmlns:mods="http://www.loc.gov/mods/v3"></code> , which is valid XML.

Root XML Element:

e.g. `mods:mods`, `oai_dc`, etc.

Re-write Root XML Declaration (if needed):

e.g. `xmlns:mods="http://www.loc.gov/mods/v3"`

Fig. 86: Showing form to provide root XML element for locating Records, and optional XML declarations

Once a file has been selected for uploading, and these required parameters are set, click “Run Job” at the bottom to harvest.

Is this altering the XML records that I am providing Combine?

The short answer is, **yes**. But, it’s important to remember that XML files are often altered in some way when parsed and re-serialized. Their integrity is not character-by-character similarity, but what data can be parsed. This approach only alters the declarations in the root XML element.

Uploads to Combine that already include namespaces, and all required declarations, at the level of each individual Record, do not require this re-writing and will leave the XML untouched.

What kind of files and/or structures can be uploaded?

Quite a few! Static harvests will scour what is uploaded – through a single XML file, across multiple files within a zipped or tarred archive file, even recursively through directories if they are present in an archive file – for the **root XML element**, e.g. `mods:mods`, parsing each it encounters.

Examples include:

- METS file with metadata in `<dmdSec>` sections
- zip file of directories, each containing multiple XML files
- single MODS XML *file*, that contains multiple MODS *records*
- though not encouraged, even a `.txt` file with XML strings contained therein!

Identifiers for Static harvesting

For static harvests, identifiers can be created in one of two ways:

- by providing an XPath expression to retrieve a string from the parsed XML record
- a random, UUID is assigned based on a hash of the XML record as a string

Locate Identifier in Document

Finally, an optional XPath expression can be provided to locate a unique, meaningful identifier for each document.

Note: While not required, this is encouraged. In the absence of an XPath expression to locate an identifier, an MD5 hash of the document's contents will be created, which might help with potential duplicate records. However, if the document changes harvest-to-harvest, this identifier will not remain constant. Furthermore, this MD5 hash identifier will propagate through Combine and eventually be used for publishing, which is important to consider.

Description	Example
Use Dublin Core <code><dc:identifier></code> element.	<code>//dc:identifier</code>
Locate <code><mods:url></code> element with <code>access</code> attribute.	<code>//mods:mods/mods:location/mods:url[@access]</code>

XPath for Record Identifier:

e.g. `//dc:identifier`

Fig. 87: Form for providing optional XPath for retrieving identifier

3.7 Transforming Records

Transformation Jobs are how Records are transformed in some way in Combine; all other Jobs merely copy and/or analyze Records, but Transformation Jobs actually alter the Record's XML that is stored in MySQL

The following will outline specifics for running Transformation Jobs, with more [general information about running Jobs here](#).

Similar to Harvest Jobs, you must first [configure a Transformation Scenario](#) that will be selected and used when running a Transformation Job.

The first step is to select a **single** input Job to supply the Records for transformation:

Search:

Select	Job ID	Name	Organization	Record Group	Job Type	Status	Finished	Is Valid	Timestamp	Input	Notes	Record Count
<input checked="" type="checkbox"/>	254	HarvestOAIJob @ May. 10, 2018, 12:04:10 PM	Amazing University	Fedora Repository	HarvestJob	available	True	False	May 10, 2018, 12:04 p.m.		None	438
			<div>All</div>	<div>All</div>	<div>All</div>	<div>All</div>	<div>All</div>	<div>All</div>				

Showing 1 to 1 of 1 entries

Fig. 88: Selecting an input Job for transformation

Next, will be selecting your pre-configured Transformation Scenario:

As most of the configuration is done in the Transformation Scenario, there is very little to do here! Select optional parameters and click "Run Job" at the bottom.

The following are **required** parameters for this Job ✕

Select Transformation Scenario to use

MODS to Service Hub profile (xslt)

Fig. 89: Selecting Transformation Scenario

3.8 Merging Records

The following will outline specifics for running Merge / Duplicate Jobs, with more [general information about running Jobs here](#).

“Merge / Duplicate” Jobs are precisely what they sound like: they are used to copy, merge, and/or duplicate Records in Combine. They might also be referred to as simply “Merge” Jobs throughout.

To run a Merge Job, select “Duplicate / Merge Job” from the Record Group screen.

From the Merge Job page, you may notice there are no required parameters! However, unlike Transformation Jobs where input Jobs could only be selected from the same Record Group, the input Job selection screen will show Jobs from across *all* Organizations and Record Groups:

Showing all Jobs from all Record Groups. To view Jobs only from this Record Group, **Fedora Repository**, use the dropdown filters in the table below ✕

#251, HarvestStaticXMLJob @ May. 08, 2018, 1:03:29 PM

Records (250) ➔

#252, TransformJob @ May. 08, 2018, 1:05:40 PM

All Records (250) ➔

#253, PublishJob @ May. 08, 2018, 3:56:27 PM

#254, Ramsey Books collection

#256, Rencen Collection

Select	Job ID	Name	Organization	Record Group	Job Type	Status	Finished	Is Valid	Timestamp	Input	Notes	Record Count
<input type="checkbox"/>	251	HarvestStaticXMLJob @ May. 08, 2018, 1:03:29 PM	SuperOrg	TurboRG	HarvestJob	gone	True	False	May 8, 2018, 1:03 p.m.		None	250
<input type="checkbox"/>	252	TransformJob @ May 08, 2018, 1:05:40 PM	SuperOrg	TurboRG	TransformJob	gone	True	True	May 8, 2018, 1:05 p.m.	HarvestStaticXMLJob @ May 08, 2018, 1:03:29 PM, Job #251, from Record Group: TurboRG	None	250
<input type="checkbox"/>	253	PublishJob @ May. 08, 2018, 3:56:27 PM	SuperOrg	TurboRG	PublishJob	gone	True	True	May 8, 2018, 3:56 p.m.	TransformJob @ May 08, 2018, 1:05:40 PM, Job #252, from Record Group: TurboRG	None	250
<input type="checkbox"/>	254	Ramsey Books collection	Amazing University	Fedora Repository	HarvestJob	available	True	False	May 10, 2018, 12:04 p.m.		None	438
<input type="checkbox"/>	256	Rencen Collection	Amazing University	Fedora Repository	HarvestJob	available	True	False	May 10, 2018, 1:08 p.m.		None	323
			All	All	All	All	All	All				

Showing 1 to 5 of 5 entries

Fig. 90: Showing possible input Jobs from all Organizations and Record Groups

There not many Jobs in this instance, but this could get intimidating if lots of Jobs were present in Combine. Both the Job “lineage graph” – visual graph near the top – or the table of Jobs can be useful for limiting.

Clicking on a single Job in the lineage graph will filter the table to include only **that** Job, and all Jobs there **input** for that Job, and graying out Jobs in the lineage graph to represent the same. Clicking outside of a Job will clear the filter.

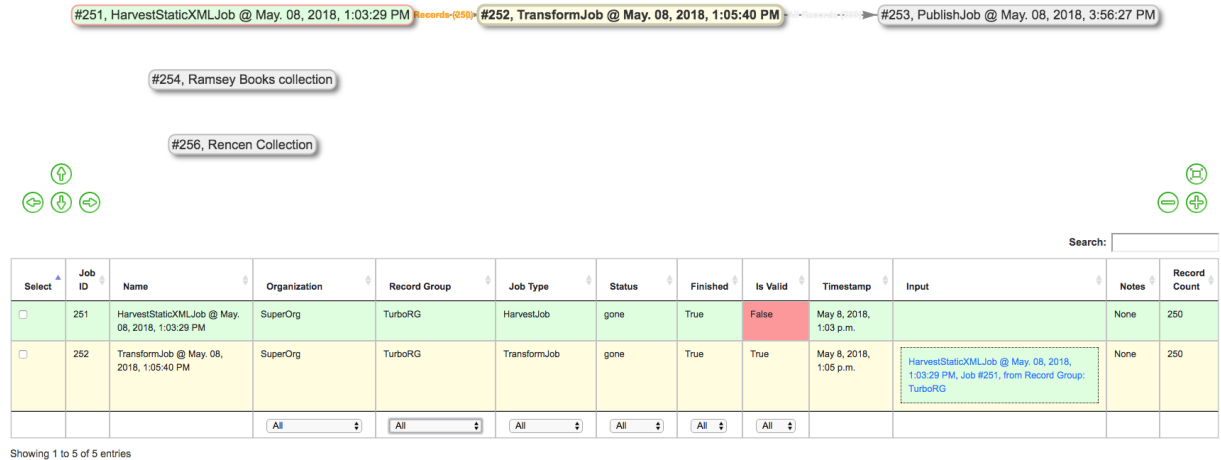


Fig. 91: Clicking a Job will highlight that Job, and upstream “input” Jobs

Additionally, filters from the Jobs table can be used to limit by Organization, Record Group, Job Type, Status, or even keyword searching. When filters are applied, Jobs will be grayed out in the lineage graph.

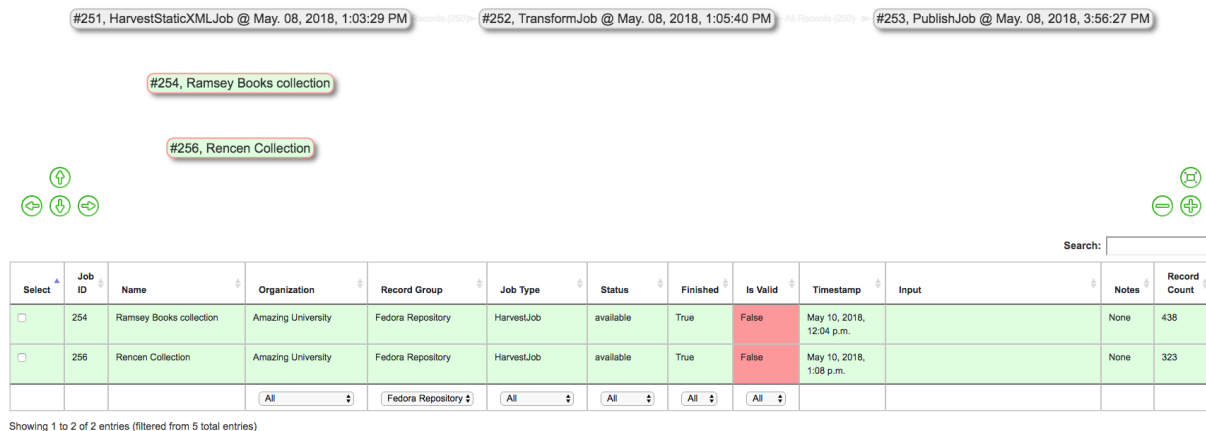


Fig. 92: Showing filtering table by Record Group “Fedora Repository”

Also of note, you can select **multiple** Jobs for Merge / Duplicate Jobs. When Jobs are merged, a duplicate check is run for the **Record Identifiers only**.

Select desired Jobs to merge or duplicate – which can be a single Job – and click “Run Job”.

The following screenshot shows the results of a Merge Job with *two* input Jobs from the Record Group screen:

3.8.1 Why Merge or Duplicate Records?

With the flexibility of the data model,

```
Organization --> Record Group --> Job --> Record
```

comes some complexity in execution.

Merge Jobs have a variety of possible use cases:

Diagram illustrating the merging of two jobs into one. Job #254 (Ramsay Books collection) and Job #256 (Rencen Collection) are merged into Job #257 (MergeJob @ May. 10, 2018, 1:19:51 PM). Arrows indicate 'All Records (438)' from Job #254 and 'All Records (323)' from Job #256 merging into Job #257.

Job ID	Timestamp	Name	Organization	Record Group	Job Type	Status	Finished	Is Valid	Elapsed	Input	Notes	Total Record Count	Monitor	Actions
254	May 10, 2018, 12:04 p.m.	Ramsay Books collection	Amazing University	Fedora Repository	HarvestJob	available	True	False	0:00:56 7.8 r/s	None	None ✓	438	Livy / Spark	Details
256	May 10, 2018, 1:08 p.m.	Rencen Collection	Amazing University	Fedora Repository	HarvestJob	available	True	False	0:00:33 9.8 r/s	None	None ✓	323	Livy / Spark	Details
257	May 10, 2018, 1:19 p.m.	MergeJob @ May. 10, 2018, 1:19:51 PM	Amazing University	Fedora Repository	MergeJob	running	False	True	0:00:00 None r/s	<div> Ramsay Books collection, Job #254, from Record Group: Fedora Repository </div> <div> Rencen Collection, Job #256, from Record Group: Fedora Repository </div>	None ✓	0	Livy / Spark	Details

Showing 1 to 3 of 3 entries

Previous 1 Next

Fig. 93: Merging two Jobs into one

- duplicate a Job solely for [analysis](#) purposes
- with a single Record Group, optionally perform multiple, small harvests, but eventually merge them in preparation for [publishing](#)
- Merge Jobs are actually what run behind the scenes for [Analysis Jobs](#)
- Merge Jobs are the only Job type that can pull Jobs from *across* Organizations or Record Groups
- shunt a subset of valid or invalid records from Job for more precise transformations or analysis

As mentioned above, one possible use of Merge / Duplicating Jobs would be to utilize the “Record Input Validity Valve” option to shunt valid or invalid Records into a new Job. In this possible scenario, you could:

- from Job A, select only *invalid* Records to create Job B
- assuming Job B fixed those validation problems, merge *valid* Records from Job A with now *valid* Records from Job B to create Job C

This can be helpful if Job A is quite large, but only has a few Records that need further transformation, *or*, the Transformation that will fix invalid Records, would break – invalidate – other perfectly good Records from Job A. Here is a visual sense of this possible workflow, notice the record counts for each edge:



Fig. 94: Example of shunting Records based on validity, and eventually merging all valid Records

3.9 Publishing Records

The following will outline specifics for Publishing a Record Group, with more [general information about running Jobs here](#).

3.9.1 How does Publishing work in Combine?

As a tool for aggregating metadata, Combine must also have the ability to serve or distribute aggregated Records again. This is done by “publishing” in Combine, which happens at the [Job level](#).

When a Job is published, a user may a Publish Set Identifier (`publish_set_id`) that is used to aggregate and group published Records. For example, in the built-in OAI-PMH server, that Publish Set Identifier becomes the OAI set ID, or for exported flat XML files, the `publish_set_id` is used to create a folder hierarchy. Multiple Jobs can publish under the same Publish Set ID, allowing for grouping of materials when publishing.

On the back-end, publishing a Job adds a flag to Job that indicates it is published, with an optional `publish_set_id`. Unpublishing removes these flags, but maintains the Job and its Records.

Currently, the the following methods are available for publishing Records from Combine:

- *OAI-PMH Server*
- *Export of Flat Files*

3.9.2 Publishing a Job

Publishing a Job can be initiated one of two ways: from the Record Group’s list of Jobs which contains a column called “Publishing”:

Or the “Publish” tab from a Job’s details page. Both point a user to the same screen, which shows the current publish status for a Job.

If a Job is unpublished, a user is presented with a field to assign a Publish Set ID and publish a Job:

If a Job is already published, a user is presented with information about the publish status, and the ability to *unpublish*:

Both publishing and unpublishing will run a background task.

Note: When selecting a Publish Set ID, consider that when the Records are later harvested *from* Combine, this Publish Set ID – at that point, an OAI set ID – will prefix the Record Identifier to create the OAI identifier. This behavior is consistent with other OAI-PMH aggregators / servers like REPOX. It is good to consider what OAI sets these Records have been published under in the past (thereby effecting their identifiers), and/or special characters should probably be avoided.

Identifiers during metadata aggregation is a complex issue, and will not be addressed here, but it’s important to note that the Publish Set ID set during Publishing Records in Combine will have bearing on those considerations.

3.9.3 Viewing Publishing Records

All published Records can be viewed from the “Published” section in Combine, which can be navigated to from a consistent link at the top of the page.

The “Published Sets” section in the upper-left show all published Jobs:

As can be seen here, two Jobs are published, both from the same Record Group, but with different Publish Set IDs.



Fig. 95: Column in Jobs table for publishing a Job

[Records](#) [Mapped Fields](#) [Publish](#) [Input Jobs](#) [Validation](#) [DPLA Bulk Data Matches](#) [Harvest Details](#) [Export](#) [Spark Details](#)

Publish Job

This job is currently **unpublished**. Publish using the form below:

Publish Set Id

[Publish Job](#)

Fig. 96: Screenshot to publish a Job

Records	Mapped Fields	Publish	Input Jobs	Validation	DPLA Bulk Data Matches	Merge / Duplicate Details	Export	Spark Details
---------	---------------	---------	------------	------------	------------------------	---------------------------	--------	---------------

Publish Job

This Job is currently **published**:

Publish Set ID	Record Count	Actions
test_publish_set	761	<button>Unpublish</button>

Fig. 97: Screenshot of a published Job, with option to unpublish

Published Sets

Showing all published Record Groups, where one Job from each Record Group may be published with an optional **Publish Set ID**. This **Publish Set ID** is used during publishing for the outgoing OAI set. In some cases this may be **"None"**, resulting in Records that are not aggregated under an OAI set, but will be returned via **ListRecords**.

Publish Set ID	Record Group	Published Job	Record Count	Action
rencen	Digital Collections	Rencen Collection	323	<button>Unpublish</button>
testing	Digital Collections	MergeJob @ Aug. 01, 2018, 2:35:31 PM	761	<button>Unpublish</button>
		Total:	1084	

Fig. 98: Published Jobs

To the right, is an area called “Analysis” that allows for running an **Analysis Job** over *all* published records. While this would be possible from a manually started Analysis Job, carefully selecting all Publish Jobs throughout Combine, this is a convenience option to begin an Analysis Jobs with all published Records as input.

Below these two sections is a table of all published Records. Similar to tables of Records from a Job, this table also contains some unique columns specific to Published Records:

- **Outgoing OAI Set** - the OAI set, aka the Publish Set ID, that the Record belongs to
- **Harvested OAI Set** - the OAI set that the Record was *harvested* under (empty if not harvested via OAI-PMH)
- **Unique Record ID** - whether or not the Record ID (`record_id`) is unique among all Published Records

Next, there is a now hopefully familiar breakdown of mapped fields, but this time, for all published Records.

While helpful in the Job setting, this breakdown can be particularly helpful for analyzing the distribution of metadata across Records that are slated for Publishing.

For example: **determining if all records have an access URL**. Once the mapped field has been identified as where this information should be – in this case `mods_location_url_usage=primary` – we can search for this field and confirm that 100% of Records have a value for that mapped field.

More on this in [Analyzing Indexed Fields Breakdown](#).

Records Mapped Fields Outgoing OAI-PMH Server Export Flat Files

Published Records

The table below shows all published records, across all Record Groups and OAI sets.

Show 10 entries Search:

ID	Record ID	Record Group	Publsh Set ID	Harvested OAI set	Unique Record ID	Document
10521222	oai:digital.library.wayne.edu:wayne:RENCEN18s	Digital Collections	rencen	wayne:collectionrencen	False	Valid XML
10521223	oai:digital.library.wayne.edu:wayne:RENCEN05c	Digital Collections	rencen	wayne:collectionrencen	False	Valid XML
10521224	oai:digital.library.wayne.edu:wayne:RENCEN09f	Digital Collections	rencen	wayne:collectionrencen	False	Valid XML
10521225	oai:digital.library.wayne.edu:wayne:RENCEN06b	Digital Collections	rencen	wayne:collectionrencen	False	Valid XML
10521226	oai:digital.library.wayne.edu:wayne:RENCEN10i	Digital Collections	rencen	wayne:collectionrencen	False	Valid XML
10521227	oai:digital.library.wayne.edu:wayne:RENCEN16d	Digital Collections	rencen	wayne:collectionrencen	False	Valid XML
10521228	oai:digital.library.wayne.edu:wayne:RENCEN29d	Digital Collections	rencen	wayne:collectionrencen	False	Valid XML
10521229	oai:digital.library.wayne.edu:wayne:RENCEN09m	Digital Collections	rencen	wayne:collectionrencen	False	Valid XML
10521230	oai:digital.library.wayne.edu:wayne:RENCEN18p	Digital Collections	rencen	wayne:collectionrencen	False	Valid XML
10521231	oai:digital.library.wayne.edu:wayne:RENCEN29h	Digital Collections	rencen	wayne:collectionrencen	False	Valid XML

Showing 1 to 10 of 1,084 entries

Previous 1 2 3 4 5 ... 109 Next

Fig. 99: Table showing all Published Records

Records Mapped Fields Outgoing OAI-PMH Server Export Flat Files

Mapped Fields Analysis

What do these numbers mean?

Show 100 entries Search:

Field Name	Documents with Field (of total 1084)	Documents without	Count of Total Values for Field	Count of Distinct Values for Field	Percentage of Field Values that are Unique	Percentage of Documents with Field
mods_abstract	793	291	710	327	46%	73%
mods_accessCondition	1084	0	438	1	0%	100%
mods_classification	40	1044	40	39	98%	4%
mods_extension_bisMo	437	647	437	416	95%	40%
mods_extension_elecBisMo	217	867	217	208	96%	20%
mods_extension_originalTitle	646	438	646	318	49%	60%
mods_extension_PID	1084	0	1084	767	71%	100%
mods_extension_subject_topic	6	1078	6	1	17%	1%
mods_genre	8	1076	8	2	25%	1%
mods_identifier	1084	0	1270	932	73%	100%
mods_language_languageTerm	438	646	521	14	3%	40%
mods_location_url	761	323	1521	1525	100%	70%
mods_location_url_access-preview	323	761	323	319	99%	30%
mods_location_url_usage-primary	323	761	323	327	101%	30%
mods_name_namePart	392	692	1049	704	67%	36%
mods_name_roleTerm	367	717	439	13	3%	34%
mods_note	438	646	742	226	30%	40%
mods_xsi:nil	0	1084	0	0	0%	0%

Fig. 100: Screenshot of Mapped Fields across ALL published Records

Show 100 entries Search: mods_location_url

Field Name	Documents with Field (of total 1084)	Documents without	Count of Total Values for Field	Count of Distinct Values for Field	Percentage of Field Values that are Unique	Percentage of Documents with Field
mods_location_url_access-preview	1084	0	1084	797	74%	100%
mods_location_url_usage-primary	1084	0	1084	782	72%	100%

Showing 1 to 2 of 2 entries (filtered from 61 total entries)

Previous 1 Next

Fig. 101: Confirm important field exists in published Records

3.9.4 OAI-PMH Server

Combine comes with a built-in OAI-PMH server that serves records directly from the MySQL database via the OAI-PMH protocol. This can be found under the “Outgoing OAI-PMH Server” tab:

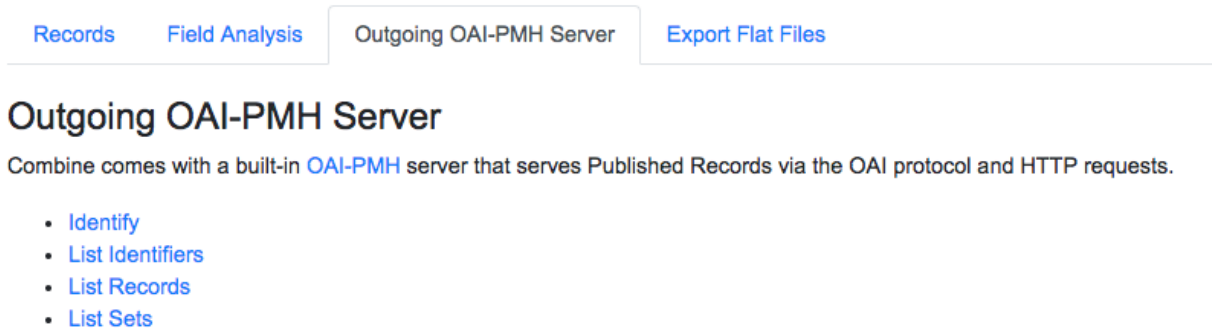


Fig. 102: Simple set of links that expose some of Combine’s built-in OAI-PMH server routes

3.9.5 Export Flat Files

Another way to “publish” or distribute Records from Combine is by exporting flat files of Record XML documents as an archive file. This can be done by clicking the “Export” tab and then “Export Documents”. [Read more about exporting here.](#)

Publish Set IDs will be used to organize the exported XML files in the resulting archive file. For example, if a single Job was published under the Publish ID `foo`, and two Jobs were published under the Publish ID `bar`, and the user specified 100 Record per file, the resulting export structure would look similar to this:

3.10 Analysis

In addition to supporting the actual harvesting, transformation, and publishing of metadata for aggregation purposes, Combine strives to also support the analysis of groups of Records. Analysis may include looking at the use of metadata fields across Records, or viewing the results of Validation tests performed across Records.

This section will describe some areas of Combine related to analysis. This includes *Analysis Jobs* proper, a particular kind of Job in Combine, and analysis more broadly when looking at the results of Jobs and their Records.

3.10.1 Analysis Jobs

Analysis Jobs are a bit of an island. On the back-end, they are essentially Duplicate / Merge Jobs, and have the same input and configuration requirements. They can pull input Jobs from across Organizations and Records Groups.

Analysis Jobs *differ* in that they do not exist within a Record Group. They are imagined to be ephemeral, disposable Jobs used entirely for analysis purposes.

You can see previously run, or start a new Analysis Job, from the “Analysis” link from the top-most navigation.

Below, is an example of an Analysis Job comparing two Jobs, from *different* Record Groups. This ability to pull Jobs from different Record Groups is shared with Merge Jobs. You can see only one Job in the table, but the entire lineage of what Jobs contribute to this Analysis Job. When the Analysis Job is deleted, none of the other Jobs will be touched (and currently, they are not aware of the Analysis Job in their own lineage).

▼	published_documents	--	Folder
▼	bar	--	Folder
	part-00000.xml	595 KB	XML Document
	part-00001.xml	529 KB	XML Document
	part-00002.xml	388 KB	XML Document
	part-00003.xml	253 KB	XML Document
	part-00004.xml	148 KB	XML Document
	part-00005.xml	118 KB	XML Document
	part-00006.xml	152 KB	XML Document
	part-00007.xml	270 KB	XML Document
	part-00008.xml	370 KB	XML Document
	part-00009.xml	447 KB	XML Document
	part-00010.xml	500 KB	XML Document
	part-00011.xml	544 KB	XML Document
	part-00012.xml	600 KB	XML Document
	part-00013.xml	671 KB	XML Document
▼	foo	--	Folder
	part-00000.xml	340 KB	XML Document
	part-00001.xml	350 KB	XML Document
	part-00002.xml	355 KB	XML Document
	part-00003.xml	364 KB	XML Document
	part-00004.xml	338 KB	XML Document

Fig. 103: Publish IDs as folder structured in exported Published Records

Home / Analysis

Analysis

This page provides the ability to view, and run, analysis of Jobs. These special "Analysis" jobs are designed explicitly for analyzing pre-existing jobs, and are not included in the workflow or progression of records for publishing.

Jobs

Showing 100 entries

Job ID	Timestamp	Name	Job Type	Livy Status	Finished	Is Valid	Elapsed	Input	Notes	Total Record Count	Monitor	Actions
262	May 14, 2018, 4:41 p.m.	Validation check across two record groups	AnalysisJob	available	True	False	0:00:19 63.7 r/s	MergeJob @ May. 10, 2018, 1:19:51 PM, Job #257, from Record Group: Fedora Repository Job C, Job #260, from Record Group: merge_shunt	None	1199	Livy / Spark	Details

Showing 1 to 1 of 1 entries

Previous 1 Next

Fig. 104: Analysis Job showing analysis of two Jobs, across two different Record Groups

3.10.2 Analyzing Indexed Fields

Undoubtedly one of Combine’s more interesting, confusing, and potentially powerful areas is the indexing of Record’s XML into ElasticSearch. This section will outline how that happens, and some possible insights that can be gleaned from the results.

How and Why?

All Records in Combine store their raw metadata as XML in MySQL. With that raw metadata, are some other fields about validity, internal identifiers, etc., as they relate to the Record. But, because the metadata is still an opaque XML “blob” at this point, it does not allow for inspection or analysis. To this end, when all Jobs are run, all Records are also **indexed** in ElasticSearch.

As many who have worked with complex metadata can attest to, flattening or mapping hierarchical metadata to a flat document store like ElasticSearch or Solr is difficult. Combine approaches this problem by generically flattening all elements in a Record’s XML document into XPath paths, which are converted into field names that are stored in ElasticSearch. This includes attributes as well, further dynamically defining the ElasticSearch field name.

For example, the following XML metadata element:

```
<mods:accessCondition type="useAndReproduction">This book is in the public domain.</
↳mods:accessCondition>
```

would become the following ElasticSearch field name:

```
mods_accessCondition_@type_useAndReproduction
```

While `mods_accessCondition_@type_useAndReproduction` is not terribly pleasant to look at, it’s telling where this value came from inside the XML document. And most importantly, this generic XPath flattening approach can be applied across all XML documents that Combine might encounter.

When running Jobs, users can select what “Index Mapper” to use, and a user may notice in addition to the Generic XPath based mapper, which is outlined above, Combine also ships with another mapper called Custom MODS mapper. This is mentioned to point out that other, custom mappers could be created and used if desired.

The Custom MODS mapper is based on an old XSLT flattening map from MODS to Solr that early versions of Islandora used. The results from this mapper result in far fewer indexed fields, which has pros and cons. If the mapping is known and tightly controlled, this could be helpful for precise analysis of where information is going. But, the generic mapper will – in some way – map all values from the XML record to ElasticSearch for analysis, albeit with unsightly field names. Choices, choices!

Creating a custom mapper would require writing a new class in the file `core/spark/es.py`, matching the functionality of a pre-existing mapper like `class GenericMapper(BaseMapper)`.

Breakdown of indexed fields for a Job

When viewing the details of a Job, the tab “Field Analysis” shows a breakdown of all fields, for all documents in ElasticSearch, from this job in a table. These are essentially facets.

There is a button “Show field analysis explanation” that outlines what the various columns mean:

All columns are sortable, and some are linked out to another view that drills further into that particular field. One way to drill down into a field is to click on the field name itself. This will present another view with values from that field. Below is doing that for the field `mods_subject_topic`:

At the top, you can see some high-level metrics that recreate numbers from the overview, such as:

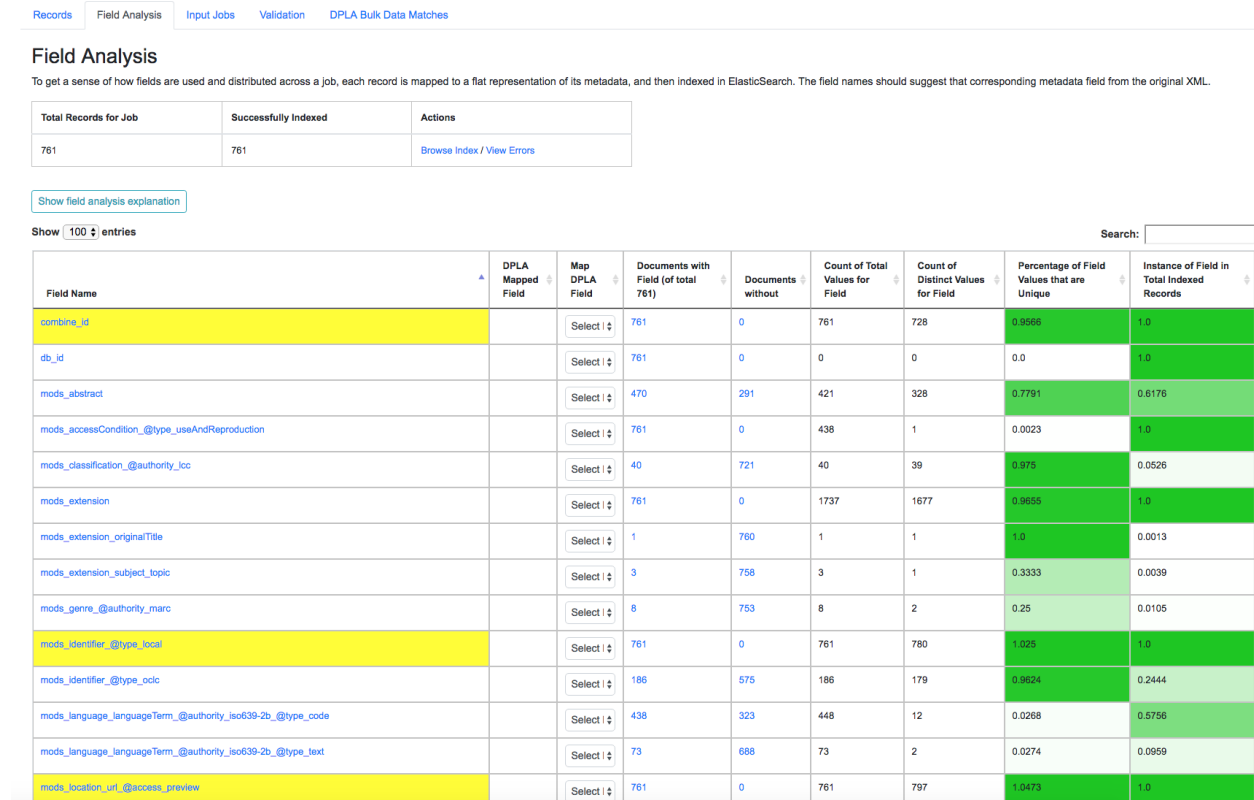


Fig. 105: Example of Field Analysis tab from Job details, showing all indexed fields for a Job

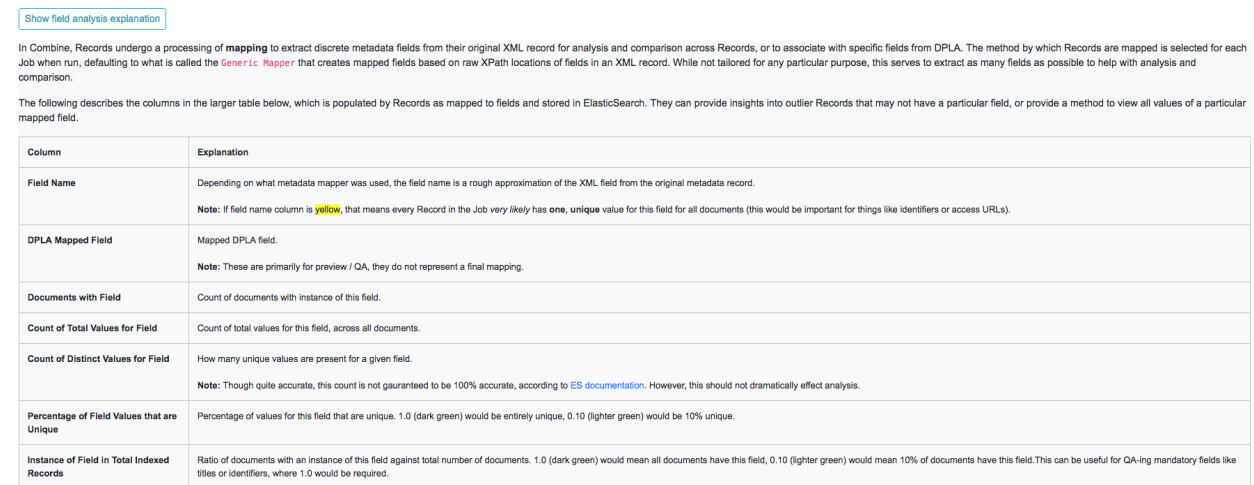


Fig. 106: Collapsible explanation of indexed fields breakdown table

Field Analysis: **mods_subject_topic**

Metrics

Documents with Field	Documents without	Count of Total Values for Field	Count of Distinct Values for Field	Percentage of Field Values that are Unique	Instance of Field in Total Indexed Records
733	28	2092	388	0.1855	0.9832

[Show field analysis explanation](#)

Values

The following table shows values for the field **mods_subject_topic** across all documents for this job's Elasticsearch index, with a count for how many times that value occurs.

Show entries Search:

Field Value	Count	Actions
Construction industry	309	Records with value / Records without
Cranes, derricks, etc.	168	Records with value / Records without
Construction equipment	163	Records with value / Records without
Juvenile fiction	148	Records with value / Records without
Scaffolding	130	Records with value / Records without
Conduct of life	83	Records with value / Records without
Juvenile literature	61	Records with value / Records without
Children's stories	55	Records with value / Records without
Fairy tales	46	Records with value / Records without
Children	39	Records with value / Records without

Showing 1 to 10 of 398 entries Previous 2 3 4 5 ... 40 Next

Fig. 107: Drill down to **mods_subject_topic** indexed field

- how many documents have this field
- how many do not
- how many total *values* are there, remembering that a single document can have multiple values
- how many *distinct* values are there
- percentage of unique (distinct / total values)
- and percentage of all documents that have this field

In the table, you can see actual values for the field, with counts across documents in this Job. In the last column, you can click to see Records that **have** or **do not have** this particular value for this particular field.

Clicking into a subject like “fairy tales”, we get the following screen:

Field Analysis: **mods_subject_topic**

Showing records where **mods_subject_topic** matches or includes “Fairy tales”

Show entries Search:

DB ID	Combine ID	Record ID	mods_subject_topic Value
1595012	f756d0cf-e98c-4aa7-a2c0-e0a8f29dda5e	oai:digitallibrary.wayne.edu:wayne:trishai/18002c21501555	[Tales', 'Fairy tales']
1595452	fa7a1aa-ea89-4ab0-a1b3-3558facaab06	oai:digitallibrary.wayne.edu:wayne:Celticworld1923b48447316	[Fairy tales', 'Folklore']
1595102	f6c029a3-d9e5-43ec-90a0-02d788c0bc21	oai:digitallibrary.wayne.edu:wayne:Fairytales/1818b51075003	Fairy tales
1595017	e58a8332-c4aa-4937-ac8d-2a2e2a0d769c	oai:digitallibrary.wayne.edu:wayne:Fairytales/1827b51074989	Fairy tales
1595080	d586cc5-b4cf-4825-8f21-aa3a4208b943	oai:digitallibrary.wayne.edu:wayne:firstofm/1861b51098251	[Fairy tales', 'English drama']
1595374	d6cd2bbb-cb77-4a35-8ee1-2c308c250a3c	oai:digitallibrary.wayne.edu:wayne:Oriental/1854b22342618	[Tales', 'Juvenile fiction', 'Fairy tales']
1595729	dab89ec0-61db-449e-bef8-a958871ea716	oai:digitallibrary.wayne.edu:wayne:fairygt/1840b51098305	[Fairy tales', 'Fairy tales', 'Translations into English']
1595107	d5c8366a-8d04-4b75-8c31-b31b5015a3d1	oai:digitallibrary.wayne.edu:wayne:blindfar/1830b19546452	[Blindness', 'Juvenile literature', 'Christian life', 'Juvenile fiction', 'Fairy tales']
1595310	d51f3ab4-46a3-450f-b699-67c3aadb779a	oai:digitallibrary.wayne.edu:wayne:uglyduck/1900b50082061	Fairy tales
1595523	d4e4e408-4029-4383-8a29-a18003a80fb8	oai:digitallibrary.wayne.edu:wayne:Households/1882b48015830	Fairy tales

Showing 1 to 10 of 46 entries (filtered from 761 total entries) Previous 2 3 4 5 Next

Fig. 108: Details for “fairy tales” **mods_subject_topic** indexed field

At this level, we have the option to click into individual Records.

3.10.3 Validation Tests Results

Results for Validation Tests run on a particular Job are communicated in the following ways:

- in the Records Table from a Job’s details page
- a quick overview of all tests performed, and number passed, from a Job’s details page
- exported as an Excel or .csv from a Job’s details page
- results for each Validation test on a Record’s details page

When a Record fails *any* test from *any* applied Validation Scenario to its parent Job, it is considered “invalid”. When selecting an input Job for another Job, users have the options of selecting all Records, those that passed all validations tests, or those that may have failed one or more.

The following is a screenshot from a Job Details page, showing that one Validation Scenario was run, and 761 Records failed validation:

Validation Name	Validation Type	Record Validation Failure Count	Actions
DPLA minimum	Schematron	761	See Failures

[Run validation results report](#)

Fig. 109: Results of all Validation Tests run for this Job

Clicking into “See Failures” brings up the resulting screen:

Job Validation Scenario Failures: **DPLA minimum**

The following records had failures for the Validation Scenario, “DPLA minimum”. Click a record to see record details, including breakdown of each validation failure.

Show **10** entries Search:

Combine ID	Record ID	Validation Results Payload	Failure Count
1595734	oai:digital.library.wayne.edu:wayne:RENCEN13d	There must be a rights statement	1
1595733	oai:digital.library.wayne.edu:wayne:RENCEN07i	There must be a rights statement	1
1595732	oai:digital.library.wayne.edu:wayne:RENCEN02h	There must be a rights statement	1
1595731	oai:digital.library.wayne.edu:wayne:Children1841b22352909	There must be a rights statement	1
1595730	oai:digital.library.wayne.edu:wayne:midsummer1908b48015866	There must be a rights statement	1
1595729	oai:digital.library.wayne.edu:wayne:fairyt1842b51099305	There must be a rights statement	1
1595728	oai:digital.library.wayne.edu:wayne:PeterPar1839b21583778	There must be a rights statement	1
1595727	oai:digital.library.wayne.edu:wayne:RENCEN17e	There must be a rights statement	1
1595726	oai:digital.library.wayne.edu:wayne:RENCEN15b	There must be a rights statement	1
1595725	oai:digital.library.wayne.edu:wayne:PoorCook1869b17887902	There must be a rights statement	1

Showing 1 to 10 of 761 entries Previous **1** 2 3 4 5 ... 77 Next

Fig. 110: Table of all Validation failures, for a particular Validation, for a Job

The column **Validation Results Payload** contains the message from the Validation Test (results may be generated from Schematron, or Python, and there may be multiple results), and the **Failure Count** column shows how many specific tests were failed for that Record (a single Validation Scenario may contain multiple individual tests).

Clicking into a single Record from this table will reveal the Record details page, which has its own area dedicated to what Validation Tests it may have failed:

From this screen, it is possible to Run the Validation and receive the raw results from the “Run Validation” link:

Or, a user can send this single Record to the Validation testing area to re-run validation scenarios, or test new ones, by clicking the “Test Validation Scenario on this Record” button. From this page, it is possible select pre-existing

Record XML Indexed Fields Record Stages Validation DPLA Link Job Type Specific

Validation Result: **Failed**

The following Validation Scenarios were run for this Job.

Validation Scenario	Test Failed	Actions
DPLA minimum	There must be a rights statement	Run Validation

Test Validation Scenario on this Record

Fig. 111: Record's Validation Results tab

```
<svrl:schematron-output xmlns:svrl="http://purl.oclc.org/dsdl/svrl" xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:schold="http://www.ascc.net/xml/schematron" xmlns:sch="http://www.ascc.net/xml/schematron"
  xmlns:iso="http://purl.oclc.org/dsdl/schematron" xmlns:mods="http://www.loc.gov/mods/v3" title="" schemaVersion="">
  <!--      &#160;
              &#160;
              &#160;
              -->
  <svrl:ns-prefix-in-attribute-values uri="http://www.loc.gov/mods/v3" prefix="mods"/>
  <svrl:active-pattern name="Required Elements for Each MODS record"/>
  <svrl:fired-rule context="mods:mods"/>
  <svrl:failed-assert test="count(mods:accessCondition[@type='use and reproduction'])=1" location="/*[local-
    name()='mods' and namespace-uri()='http://www.loc.gov/mods/v3']">
    <svrl:text>There must be a rights statement</svrl:text>
  </svrl:failed-assert>
  <svrl:active-pattern name="Subelements and Attributes used in TitleInfo"/>
  <svrl:fired-rule context="mods:mods/mods:titleInfo"/>
  <svrl:fired-rule context="mods:mods/mods:titleInfo/*"/>
  <svrl:active-pattern name="Additional URL requirements"/>
  <svrl:fired-rule context="mods:mods/mods:location/mods:url"/>
  <svrl:fired-rule context="mods:mods/mods:location/mods:url"/>
</svrl:schematron-output>
```

Fig. 112: Raw Schematron validation results

Validation Scenarios to apply to this Record in real-time, users can then edit those to test, or try completely new ones (see [Validation Scenarios for more on testing](#)):

3.11 Re-Running Jobs

3.11.1 Overview

Jobs in Combine may be “re-run” in a way that makes a series of interlinked Jobs resemble that of a “pipeline”. This functionality can be particularly helpful when a series of harvests, merges, validations, transformations, and other checks, will be repeated in the future, but with new and/or refresh records.

When a Job is re-run, the following actions are performed in preparation:

- all Records for that Job are dropped from the DB
- all mapped fields in ElasticSearch are dropped (the ElasticSearch index)
- all validations, DPLA bulk data tests, and other information that is based on the Records are removed

However, what remains is important:

- the Job ID, name, notes
- all configurations that were used
 - field mapping
 - validations applied
 - input filters

[illegible]

Fig. 113: Validation Scenario testing screen

– etc.

- linkages to other Jobs; what Jobs were used as input, and what Jobs used *this* Job as input
- publish status of a Job, with corresponding `publish_set_id` (if a Job is published before re-running, the updated Records will automatically publish as well)

3.11.2 Examples

Consider the example below, with five Jobs in Combine:

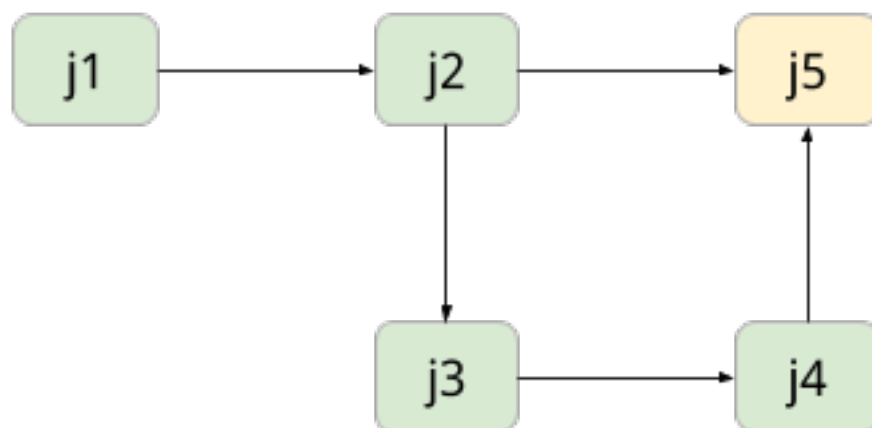


Fig. 114: Hypothetical “family” or “lineage” of Jobs in Combine

In this figure, Records from an OAI Harvest J_1 are used as input for J_2 . A subset of these are passed to J_3 , perhaps failing some kind of validation, and are fixed then in J_4 . J_5 is a final merge of the valid records from J_2 and J_4 , resulting in a final form of the Records. At each hop, there may be various validations and mappings to support the validation and movement of Records.

Now, let's assume the entire workflow is needed again, but we know that J_1 needs to re-harvest Records because there are new or altered Records. Without re-running Jobs in Combine, it would be necessary to recreate each hop in this pipeline, thereby also duplicating the amount of Records. Duplication of Records may be beneficial in some cases, but not all. In this example, a user would only need to re-run Job J_1 , which would trigger all “downstream” Jobs, all the way to Job J_5 .

Let's look at a more realistic example, with actual Combine Jobs. Take the following:

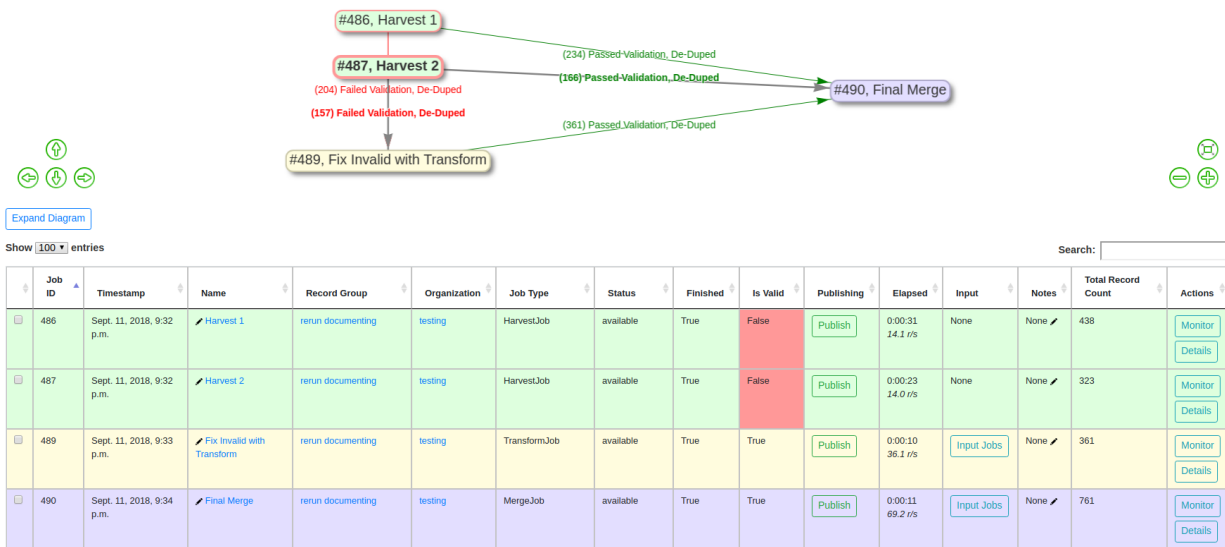


Fig. 115: Combine Re-Run “Pipeline”

In this “pipeline”:

- two OAI harvests are performed
- all *invalid* Records are sent to a Transform that fixes validation problems
- all *valid* Records from that Transform Job, and the original Harvests, are merged together in a final Job

The details of these hops are hidden from this image, but there are validations, field mappings, and other configurations at play here. If a re-harvest is needed for one, or both, of the OAI harvests, a re-run of those Jobs will trigger all “downstream” Jobs, refreshing the Records along the way.

If we were to re-run the two Harvest Jobs, we are immediately kicked back to the Record Group screen, where it can be observed that all Jobs have 0 Records, and are currently running or queued to run:

3.12 Exporting

3.12.1 Exporting Records

Records can be exported in two ways: a series of XML files aggregating the XML document for each Record, or the Mapped Fields for each Record as structured data. Records from a Job, or all Published Records, may be exported. Both are found under the “Export” tab in their respective screens.

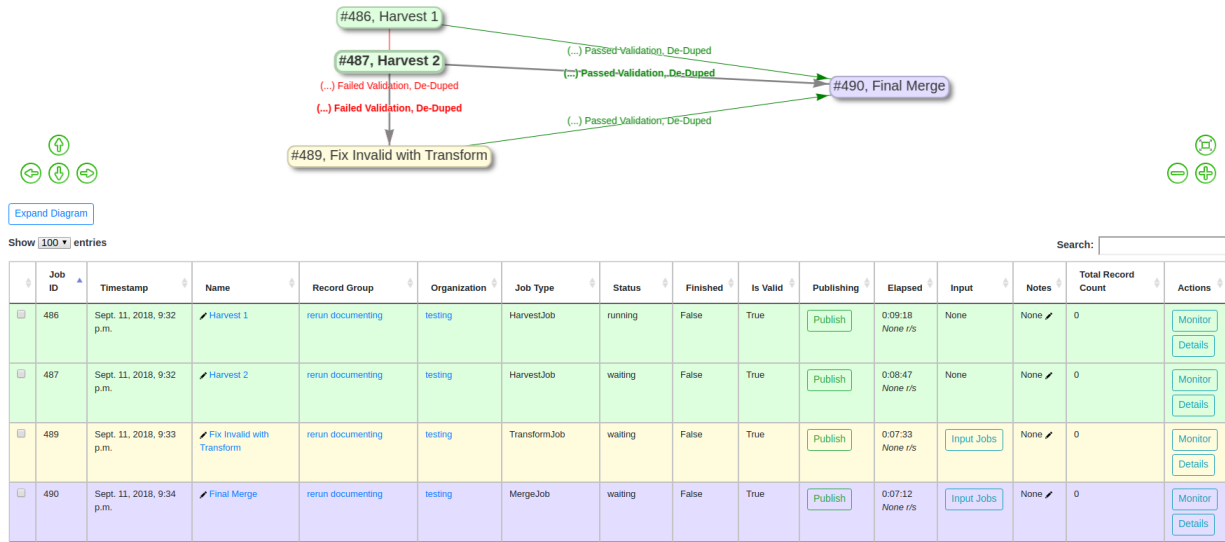


Fig. 116: Re-Run triggered, Jobs running and/or queued

Export XML Documents

Exporting documents will export the XML document for all Records in a Job or published, distributed across a series of XML files with an optional number of Records per file and a root element `<root>` to contain them. This is for ease of working with outside of Combine, where a single XML document containing 50k, 500k, 1m records is cumbersome to work with. The default is 500 Records per file.

Records may be exported in a variety of ways, [see documentation for more information](#):

- **Documents (XML):** XML documents for each Record, as harvested and/or transformed.
- **Mapped Fields (JSON):** Mapped fields as line-delimited JSON rows. *Recommended for mapped field export.*
- **Mapped Fields (CSV):** Mapped fields as a tabular, .csv file.

Select a tab below to export Records as **full XML Documents**, **Mapped Fields as JSON**, or **Mapped Fields as CSV**

Export Documents

[Export Mapped Fields](#)

XML Records per file

e.g. 500 (default)

Optional

Select archive file type

Compressed Zip file

Export Documents

Fig. 117: Export Documents tab

You may enter how many records per file, and what kind of compression to use (if any) on the output archive file.

Export Mapped Fields

Mapped fields from Records may also be exported, in one of two ways:

- Line-delimited JSON documents (**recommended**)
- Comma-separated, tabular .csv file

Both default to exporting all fields, but these may be limited by selecting specific fields to include in the export by clicking the “Select Mapped Fields for Export”.

Both styles may be exported with an optional compression for output.

JSON Documents

This is the preferred way to export mapped fields, as it handles characters for field values that may disrupt column delimiters and/or newlines.

Select export format:

JSON (line-delimited, recommended)

Note: Mapped fields exported as .csv are more prone to messy data due to column delimiters and newline characters that may exist within field values. Exporting Mapped Fields as JSON handles this more deftly, and is recommended.

Select archive file type

Uncompressed

By default, all mapped fields are included in the report, but you may optionally select a smaller subset of fields to export by clicking the button below.

Select Mapped Fields for Export

Export Mapped Fields

Fig. 118: Export Mapped Fields as JSON documents

Combine uses [ElasticSearch-Dump](#) to export Records as line-delimited JSON documents. This library handles well special characters and newlines, and as such, is recommended. This output format also handles multivalued fields and maintains field type (integer, string).

CSV

Alternatively, mapped fields can be exported as comma-separated, tabular data in .csv format. As mentioned, this does not as deftly handle characters that may disrupt column delimiters

If a Record contains a mapped field such as `mods_subject_topic` that is repeating, the default export format is to create multiple columns in the export, appending an integer for each instance of that field, e.g.,

```
mods_subject_topic.0, mods_subject_topic.1, mods_subject_topic.0
history, michigan, snow
```

But if the checkbox, Export CSV "Kibana style"? is checked, all multi-valued fields will export in the “Kibana style” where a single column is added to the export and the values are comma separated, e.g.,

```
mods_subject_topic
history,michigan,snow
```

Select export format:

JSON (line-delimited, recommended)

Note: Mapped fields exported as .csv are more prone to messy data due to column delimiters and newline characters that may exist within field values. Exporting Mapped Fields as JSON handles this more deftly, and is recommended.

Select archive file type

Uncompressed

By default, all mapped fields are included in the report, but you may optionally select a smaller subset of fields to export by clicking the button below.

[Select Mapped Fields for Export](#)
[Export Mapped Fields](#)

Fig. 119: Export Mapped Fields as JSON documents

3.13 Background Tasks

Combine includes a section for viewing and managing long running tasks. This can be accessed from the “Background Tasks” link at the top-most navigation from any page.

Note: Background tasks do *not* include normal workflow Jobs such as Harvests, Transforms, Merges, etc., but do include the following tasks in Combine:

- deleting of Organizations, Record Groups, or Jobs
- generating reports of Validations run for a Job
- exportings Jobs as mapped fields or XML documents
- re-indexing Job with optionally changed mapping parameters
- running new / removing validations from Job

The following screenshot shows a table of all Background Tasks, and their current status:

Home / Background Tasks

Background Tasks

Some tasks in Combine are long running and must be run in the background. This includes **Job Deletion**, **Validation Reports**, **Exports**, among other things. See the table below for the status of these tasks, and any further actions that be taken such as downloading or viewing results.

Hold Shift while clicking a column header to sort by multiple columns

Show 10 entries

DB ID	Started	Name	Type	Task ID	Status	Duration	Actions
346	2018-06-11T14:39:06.791Z	Delete Job: #py trans, new - cold (DELETING)	Job Deletion	urn:uuid:abf2c7e7-256a-4956-a1ff-f971de0b24c5	Finished	0:00:03	Results Delete
345	2018-06-11T14:37:43.371Z	Delete Job: #py trans, new - cold (DELETING)	Job Deletion	urn:uuid:c52ee3ab-alc4-4e94-b3fe-31ba96d38385	Finished	0:00:05	Results Delete
344	2018-06-11T14:02:23.905Z	Delete Job: #MergeJob @ Jun. 08, 2018, 6:56:53 PM (DELETING)	Job Deletion	urn:uuid:84e821d7-c697-44ce-aba5-985fb83681f	Finished	0:00:02	Results Delete
343	2018-06-08T19:22:01.355Z	Validation Report: testing_testing	Validation Report Generation	urn:uuid:87449c33-3235-4a32-8e75-87dae2a9eb2	Finished	0:00:05	Results Delete
342	2018-06-08T18:55:47.783Z	Export Mapped Fields for Job: HarvestStaticXMLJob @ Jun. 08, 2018, 6:54:48 PM	Job Export Mapped Fields	urn:uuid:f46e4ae7-e283-4ed5-b016-c6784812b2ca	Finished	0:00:01	Results Delete

Showing 41 to 45 of 45 entries

Previous 1 2 3 4 5 Next

Manage Tasks

[Remove All Tasks](#)

Fig. 120: Table of running and completed Background Tasks

Clicking on the “Results” button for a single task will take you to details about that task:

Home / Background Tasks / Task - Export Mapped Fields for Job: HarvestStaticXMLJob @ Jun. 08, 2018, 6:54:48 PM

Combine Background Task: **Export Mapped Fields for Job: HarvestStaticXMLJob @ Jun. 08, 2018, 6:54:48 PM**

Type	Started	Duration	Task ID
Job Export Mapped Fields	June 8, 2018, 6:55 p.m.	0:00:01	uptr:ouid:f46e4ae7-e283-4e05-b010-c570401202ca

Task Details

[Download Mapped Fields .csv](#)

Task Output

```
{
  "name": "job_341_mapped_fields.csv",
  "export_output": "/tmp/job_341_mapped_fields.csv"
}
```

Task Parameters

```
{
  "kibana_style": false,
  "job_id": 341
}
```

[Delete this Task](#)

Fig. 121: Example of completed Job Export Background Task

The results page for each task type will be slightly different, depending on what further actions may be taken, but an example would be a download link (as in the figure above) for job export or validation reports.

3.14 Tests

Though Combine is by and large a Django application, it has characteristics that do not lend themselves towards using the built-in Django unit tests. Namely, DB tables that are not managed by Django, and as such, would not be created in the test DB scaffolding that Django tests usually use.

Instead, Combine uses out-of-the-box `pytest` for unit tests.

3.14.1 Demo data

In the directory `/tests`, some demo data is provided for simulating harvest, transform, merge, and publishing records.

- **mods_250.xml** - 250 MODS records, as returned from an OAI-PMH response
 - during testing this file is parsed, and 250 discrete XML files are written to a temp location to be used for a test static XML harvest
- **mods_transform.xsl** - XSL transformation that performs transformations on the records from **mods_250.xml**
 - during transformation, this XSL file is added as a temporary transformation scenario, then removed post-testing

3.14.2 Running tests

Note: *Because Combine currently only allows one job to run at a time, and these tests are essentially small jobs that will be run, it is important that no other jobs are running in Combine while running tests.*

Tests should be run from the root directory of Combine, if Ansible or Vagrant builds, likely at `/opt/combine`. Also requires sourcing the anaconda Combine environment with `source activate combine`.

It is worth noting whether or not there is an active Livy session already for Combine, or if one should be created and destroyed for testing. Combine, at least at the time of this writing, operates only with a single Livy instance. By

default, tests will create and destroy a Livy session, but this can be skipped in favor of using an active session by including the flag `--use_active_livy`.

Testing creates a test `Organization`, `RecordGroup`, and `Job`'s during testing. By default, these are removed after testing, but can be kept for viewing or analysis by including the flag `--keep_records`.

3.14.3 Examples

run tests, no output, create Livy session, destroy records

```
pytest
```

run tests, see output, use active Livy session, keep records after test

```
pytest -s --use_active_livy --keep_records
```

3.15 Command Line

Though Combine is designed primarily as a GUI interface, the command line provides a powerful and rich interface to the models and methods that make up the Combine data model. This documentation is meant to expose some of those patterns and conventions.

There are two primary command line contexts:

- **Django shell:** A shell that loads all Django models, with some additional methods for interacting with Jobs, Records, etc.
- **Pyspark shell:** A pyspark shell that is useful for interacting with Jobs and Records via a spark context.

These are described in more detail below.

Note: For both, the Combine [Miniconda](#) python environment must be used, which can be activated from any filepath location by typing:

```
source activate combine
```

3.15.1 Django Python Shell

Starting

From the location `/opt/combine` run the following:

```
./runconsole.py
```

Useful and Example Commands

Convenience methods for retrieving instances of Organizations, Record Groups, Jobs, Records

```
'''
Most all convenience methods are expecting a DB identifier for instance retrieval
'''
```

(continues on next page)

(continued from previous page)

```
# retrieve Organization #14
org = get_o(14)

# retrieve Record Group #18
rg = get_rg(18)

# retrieve Job #308
j = get_j(308)

# retrieve Record by id '5ba45e3f01762c474340e4de'
r = get_r('5ba45e3f01762c474340e4de')

# confirm these retrievals
'''
In [2]: org
Out[2]: <Organization: Organization: SuperOrg>
In [5]: rg
Out[5]: <RecordGroup: Record Group: TurboRG>
In [8]: j
Out[8]: <Job: TransformJob @ May. 30, 2018, 4:10:21 PM, Job #308, from Record Group: TurboRG>
In [10]: r
Out[10]: <Record: Record: 5ba45e3f01762c474340e4de, record_id: 0142feb40e122a7764e84630c0150f67, Job: MergeJob @ Sep. 21, 2018, 2:57:59 AM>
'''
```

Loop through Records in Job and edit Document

This example shows how it would be possible to:

- retrieve a Job
- loop through Records of this Job
- alter Record, and save

This is not a terribly efficient way to do this, but it demonstrates the data model as accessible via the command line for Combine. A more efficient method would be to write a custom, Python snippet [Transformation Scenario](#).

```
# retrieve Job model instance
In [3]: job = get_j(563)

# loop through records via get_records() method, updating record.document (replacing
# 'foo' with 'bar') and saving
In [5]: for record in job.get_records():
...:     record.document = record.document.replace('foo', 'bar')
...:     record.save()
```

3.15.2 Pyspark Shell

The pyspark shell is an instance of Pyspark, with some configurations that allow for loading models from Combine.

Note:

The pyspark shell requires the Hadoop Datanode and Namenode to be active. These are likely running by default, but in the event they are not, they can be started with the following (Note: the trailing `:` is required, as that indicates a group of processes in [Supervisor](#)):

```
sudo supervisorctl restart hdfs:
```

Note:

The pyspark shell when invoked as described below, will be launched in the same Spark cluster that Combine's Livy instance uses. Depending on available resources, it's likely that users will need to **stop** any active Livy sessions as [outlined here](#) to allow this pyspark shell the resources to run.

Starting

From the location `/opt/combine` run the following:

```
./pyspark_shell.sh
```

Useful and Example Commands

Open Records from a Job as a Pyspark DataFrame

```
# import some convenience variables, classes, and functions from core.spark.console
from core.spark.console import *

# retrieve Records from MySQL as pyspark DataFrame
'''
In this example, retrieving records from Job #308
Also of note, must pass spark instance as first argument to convenience method,
which is provided by pyspark context
'''
job_df = get_job_as_df(spark, 308)

# confirm retrieval okay
job_df.count()
...
Out[5]: 250

# look at DataFrame columns
job_df.columns
Out[6]:
['id',
 'combine_id',
 'record_id',
 'document',
 'error',
 'unique',
 'unique_published',
 'job_id',
 'published',
 'oai_set',
 'success',
 'valid',
 'fingerprint']
```

3.16 Installing Combine

There are two deployment methods explained below. Choose the one that meets your needs. Please be aware that running this system requires not insignificant resources. Required is at least 8GB RAM and 2 processor cores. Combine, at its heart, is a metadata aggregating and processing framework that runs within a software called Django. It requires other components such as Elasticsearch, Spark, among others, in order to work properly. If you are looking to test-drive or develop on Combine, you have arrived at the right place.

3.16.1 Pre-Installation Notes:

- Both installation methods listed below assume an Ubuntu 18.04 server
- For either installation, there are a host of variables that set default values. They are all found in the `all.yml` file inside the `group_vars` folder.
 - If you are installing this system on a remote server, you **MUST** update the `ip_address` variable found in `all.yml`. Change it to your remote server's ip address.
 - If you are installing the system locally with Vagrant, you don't need to do anything. Your server will be available at 192.168.45.10.

3.16.2 Vagrant-based Installation (local)

- If you are looking to run an instance of the Combine ecosystem on your own computer, you will use the Vagrant-based installation method. This method assumes that you have 8GB of RAM and 2 processor cores available to devote to this system. Double-check and make sure you have this available on your computer. This means you will need **MORE** than that in RAM and cores in order to not bring your computer to a complete halt. Local testing has been performed on iMacs running MacOS Sierra that have a total of 4 cores and 16 GB of RAM.
- Install [VirtualBox](#), [Vagrant](#), and [Ansible](#), [Python](#), and [Passlib](#).
 - NB: when installing Passlib, you should be able to simply run `pip install passlib` if you have the pip tool installed. If you're not certain or if that command doesn't successfully run, see the following link for instructions on installing Pip: <http://www.pythonforbeginners.com/basics/how-to-use-pip-and-pypi>.
- Clone the following Github repository: [combine-playbook](#)
- Navigate to the repository in your favorite terminal/shell/command line interface.

Within the root directory of the repository, run the commands listed below:

- Install pre-requisites.

```
ansible-galaxy install -f -c -r requirements.yml
```

- Build the system.

```
vagrant up
```

- This installation will take a while. The command you just ran initializes the vagrant tool to manage the installation process. It will first download and install a copy of Ubuntu Linux (v.18.04) on your VirtualBox VM. Then, it will configure your networking to allow SSH access through an account called `vagrant` and make the server available only to your local computer at the IP address of 192.168.45.10. After that initial work, the vagrant tool will use ansible to provision (i.e. install all components and dependencies) to a VM on your computer.

- After completed, your server will be available at <http://192.168.45.10>. Navigating to <http://192.168.45.10/admin> will allow you to setup your system defaults (OAI endpoints, etc). Going to <http://192.168.45.10/combine> will take you to the heart of the application where you can ingest, transform, and analyze metadata. Login using the credentials the following credentials:

```
username: combine
password: combine
```

- Access via SSH is available through the accounts below. Both have sudo privileges. The combine password defaults to what is listed below. If you have edited `group_vars/all.yml` and changed the password listed there, please adjust accordingly. ““ username: combine password: combine
username: vagrant password: vagrant ““

3.16.3 Ansible-based Installation (remote server)

- If you have a remote server that you want to install the system upon, these installation instructions are for you. Your server should already be running Ubuntu 18.04. It needs to be remotely accessible through SSH from your client machine and have at least port 80 accessible. Also, it needs Python 2.7 installed on it. Your server will need at least 8GB of RAM and 2 cores, but more is better.
- Install [Ansible](#), [Python](#), and [Passlib](#) on your client machine. This installation method has not been tested using Windows as client machine, and, therefore, we offer no support for running an installation using Windows as a client. For more information, please refer to these Windows-based instructions: http://docs.ansible.com/ansible/latest/intro_windows.html#using-a-windows-control-machine
 - NB: when installing Passlib, you should be able to simply run `pip install passlib` if you have the pip tool installed. If you’re not certain or if that command doesn’t successfully run, see the following link for instructions on installing Pip: <http://www.pythonforbeginners.com/basics/how-to-use-pip-and-pypi>.
- Exchange ssh keys with your server.
 - Example command on MacOS

```
ssh-keygen -t rsa
cat ~/.ssh/id_rsa.pub | ssh USERNAME@IP_ADDRESS_OR_FQDN "mkdir -p ~/.ssh &&_
↵cat >> ~/.ssh/authorized_keys"
```

- Point ansible to remote server.
 - You do this by creating a file named `hosts` inside the following directory: `/etc/ansible`. If you are using a Linux or MacOS machine, you should have an `etc` directory, but you will probably have to create the `ansible` folder. Place your server’s IP address or FQDN in this `hosts` file. If the username you used to exchange keys with the server is anything other than root, you will have to add `ansible_user=YOUR_USERNAME`. Your hosts file could end up looking something like this: `192.168.45.10 ansible_user=USERNAME`. For more information see: http://docs.ansible.com/ansible/latest/intro_getting_started.html#your-first-commands
- Check your target machine is accessible and ansible is configured by running the following command:

```
ansible all -m ping
```

- A successful response will look something similar to this. Substitute your IP for the one listed below in the example.

```
192.168.44.10 | SUCCESS => {
  "changed": false,
```

(continues on next page)

(continued from previous page)

```
"failed": false,
"ping": "pong"
}
```

- If the response indicates a failure, it might look something like below. This below type of failure indicates that it could successfully connect to the server, but that it didn't find Python 2.7 installed on the remote server. This is fine. The important part is that it could connect to the server. The ansible playbook will automatically install Python 2.7 when it begins, so you should be fine to proceed to the next step(s).

```
192.168.44.10 | FAILED! => {
  "changed": false,
  "failed": true,
  "module_stderr": "Warning: Permanently added '192.168.44.10' (ECDSA)
↳to the list of known hosts.\r\n/bin/sh: 1: /usr/bin/python: not
↳found\n",
  "module_stdout": "",
  "msg": "MODULE FAILURE",
  "rc": 127
}
```

- Clone the following Github repository: [combine-playbook](#)
- Navigate to the repository in your favorite terminal/shell/command line interface.
- Update `ip_address` in `group_vars/all.yml`
 - Change the `ip_address` variable to your remote server's IP address.
- Within the root directory of the repository, run the commands listed below:
 - Install pre-requisites

```
ansible-galaxy install -f -c -r requirements.yml
```

- Run ansible playbook

```
ansible-playbook playbook.yml
```

- This installation will take a while. Ansible provisions the server with all of the necessary components and dependencies.
- After the installation is complete, your server will be ready for you to use Combine's web-based interface. Go to your server's IP address. Navigating to `/admin` will allow you to setup your system defaults (OAI endpoints, etc). Going to `/combine` will take you to the heart of the application where you can ingest, transform, and analyze metadata. Login using the following credentials:

```
username: combine
password: combine
```

- Access via SSH is available through the account below. It has sudo privileges. The password below is correct unless you have changed it inside `group_vars/all.yml`.

```
username: combine
password: combine
```

3.16.4 Post-Installation walkthrough

Once you do have an instance of the server up and running, you can find a [QuickStart walkthrough here](#).

3.16.5 Troubleshooting

Restarting Elasticsearch

```
sudo systemctl restart combine_elasticsearch.service
```

3.17 Tuning and Configuring Server

Combine is designed to handle sets of metadata small to large, 400 to 4,000,000 Records. Some of the major associated server components include:

- *MySQL*
 - store Records and their associated, full XML documents
 - store Transformations, Validations, and most other enduring, user defined data
 - store transactions from Validations, OAI requests, etc.
- *ElasticSearch*
 - used for indexing mapped fields from Records
 - main engine of field-level analysis
- *Apache Spark*
 - the workhorse for running Jobs, including Harvests, Transformations, Validations, etc.
- *Apache Livy*
 - used to send and queue Jobs to Spark
- *Django*
 - the GUI
- *Django Background Tasks*
 - for long running tasks that may that would otherwise prevent the GUI from being responsive
 - includes deleting, re-indexing, exporting Jobs, etc.

Given the relative complexity of this stack, and the innerconnected nature of the components, Combine is designed to be deployed via an Ansible playbook, [which you can read more about here](#). The default build requires **8g** of RAM, with the more CPU cores the better.

This part of the documentation aims to explain, and indicate how to modify or configure, some of these critical components.

3.17.1 MySQL

3.17.2 ElasticSearch

3.17.3 Apache Spark

3.17.4 Apache Livy

3.17.5 Django

3.17.6 Django Background Tasks