Importing Items via basic bibliographic formats (Endnote, BibTex, RIS, TSV, CSV)

This functionality is an extension of that provided by Importing and Exporting Items via Simple Archive Format so please read that section before continuing. It is underpinned by the Biblio Transformation Engine (https://github.com/EKT/Biblio-Transformation-Engine)

About the Biblio-Transformation-Engine (BTE)

The BTE is a Java framework developed by the Hellenic National Documentation Centre (EKT, www.ekt.gr) and consists of programmatic APIs for filtering and modifying records that are retrieved from various types of data sources (eg. databases, files, legacy data sources) as well as for outputing them in appropriate standards formats (eg. database files, txt, xml, Excel). The framework includes independent abstract modules that are executed seperately, offering in many cases alternative choices to the user depending of the input data set, the transformation workflow that needs to be executed and the output format that needs to be generated.

The basic idea behind the BTE is a standard workflow that consists of three steps, the data loading step, the processing step (record filtering and modification) and the output generation. The data loader provides the system with a set of Records (a list of key/value pairs - value is actually a list of Java Objects), the processing steps is responsible for filtering or modifying these records and the output generator outputs them in the appropriate format.

The standard BTE version comes with predefined Data Loaders as well as Output Generators for basic bibliographic formats. However, Spring Dependency Injection can be utilized in order to load custom data loaders, filters, modifiers and output generators.

BTE in DSpace

The functionality of batch importing items in DSpace using the BTE has been incorporated in the "import" script already used in DSpace for years.

In the import script, there is a new option (option "-b") to import using the BTE and an option -i to declare the type of the input format. All the other options are the same apart from option "-s" that in this case points to a file (and not a directory as it used to) that is the file of the input data.

Thus, to import metadata from the various input format use the following commands:

Input	Command
BibTex	`[dspace]/bin/dspace import -b -m mapFile -e example@email.com -c 123456789/1 -s /export/export-bibtex -i bibtex`
CSV	`[dspace]/bin/dspace import -b -m mapFile -e example@email.com -c 123456789/1 -s /export/export-csv -i csv`
TSV	`[dspace]/bin/dspace import -b -m mapFile -e example@email.com -c 123456789/1 -s /export/export-tsv -i tsv`
RIS	`[dspace]/bin/dspace import -b -m mapFile -e example@email.com -c 123456789/1 -s /export/export-ris -i ris`
EndNote	`[dspace]/bin/dspace import -b -m mapFile -e example@email.com -c 123456789/1 -s /export/export-endnote -i endnote`

Keep in mind that the value of option "-e" must be a valid email of a DSpace user and value of option "-c" must be the collection handle the items will be imported to.

BTE Configuration

Since DSpace administrators may have incorporated their own metadata scheme within DSpace (apart from default Dublin Core), someone may need to configure BTE to match their custom schemes.

BTE configuration file is located in path: `[dspace]/config/spring/api/bte.xml` and it's a Spring XML configuration file that consists of beans. (If these terms are unknown to you, please refer to Spring Dependency Injection web site for more information)

Explanation of beans:

```
bean id="gr.ekt.transformationengine.core.TransformationEngine"
```

This bean is instantiated when the import takes place. It deploys a new BTE transformation engine that will do the transformation from one format to the other. It needs two input arguments, the workflow (the processing step mentioned before) and the output generator.

```
bean id="org.dspace.app.itemimport.DataLoaderService"
```

Within this bean we declare all the possible data loaders that we need to support. Keep in mind that for each data loader we specify a key value that can be used as the value of option "-i" in the import script that we mentioned earlier.

```
bean id="conjunctionTransformationWorkflow"
```

This bean describes the processing steps. Currently there are no processing steps meaning that all records loaded by the data loader will pass to the output generator, unfiltered and unmodified. (See next section "Case studies" for info about how to add a filter or a modifier)

```
bean id="outputGenerator"
```

This bean declares the output generator, that in this case the default DSpace output generator is used. As anyone can see, the input properties for this bean is the configuration of metadata schemas that are used in the DSpace repository. This is the place where someone can configure how the input data map to the metadata schema of DSpace. The references maps are actually key/value pairs where the key is the record key mentioned earlier and the value is the metadata element (schema, element, qualifier) that this key will map to.
By default, the data loaders provided with the BTE use as record keys the values that are specified in the input data file. For example, in the case of a bibtex input:

```
================
@article{small,
author = {Freely, I.P.},
title = {A small paper},
journal = {The journal of small papers},
year = 1997,
volume = {-1},
note = {to appear},
}
================
```

the record keys will be "author", "title", "journal", "year" and so on.

In the case of a RIS format input file:

```
================
TY  - JOUR
AU  - Shannon,Claude E.
PY  - 1948/07//
TI  - A Mathematical Theory of Communication
JO  - Bell System Technical Journal
SP  - 379
EP  - 423
VL  - 27
ER  -
=================
```

the record keys will be "TY", "AU" and so on.

Thus, depending on the value of "-i" option in import script, the user needs to change the configuration of this bean accordingly.

Case Studies

1) I have my data in a format different from the ones that are supported by this functionality. What can I do?

In this case, you will need to create a new data loader. To do this, just create a new Java class by extending the abstract class

```
gr.ekt.transformationengine.core.DataLoader
```

of BTE. You will need to implement the method

```
public abstract RecordSet loadData();
```

in which you have to create records from the input file - most propably you will need to create your own Record class (by extending gr.ekt.transformationengine.core.Record class) and fill a RecordSet.

After that, you will need to declare the new DataLoader in the Spring XML configuration file (in the bean with id: "org.dspace.app.itemimport.DataLoaderService") using your own key. Use this key as a value for option "-i" in the import key so as to specify that your data loader must run.

2) I need to filter some of the input records or modify some value from records before outputting them

In this case you will need to create your own filters and modifiers.

To create a new filter, you need to extend BTE abstact class:

```
gr.ekt.transformationengine.core.Filter
```

You will need to implement the method:

```
public abstract void filter(Record record)
```

Return true if the specified record needs to be filtered, otherwise return false.

To create a new modifier, you need to extend BTE abstact class:

```
gr.ekt.transformationengine.core.Modifier
```

You will need to implement the method

```
public abstract void modify(Record record)
```

within you can make any changes you like in the record.

After you create your own filters or modifiers you need to add them in the Spring XML configuration file as in the following example:

```
<bean id="customfilter" class="org.mypackage.MyFilter"/>

<bean id="conjunctionTransformationWorkflow" class="gr.ekt.transformationengine.core.ConjunctionTransformat
ionWorkflow">
 <property name="steps">
 <list>
    <ref bean="customfilter"/>
 </list>
 </property>
</bean>
```

You can add in the conjunctionTransformationWorkflow as many filters and modifiers you like, the will run the one after the other in the specified order.