2016 Framework for live import from external sources

General Framework

Introduction

This documentation explains the features and the usage of the importer framework.
Enabling the framework can be achieved by uncommenting the following step in `item-submission.xml`.
Implementation specific or additional configuration can be found in their related documentation, if any.
Please refer to subdivisions of this documentation for specific implementations of the framework.

Enabling framework

```
<step>
   <heading>submit.progressbar.lookup</heading>
   <processing-class>org.dspace.submit.step.XMLUIStartSubmissionLookupStep</processing-class>
   <xmlui-binding>org.dspace.app.xmlui.aspect.submission.submit.StartSubmissionLookupStep</xmlui-binding>
   <workflow-editable>true</workflow-editable>
</step>
```

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 Implementation of an import source for more information and how to enable the framework.

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.

Relation with BTE

While there is some overlap between this framework and BTE, this framework supports some features that are hard to implement using the BTE. It has explicit support to deal with network failure and throttling imposed by the data source. It also has explicit support for distinguishing between network caused errors and invalid requests to the source. Furthermore the framework doesn't impose any restrictions on the format in which the data is retrieved. It uses java generics to support different source record types. A reference implementation of using XML records is provided for which a set of metadata can be generated from any xpath expression (or composite of xpath expressions). Unless 'advanced' processing is necessary (e.g. lookup of authors in an LDAP directory) this metadata mapping can be simply configured using spring. No code changes necessary. A mixture of advanced and simple (xpath) mapping is also possible.

This design is also in line with the roadmap to create a Modular Framework as detailed in https://wiki.duraspace.org/display/DSPACE/Design+-+Module+Framework+and+Registry This modular design also allows it to be completely independent of the user interface layer, be it JSPUI, XMLUI, command line or the result of the new UI projects: https://wiki.duraspace.org/display/DSPACE/Design+-+Single+UI+Project

Implementation of an import source

Each importer implementation must at least implement interface *org.dspace.importer.external.service.components.MetadataSource* and implement the inherited methods.

One can also choose to implement class *org.dspace.importer.external.service.components.AbstractRemoteMetadataSource* next to the MetadataSource interface. This class contains functionality to handle request timeouts and to retry requests.

A third option is to implement class *org.dspace.importer.external.service.AbstractImportSourceService*. This class already implements both the MetadataSource interface and Source class. AbstractImportSourceService has a generic type set 'RecordType'. In the importer implementation this type set should be the class of the records received from the remote source's response (e.g. when using axiom to get the records from the remote source's XML response, the importer implementation's type set is*org.apache.axiom.om.OMElement*).

Implementing the AbstractImportSourceService allows the importer implementation to use the framework's build-in support to transform a record received from the remote source to an object of class **org.dspace.importer.external.datamodel.ImportRecord* containing DSpace metadata fields, as explained here: 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.

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>
```

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.

Framework Sources Implementations

PubMed Integration

Introduction

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

Enabling PubMed Lookup (XMLUI Only)

The PubMed specific integration of the external sources import requires the following to be active.
The PubMed lookup is done during the "XMLUIStartSubmissionLookupStep" and this can be enabled by adjusting one step in the `[dspace.dir]/config/item-submission.xml`. Uncommenting this step will permit the user to do the PubMed based lookups during their submission.

item-submission.xml

```
<!-- Find publications based on ID/DOI/Title/Author to pre-fill the submission. XMLUI ONLY.
     For JSPUI version, see JSPUIStartSubmissionLookupStep under <step-definitions> above.
<step>
    <heading>submit.progressbar.lookup</heading>
    <processing-class>org.dspace.submit.step.XMLUIStartSubmissionLookupStep</processing-class>
    <xmlui-binding>org.dspace.app.xmlui.aspect.submission.submit.StartSubmissionLookupStep</xmlui-binding>
    <workflow-editable>true</workflow-editable>
</step>
 -->
```

After uncommenting hat step, simply restart your servlet container, and this lookup step will be available within your deposit process.

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.