Bulkrax

Getting Data In and Out of Repositories

NOTCH8

NOTCH8

Rob Kaufman
Founding Partner
rob@notch.com

What is Bulkrax?

- Samvera Labs engine
- Batteries-included importer for Samvera applications
- Designed to be extensible
- Hyrax focused, but flexible
- Provides a full admin interface for imports and exports
  - Creating
  - Editing
  - Scheduling
  - Reviewing

https://github.com/samvera-labs/bulkrax

What is Bulkrax?

- Round Trip Support
- CSV
- RDF
- BagIt (CSV and RDF)
- OAI Import (export already handled by blacklight-oai gem)
- XML (various flavors have been written FoxXML, Mets, Mods)
- JSON (coming soon)

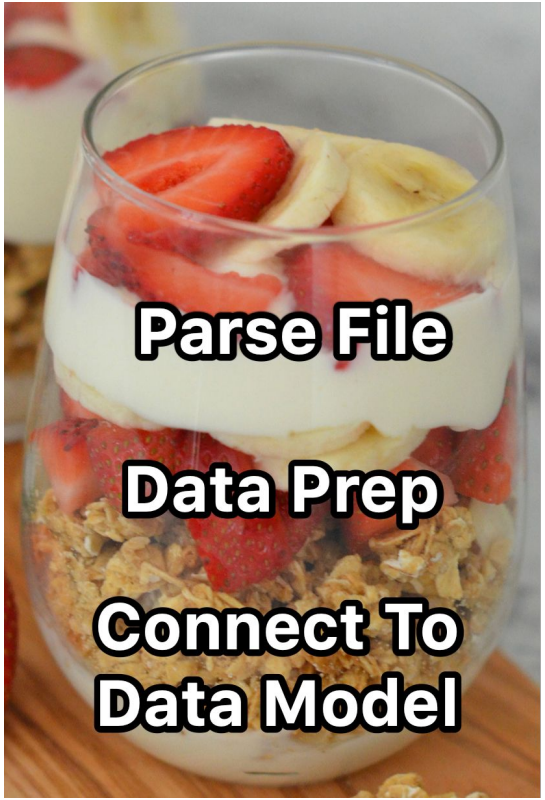
https://github.com/samvera-labs/bulkrax

A Layered Approach

You know what else everybody like? Parfaits

- *Donkey*

Layers of an Import



Parser and Entry

- The Parser class reads the incoming data stream (aka file) and is in charge of extracting individual records from the file
- This can be one record or many. This class is a PORO
- Example: OAI Parser uses the OAI Ruby Gem to read the OAI feed and converts each individual record in to a hash
- Example: The CSV Parser reads the CSV file and converts each individual row in to a hash with the keys being the column headers
- Example: The BagIt Parser unzips the bag, then reads the CSV or RDF file and connects the metadata hash with the Bags file paths.

Parser and Entry

- The Entry class is created with at minimum a hash of the raw metadata
- Entry#build will convert the raw metadata to the parsed metadata and then hand the object off to an Object Factory
- Entries are saved ActiveRecord objects. They hold state about the before and after of the parsing along with timestamp and debugging information
- They can be reprocessed individually, which makes them backgroundable, repeatable, debuggable records of what we tried to use to create a data object.

Matchers

```
config.field_mappings['Bulkrax::CsvParser'] = {
  'access_control_id' => { excluded: true },
  'add_info' => { from: ['additional_information'] },
  'admin_set_id' => { from: ['admin_set'] },
  'alt_title' => { from: ['alternative_title', 'alt_title', 'extra_title'] },
  'alternate_identifier' => { from: ['alternate_identifier'], object: 'alternate_identifier' },
  'alternate_identifier_type' => { from: ['alternate_identifier_type'], object: 'alternate_identifier' },
  'book_title' => { from: ['book_title'] },
  'ark_id' => { from: ['ark_id'], source_identifier: true },
  'collection' => { from: ['collection_id'] },
  'contributor_family_name' => { from: ['contributor_family_name'], object: 'contributor' },
  'contributor_given_name' => { from: ['contributor_given_name'], object: 'contributor' },
  'contributor_grid' => { from: ['contributor_grid'], object: 'contributor' },
  'contributor_isni' => { from: ['contributor_isni'], object: 'contributor' },
  'contributor_name_type' => { from: ['contributor_name_type'], object: 'contributor' },
  'contributor_orcid' => { from: ['contributor_orcid'], object: 'contributor' },
}
```

Matchers and Objects and ParseFields (oh my!)

- A given field has a “from” (one or more fields in the source data) and a “to” (the single destination in the data model).
- The field could be gathered up in an object: user_family_name, user_given_name, user_orcid => user: { family_name, given_name, orcid}
- It may be the source identifier, an identifier from outside the internal system that uniquely identifies the piece of data.

Matchers and Objects and ParseFields (oh my!)

- It may be a parsed field, which called a method to handle the transition like so

```
def parse_resource_type(src)
  Hyrax::ResourceTypesService.label(src.to_s.strip.titleize)
rescue KeyError
  nil
end
```

- It may be split, but only on specified characters
- It may be implied (not listed in the field_mapping)

Wrapped Up With A Bow

Once data is parsed, processed and matched we do the following:

- Save the entry with the parsed_metadata for later debugging
- Send the parsed_metadata off to the factory

It is the factories job to make the parsed_metadata hash in to a data object. Our goal of the input to the factory is to make it AS MUCH LIKE THE FORM POST as possible. This means that once the data leaves the ObjectFactory it should follow the same path as if the object were submitted via a view/controller.

Life in the Factory

- Bulkrax's default ObjectFactory assumes a Hyrax like process
- It prepares the environment object and passes in an attrs hash
- ObjectFactory makes sure files, associations etc, are just like they would be coming from the web form.
- It then hands that over to the Actor Stack
- Making a different kind of object (ActiveRecord, Valkyrie) would likely require changing the ObjectFactory to make a more AR or Valkyrie friendly params hash. **None** of the rest of Bulkrax should need to be touched for this.

Levels of Override

- Customize the field mapping in the Bulkrax initializer
- HasLocalProcessing - a concern that contains an “add_local” method. This method is called during Entry#build and is meant for overriding
- Add your own parsed_field entries
- Custom Parser: These are inheritable and can be specified from the config file including a custom form UI partial.
- Custom Entry: Override `def entry_class` to create Entry classes that are AR objects and are again, inheritable.
- Custom ObjectFactory: Override `def factory_class` in the Entry class

Next Steps

- Better control over updates with remote files (versioning of files from remote sources)
- Improved relationship support, collection and file set related metadata
- Continue to add additional parsers for new data types
- Get Gem promoted out of labs
- Ability to rerun a specific entry instead of the whole importer
- Field Mapping UI

Rob Kaufman
Founding Partner
rob@notch.com

NOTCH8

www.notch8.com
connect@notch8.com

THANK YOU

Outline

- What is Bulkrax
  - Engine for writing import and export
  - Focused on Hyrax-like applications, but flexible
  - Provides a configurable experience, and customizable parsers
- Bulkrax in Layers
  - Parsing File
  - Matching and Preparing
  - Connecting to the data model
- Parsers and Entries
- Matchers and the parsed_metadata hash
- ObjectFactory and the Actor Stack
- The config files
- Override points
  - HasLocalProcessing
  - Custom Parser and Entry classes
- Questions