Live Import from external sources

1 General Framework
- 1.1 Introduction
- 1.2 Features
- 1.3 Abstraction of input format
- 1.4 Editing Metadata Mapping
- 1.5 Transformation to DSpace Item
  - 1.5.1 Implementation of an import source for External Sources
  - 1.5.2 Implementation of an import source for files
  - 1.5.3 Mapping raw data to Metadata
  - 1.5.4 Inherited methods
  - 1.5.5 Spring configuration for External Sources
  - 1.5.6 Metadata mapping
  - 1.5.7 Available Metadata Contributor classes
- 1.6 Framework Sources Implementations
  - 1.6.1 PubMed Integration
    - 1.6.1.1 Introduction
    - 1.6.1.2 Publication Lookup URL
    - 1.6.1.3 PubMed Metadata Mapping
    - 1.6.1.4 PubMed specific classes Config
      - 1.6.1.4.1 Metadata mapping classes
      - 1.6.1.4.2 Service classes
  - 1.6.2 ArXiv Integration
    - 1.6.2.1 ArXiv Metadata Mapping

General Framework

Introduction

This framework is used by both the REST API and User Interface to help enhance or enrich submissions. One examples usage is in Importing Items via basic bibliographic formats (Endnote, BibTex, RIS, CSV, etc) and online services (arXiv, PubMed, CrossRef, CiNii, etc)

Features

- lookup publications from remote sources
- Support for multiple implementations

Abstraction of input format

The importer framework does not enforce a specific input format. Each importer implementation defines which input format it expects from a remote source. The import framework uses generics to achieve this. Each importer implementation will have a type set of the record type it receives from the remote source's response. This type set will also be used by the framework to use the correct MetadataFieldMapping for a certain implementation. Read "I mplementation of an import source" below for more information and how to enable the framework.

Editing Metadata Mapping

At a simple level, metadata mapping configurations are all in Spring configs in `[dspace.dir]/config/spring/api/`

In that directory, you'll find a mapping file per import source, e.g. "arxiv-integration.xml", "bibtex-integration.xml", "endnote-integration.xml", "pubmed-integration.xml", etc.

There are two different mapping types.

1. First, mapping from a file-based import (e.g. bibtex, endnote, ris, etc) to a DSpace metadata field.
   a. The list of all of the enabled mappings can be found in a "MetadataFieldConfig" <util:map>, usually at the top of the config file.

```
<util:map id="bibtexMetadataFieldMap" key-type="org.dspace.importer.external.metadatamapping.
MetadataFieldConfig"
             value-type="org.dspace.importer.external.metadatamapping.contributor.
MetadataContributor">
    <description>Defines which metadatum is mapped on which metadatum. Note that while the key
must be unique it
        only matters here for postprocessing of the value. The mapped MetadatumContributor has
full control over
           what metadatafield is generated.
    </description>
    <!-- These entry tags are the enabled mappings. The "value-ref" must map to a <bean> ID. -->
    <entry key-ref="dcTitle" value-ref="bibtexTitleContrib" />
    <entry key-ref="dcAuthors" value-ref="bibtexAuthorsContrib" />
    <entry key-ref="dcJournal" value-ref="bibtexJournalContrib" />
    <entry key-ref="dcIssued" value-ref="bibtexIssuedContrib" />
    <entry key-ref="dcJissn" value-ref="bibtexJissnContrib" />
</util:map>
```

b. Each field in the file is mapped to a DSpace metadata field in a "SimpleMetadataContributor" bean definition. *NOTE: a large number of DSpace defined metadata fields are already configured as MetadataFieldConfig beans in the "dublincore-metadata-mapper.xml" Spring Config in the same directory. These may be reused in other configurations.*

```
<!-- This example bean for BibTex says the "title" key in the BibTex" file should be mapped to the
DSpace metadata field
     defined in the "dcTitle" bean.  This "dcTitle" bean is found in "dublincore-metadata-mapper.
xml" and obviously maps to "dc.title" -->
<bean id="bibtexTitleContrib" class="org.dspace.importer.external.metadatamapping.contributor.
SimpleMetadataContributor">
    <property name="field" ref="dcTitle"/>
    <property name="key" value="title" />
</bean>
```

2. Second, mapping from an external API query import (e.g. arxiv, pubmed, etc) to a DSpace metadata field.
   a. Similar to above, The list of all of the enabled mappings can be found in a "MetadataFieldConfig" <util:map>, usually at the top of the config file.

```
<util:map id="arxivMetadataFieldMap" key-type="org.dspace.importer.external.metadatamapping.
MetadataFieldConfig"
             value-type="org.dspace.importer.external.metadatamapping.contributor.
MetadataContributor">
    <description>Defines which metadatum is mapped on which metadatum. Note that while the key
must be unique it
        only matters here for postprocessing of the value. The mapped MetadatumContributor has
full control over
        what metadatafield is generated.
    </description>
    <!-- These entry tags are the enabled mappings. The "value-ref" must map to a <bean> ID. -->
    <entry key-ref="arxiv.title" value-ref="arxivTitleContrib"/>
    <entry key-ref="arxiv.summary" value-ref="arxivSummaryContrib"/>
    <entry key-ref="arxiv.published" value-ref="arxivPublishedContrib"/>
    <entry key-ref="arxiv.arxiv.doi" value-ref="arxivDoiContrib"/>
    <entry key-ref="arxiv.arxiv.journal_ref" value-ref="arxivJournalContrib"/>
    <entry key-ref="arxiv.category.term" value-ref="arxivCategoryTermContrib"/>
    <entry key-ref="arxiv.author.name" value-ref="arxivAuthorContrib"/>
    <entry key-ref="arxiv.identifier.other" value-ref="arxivOtherContrib"/>
</util:map>
```

   b. Each field in the file is mapped to a DSpace metadata field, usually in a "SimpleXPathMetadatumContributor" bean definition which also uses a "MetadataFieldConfig" bean. *NOTE: a large number of DSpace defined metadata fields are already configured as MetadataFieldConfig beans in the "dublincore-metadata-mapper.xml" Spring Config in the same directory. These may be reused in other configurations.*

```
<!-- This first bean define an XPath query ("ns:title") to map to a field (ID="arxiv.title") in
DSpace -->
<bean id="arxivTitleContrib" class="org.dspace.importer.external.metadatamapping.contributor.
SimpleXpathMetadatumContributor">
    <property name="field" ref="arxiv.title"/>
    <property name="query" value="ns:title"/>
    <property name="prefixToNamespaceMapping" ref="arxivBasePrefixToNamespaceMapping"/>
</bean>
<!-- This second bean then defines which DSpace field to use when "arxiv.title" is references. In
other words, between these two beans,
     the "ns:title" XPath query value is saved to "dc.title". -->
<bean id="arxiv.title" class="org.dspace.importer.external.metadatamapping.MetadataFieldConfig">
    <constructor-arg value="dc.title"/>
</bean>
```

Transformation to DSpace Item

The framework produces an 'ImportRecord' that is completely decoupled from DSpace. It contains a set of metadata DTO's that contain the notion of schema,element and qualifier. The specific implementation is responsible for populating this set. It is then very simple to create a DSpace item from this list.

Implementation of an import source for External Sources

Each external source/API importer implementation must at least implements `org.dspace.importer.external.service.components.QuerySource`, which provides the query method used by the framework to retrieve data from the remote source (e.g. Pubmed, ArXiv, etc). Each external source importer must implements, according to the provider APIs, the declared methods.
An useful abstract for remote sources is `org.dspace.importer.external.service.components.AbstractRemoteMetadataSource`.This class contains functionality to handle request timeout and to retry requests. Using this abstract, the query method must implements java.util.concurrent.Callable.

Implementation of an import source for files

Each file importer implementation must at least implements `org.dspace.importer.external.service.components.FileSource`, which provides the basic methods used by the framework to parse and load data from the file (e.g. CSV, Endnote, etc).

Each importer must implements the method:

```
public List<ImportRecord> getRecords(InputStream inputStream) throws FileSourceException;
```

This method is responsible to transform the input data into an ImportRecord list, which will then managed by the top layer of the framework.

The conversion from raw data to an ImportRecord could be done using the framework too, using the metadata mapping structure (see below).

File sources needs to know which file extensions they have to supports. This is done by the default method `isValidSourceForFile` in `FileSource`, and is controlled by the entries in the list returned by declared method `public List<String> getSupportedExtensions();`

An useful abstract for file source is `org.dspace.importer.external.service.components.AbstractPlainMetadataSource`. It should be used whenever it is possible to model the data in the file as a list of key-value lists (e.g. for CSV files, any row is a key value list).

Mapping raw data to Metadata

The framework core is a mid-layer component which allow the conversion of raw data into metadata (ImportRecord) using xml configurable spring beans.

The core of this approach is `org.dspace.importer.externa.service.AbstractImportMetadataSourceService`. Any service that wants to generate metadata from raw data should go through this abstract.

Our service then should extends AbstractImportMetadataSourceService, and use `transformSourceRecords` to transform raw data into ImportRecords.

The most relevant concept in the framework is `private MetadataFieldMapping<RecordType, MetadataContributor<RecordType>> metadataFieldMapping`. This is the place where the framework take the mapping between row data and the associated metadatum. This map must be injected in the service, and will be used by `transformSourceRecords` to convert the data.

`RecordType` is a generic type, which rapresent a single entry of the list of data, and will be mapped to a single ImportRecord. Any metadatum will be mapped to a specific field in the RecordType using a Contributor as described in Metadata mapping.

Inherited methods

Method getImportSource() should return a unique identifier. Importer implementations should not be called directly, but class *org.dspace.importer.external.service.ImportService* should be called instead. This class contains the same methods as the importer implementations, but with an extra parameter 'url'. This url parameter should contain the same identifier that is returned by the getImportSource() method of the importer implementation you want to use.

The other inherited methods are used to query the remote source.

Spring configuration for External Sources

In order to make the live import providers available, them must be mapped as spring beans into `dspace-api/src/main/resources/spring/spring-dspace-addon-import-services.xml.`

This is an example of a provider which allow to import both files and remote source.

```
<bean id="PubmedImportService"
      class="org.dspace.importer.external.pubmed.service.PubmedImportMetadataSourceServiceImpl" scope="singleton">
    <property name="metadataFieldMapping" ref="PubmedMetadataFieldMapping"/>
    <property name="supportedExtensions">
        <list>
            <value>xml</value>
        </list>
    </property>
    ...
</bean>
```

Here is defined the service responsible to fetch and transform the data `PubmedImportMetadataSourceServiceImpl`, which is an extension of `AbstractImportMetadataSourceService` as described above.

The field `metadataFieldMapping` is an instance of `Map<MetadataFieldConfig,MetadataContributor>` and contains the effective mapping.

`supportedExtensions` is the file extension this provider supports.

To expose this provider as Live Import provider, we need to construct a bean of type `org.dspace.external.provider.impl.LiveImportDataProvider` in the following way

```
<bean id="pubmedLiveImportDataProvider" class="org.dspace.external.provider.impl.LiveImportDataProvider">
    <property name="metadataSource" ref="PubmedImportService"/>
    <property name="sourceIdentifier" value="pubmed"/>
    <property name="recordIdMetadata" value="dc.identifier.other"/>
</bean>
```

where `metadataSource` is the bean referencing to live import service as described in “Metadata mapping”, `sourceIdentifier` the name of the provider in the live import framework and `recordIdMetadata` the metadatum used as id of the ImportRecord.

Metadata mapping

When using an implementation of `AbstractImportSourceService`, a mapping of remote record fields to DSpace metadata fields can be created.

first create an implementation of class `AbstractMetadataFieldMapping` with the same type set used for the importer implementation.

Then create a spring configuration file in `[dspace.dir]/config/spring/api`.

Each DSpace metadata field that will be used for the mapping must first be configured as a spring bean of class*org.dspace.importer.external.metadatamapping.MetadataFieldConfig*.

```
<bean id="dc.title" class="org.dspace.importer.external.metadatamapping.MetadataFieldConfig">
    <constructor-arg value="dc.title"/>
</bean>
```

NOTE: A large number of these MetadataFieldConfig definitions are already provided out-of-the-box in `[dspace.dir]/config/spring/api/dublincore-metadata-mapper.xml` This allows most service-specific Spring configurations to just reuse those existing MetadataFieldConfig definitions

Now this metadata field can be used to create a mapping. To add a mapping for the "dc.title" field declared above, a new spring bean configuration of a class class *org.dspace.importer.external.metadatamapping.contributor.MetadataContributor* needs to be added. This interface contains a type argument. The type needs to match the type used in the implementation of AbstractImportSourceService. The responsibility of each MetadataContributor implementation is to generate a set of metadata from the retrieved document. How it does that is completely opaque to the AbstractImportSourceService but it is assumed that only one entity (i.e. item) is fed to the metadatum contributor.

For example `java SimpleXpathMetadatumContributor implements MetadataContributor<OMElement>` can parse a fragment of xml and generate one or more metadata values.

This bean expects 2 property values:

- field: A reference to the configured spring bean of the DSpace metadata field. e.g. the "dc.title" bean declared above.
- query: The xpath expression used to select the record value returned by the remote source.

```
    <bean id="titleContrib" class="org.dspace.importer.external.metadatamapping.contributor.
SimpleXpathMetadatumContributor">
        <property name="field" ref="dc.title"/>
        <property name="query" value="dc:title"/>
    </bean>
```

Multiple record fields can also be combined into one value. To implement a combined mapping first create a *SimpleXpathMetadatumContributor* as explained above for each part of the field.

```
    <bean id="lastNameContrib" class="org.dspace.importer.external.metadatamapping.contributor.
SimpleXpathMetadatumContributor">
        <property name="field" ref="dc.contributor.author"/>
        <property name="query" value="x:authors/x:author/x:surname"/>
    </bean>
    <bean id="firstNameContrib" class="org.dspace.importer.external.metadatamapping.contributor.
SimpleXpathMetadatumContributor">
        <property name="field" ref="dc.contributor.author"/>
        <property name="query" value="x:authors/x:author/x:given-name"/>
    </bean>
```

Note that namespace prefixes used in the xpath queries are configured in bean "FullprefixMapping" in the same spring file.

```
    <util:map id="FullprefixMapping" key-type="java.lang.String" value-type="java.lang.String">
        <description>Defines the namespace mappin for the SimpleXpathMetadatum contributors</description>
        <entry key="http://purl.org/dc/elements/1.1/" value="dc"/>
        <entry key="http://www.w3.org/2005/Atom" value="x"/>
    </util:map>
```

Then create a new list in the spring configuration containing references to all *SimpleXpathMetadatumContributor* beans that need to be combined.

```
   <util:list id="combinedauthorList" value-type="org.dspace.importer.external.metadatamapping.contributor.
MetadataContributor" list-class="java.util.LinkedList">

        <ref bean="lastNameContrib"/>
        <ref bean="firstNameContrib"/>
    </util:list>
```

Finally create a spring bean configuration of class*org.dspace.importer.external.metadatamapping.contributor.CombinedMetadatumContributor*. This bean expects 3 values:

- field: A reference to the configured spring bean of the DSpace metadata field. e.g. the "dc.title" bean declared above.
- metadatumContributors: A reference to the list containing all the single record field mappings that need to be combined.
- separator: These characters will be added between each record field value when they are combined into one field.

```
    <bean id="authorContrib" class="org.dspace.importer.external.metadatamapping.contributor.
CombinedMetadatumContributor">
        <property name="separator" value=", "/>
        <property name="metadatumContributors" ref="combinedauthorList"/>
        <property name="field" ref="dc.contributor.author"/>
    </bean>
```

Each contributor must also be added to the "MetadataFieldMap" used by the *MetadataFieldMapping* implementation. Each entry of this map maps a metadata field bean to a contributor. For the contributors created above this results in the following configuration:

```
    <util:map id="org.dspace.importer.external.metadatamapping.MetadataFieldConfig"
              value-type="org.dspace.importer.external.metadatamapping.contributor.MetadataContributor">
        <entry key-ref="dc.title" value-ref="titleContrib"/>
        <entry key-ref="dc.contributor.author" value-ref="authorContrib"/>
    </util:map>
```

Note that the single field mappings used for the combined author mapping are not added to this list.

Available Metadata Contributor classes

Class	Description

SimpleXpathMetadatumContributor	Use an XPath expression to map the XPath result to a metadatum
SimpleMetadataContributor	This contributor is used in plain metadata as exposed above. Mapping is easy because it is based on the key used in the DTO.
CombinedMetadatumContributor	Use a LinkedList of MetadataContributor to combine into the value the resulting value for each contributor.

Framework Sources Implementations

PubMed Integration

Introduction

First read the base documentation on external importing (see above). This documentation explains the implementation of the importer framework using PubMed (http://www.ncbi.nlm.nih.gov/pubmed) as an example.

Publication Lookup URL

To be able to do the lookup for our configured import-service, we need to be able to know what URL to use to check for publications. This URL the `publication-lookup.url` setting defined within the `[dspace.dir]/config/modules/publication-lookup.cfg`. You may choose to modify this setting or override it within your local.cfg.

This setting can be modified in one of two ways:

- You can choose to specific a single, specific URL. This will tell the lookup service to only use one location to lookup publication information. Valid URLs are any that are defined as a `baseAddress` for beans within the `[src]/dspace-api/src/main/resources/spring/spring-dspace-addon-import-services.xml` Spring config file.
  - For example, this setting will ONLY use PubMed for lookups: `publication-lookup.url=http://eutils.ncbi.nlm.nih.gov/entrez/eutils/`
- By default, `publication-lookup.url` is set to an asterisk ('*'). This default value will attempt to lookup the publication using ALL configured importServices in the `[src]/dspace-api/src/main/resources/spring/spring-dspace-addon-import-services.xml` Spring config file

PubMed Metadata Mapping

The PubMed metadata mappings are defined in the `[dspace.dir]/config/spring/api/pubmed-integration.xml` Spring configuration file. These metadata mappings can be tweaked as desired. The format of this file is described in the "Metadata mapping" section above

PubMed specific classes Config

These classes are simply implementations based of the base classes defined in importer/external. They add characteristic behavior for services/mapping for the PubMed specific data.

Metadata mapping classes

- "PubmedFieldMapping". An implementation of AbstractMetadataFieldMapping, linking to the bean that serves as the entry point of other metadata mapping
- "PubmedDateMetadatumContributor"/"PubmedLanguageMetadatumContributor". Pubmed specific implementations of the "MetadataContributor" interface

Service classes

- "GeneratePubmedQueryService". Generates the pubmed query which is used to retrieve the records. This is based on a given item.
- "PubmedImportMetadataSourceServiceImpl". Child class of "AbstractImportMetadataSourceService", retrieving the records from pubmed.

ArXiv Integration

ArXiv Metadata Mapping

The ArXiv metadata mappings are defined in the `[dspace.dir]/config/spring/api/arxiv-integration.xml` Spring configuration file. These metadata mappings can be tweaked as desired. The format of this file is described in the "Metadata mapping" section above