EPrints data to Foxml

1 About this document

Author

Bron Dye, RUBRIC Technical Officer

Tim McCallum, RUBRIC Technical Officer

Purpose

This technical report outlines how Eprints export data can be used for ingest as foxml objects.

Audience

RUBRIC Project Partners and other users of Fedora repositories

Requirements

Access to ePrints metadata output

Python 2.4 installed on the system to run the harvest

The libxml2 library, including the Python bindings, installed on the system to run the harvest

An instance of VITAL2.1 for ingest

References

Eprints website

http://www.eprints.org/

VITAL website at VTLS
http://www.vtls.com/Products/vital.shtml

Documentation on the FOXML (Fedora Object XML) specification
http://www.fedora.info/download/2.1.1/userdocs/digitalobjects/introFOXML.html

Python website
http://www.python.org/

Libxml2 website
http://xmlsoft.org/python.html

Py.test tool and library website
http://codespeak.net/py/current/doc/test.html

Subversion website
http://subversion.tigris.org/

Notes

The scripts have been developed on a Linux based system. Python is a cross platform programming language and therefore the scripts should also run under Microsoft Windows, and the OSX operating systems. This has not been tested.

The installation of the Python programming language and the libxml2 library, including Python bindings is outside the scope of this technical report. Many Linux distributions, such as Ubuntu, will have these already installed.

2 Background Information

A component of the work undertaken at RUBRIC-Central is the development of various data migration strategies. These strategies are designed to assist RUBRIC Project Partners to migrate data into, and out of, various systems. The data migrations specifically target the three institutional repository solutions under consideration as part of the project.

Interest was expressed in being able to migrate an Eprints export file produced from established library systems into other repositories, such as VITAL or DSpace. This technical report, and the associated Python scripts, comprise the strategy to achieve this goal.

The Python scripts create an archive or directory, similar in structure to a DSpace Archive. Within this directory are created item directories each with a temporary xml file storing dublin core metadata, all files relevant to the xml item and a file listing all relevant files attached to the EPrints item. This structure forms the basis of further migration and can then be used for various other repositories.

The Python scripts have been developed using a unit testing approach using the testing framework provided by the py.test tool and library. More information about the library is available at the website listed in the references section of this technical report. These scripts have been developed using Python version 2.4, and may work with earlier versions. However this has not been tested.

The Python scripts are modular in nature and use functionality provided by modules that have been used in other migration strategies. It is anticipated that this type of architecture will allow modification and customisation as required.

3 The Python Scripts

The data migration work is carried out by a script written in the Python programming language, for more information about Python see the official Python website listed in the references section of this technical report. The following files make up the scripts used in the data migration.

3.1 Script list

split_xml_into_archive.py

> The python script used to split a large xml file into items.

xsl_transform.py

> A Python module that converts temporary xml file into dublin core xml or marc xml files.

clean_temp_files.py

> A Python script that replaces incompatible unicode characters in the temp.xml.

dspace_archive.py

> A Python module that provides a utility class for the creation of dspace archive objects.

archive_to_foxml.py

> A Python module that provides creates a foxml object by adding a prefix to the title, merges all datastreams and exports the foxml objects into a single output directory.

obtain_eprints_files_for_archive.py

> A Python module that loops through the archive, reading the temp.xml file. When a link to an external file is found the script fetches the file and loads it into the archive.

pdf_to_full_text.py

> A python script that converts pdf files found in the archive into full text files

3.2 Downloading the Python Scripts

All of the data migration scripts, and associated code libraries, modules and files, are made available via a publicly accessibly website. If you have the subversion client installed you can download the Python scripts, test files, and other files used during development.

https://rubric-central.usq.edu.au/svn/Public/migration_toolkit/

4   Preparing the EPrints data

Before starting the migration, remove all namespace declarations from the export file, then wrap the entire Eprints export data in collection tags.

```
Eg:    <collection>

       <record>

       ....data ....

       </record>

       </collection>
```

It is assumed that you have Python and the Libxml2 library, including the Python bindings, already installed.

4.1   Split Eprints Export file

To split the Eprints export xml file into individual items, the Python script **split_xml_into_archive.py** is used.

This script will create the following archive in the current directory, where each item is represented by one item directory. These files are numbered consecutively; first directory is called **00**, **01** and so on.

```
EprintsArchive/

     0/

           contents

           temp.xml

     1/

           contents

           temp.xml
```

Note:

Change directory to **migration_toolkit**

Script structure:

python split_xml_into_archive.py [dataFileName] [archiveName] [xpath] [streamName][templateFile]

Argument definitions:

● [dataFileName] path and filename of the Eprints export.

- [archiveName] name of the Archive to be created

- [xpath] match the node for each item in the larger xml file (eg //record) If there is a namespace for example marc:record use the following

  "//*[local-name()='record']"

- [streamName] filename of the smaller output xml file to be created

- [templateFile] (opt) filepath to a wrapping file if a namespace is required. If there is NO namespace requirement, use "False" If unsure set this to false as there is another opportunity to wrap the split files with metadata later in the migration.

Example:

```
python split_xml_into_archive.py eprints_archive_export.xml
EprintsArchive //record temp.xml False
```

4.2 Clean temp.xml

The **clean_temp_files.py** script correct unicode characters in the temp.xml, that can interfere with the migration process.

Note:

Change directory to **migration_toolkit**

Script structure:

python clean_temp_files.py [archiveName][tempFileName]

Argument definitions:

Replace the following parameters:

- [archiveName] the name of the archive
- [tempFileName] the file to be cleaned

Example:

```
python clean_temp_files.py EprintsArchive temp.xml
```

5 Converting Metadata

5.1 Convert temp.xml file to marc.xml

The **xsl_transform.py** script is the Python script that converts the marc metadata into a dublin core stream ready for ingest into VITAL.

Note:

Change directory to **migration_toolkit**

Script structure:

python xsl_transform.py [InputFile][XslFilePath][OutputFile][ArchiveName] [RemoveInputFile]

Argument definitions:

- [InputFile] the filename of the input xml file to be converted, found within the item directory of the archive.
- [XslFilePath] the file path to the stylesheet to be used for the conversion
- [OutputFile] the filename to be used following the conversion, this will be found in the item directory within the archive
- [ArchiveName] the name of the archive to be accessed
- [RemoveInputFile] Remove the input file? - set to False as it will be used later

Example:

```
python  xsl_transform.py temp.xml

eprints/xsl/eprints_xml_2_marc.xsl marc.xml EprintsArchive
False
```

5.2 Add Controlfield tag

The basic metadata does not have a MARC controlfield tag that is applicable to the contents of each record. This needs to be created during the xsl transformation.
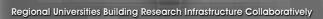
Script structure:

python create_marc_controlfieldtag_008.py [ArchiveName]

Argument definitions:

- [ArchiveName] tthe name of the archive to be accessed.

Example:

```
python create_marc_controlfieldtag_008.py EprintsArchive
```

5.3 Convert Marc datastream to MODS

The **xsl_transform.py** script is the Python script that converts the marc metadata into a MODS datastream ready for ingest into VITAL.

Script structure:

python xsl_transform.py [InputFile][XslFilePath][OutputFile][ArchiveName] [RemoveInputFile]

Argument definitions:

- [InputFile] the filename of the input xml file to be converted, this file is the file that has just been created, found within the item directory of the archive.
- [XslFilePath] the file path to the stylesheet to be used for the conversion
- [OutputFile] the filename to be used following the conversion, this will be found in the item directory within the archive (dublin_core is hard coded into the code that creates the foxml objects so this name will have to be used)
- [ArchiveName] the name of the archive to be accessed
- [RemoveInputFile] Remove the input file? - set to False as it will be used later

Example:

```
python  xsl_transform.py marc.xml

xsl/marc_to_mods.xsl mods.xml voyagerArchive False
```

5.4 Convert marc.xml to Dublin Core

The **xsl_transform.py** script is the Python script that converts the marc metadata into dublin core.

Note:

Change directory to **migration_toolkit**

Script structure:

python xsl_transform.py [InputFile][XslFilePath][OutputFile][ArchiveName] [RemoveInputFile]

Argument definitions:

- [InputFile] input xml file to be converted, this file is found within the item directory of the archive.
- [XslFilePath] the file path to the stylesheet to be used for the conversion
- [OutputFile] the filename to be used following the conversion, this will be found in the item directory within the archive (dublin_core.xml is hard coded into the code that creates the foxml objects so is used as defaullt at this point)

- [ArchiveName] the name of the archive to be accessed
- [RemoveInputFile] Remove the input file? - set to False as it will be used later

Example:

```
python  xsl_transform.py

marc.xml xsl/marc_to_oai_dc_minus_namespace.xsl

dublin_core.xml    EprintsArchive False
```

6 Files associated with Eprints Items

6.1 Obtain attached files for archive

If the data contains pdfs, or other files, to be harvested then carry out the following instructions otherwise proceed to the next step

The obtain_eprints_files_for_archive.py script is a Python script that searches the temp.xml files looking for URL links to external pdf files. Once found the files are then downloaded by the script and put into the archive.

Note:

Change Directory to **eprints/python**

Script structure:

python obtain_eprints_files_for_archive.py [archiveName][targetFileName]

Argument definitions

- [archiveName] the name of the archive

- [targetFileName] the name of the file in each archive to be accessed

```
python obtain_eprints_files_for_archive.py    EprintsArchive
temp.xml
```

6.2 Create a full text version of pdf files

The **pdf_to_full_text.py** script is the Python script that will converts any harvested pdf files to **fulltext**. Replace [ArchiveName] with the name of the archive to be accessed

Note:

Change directory to **migration_toolkit**

Script structure:

pdf_to_full_text.py [ArchiveName]

Argument definitions:

- [ArchiveName] the name of the archive to be accessed

Example:

```
python pdf_to_full_text.py EprintsArchive
```

Once this script has run, an additional file will be found in the item directory called **fulltext** this will contain the fulltext of ALL the pdf files belonging to the item.

```
EPrintsArchive/
```

```
0/

        01front.pdf

        02whole.pdf

        contents

        dublin_core.xml

        marc.xml

        fulltext
```

7 Ingest Preparation

7.1 Create foxml objects

The **archive_to_foxml.py** script is the Python script that will create a directory of foxml objects.

Script structure:

python archive_to_foxml.py [archiveName] [startNum][PIDPrefix]

[outputDirectory][labelPrefix][foxmlObjectState][MARCFileName]

[MARCDataStreamState][DCFileName]
[DCDataStreamState][MODSFileName][MODSObjectState]
[URLforNonXmlDataStreams]

Argument definitions:

- [archiveName] the full path to name of the archive to be accessed
- [startNum] the starting number for the PID increment
- [PIDPrefix] the name of the PID
- [outputDirectory] the name of the directory for storing the foxml objects
- [labelPrefix] Prefix to be added to title for reference. Eg Imported Item:
- [foxmlObjectState] set this to Active (A), Inactive(I) or Deleted (D).
- [MARCFileName] the name of the MARC xml file contained in the Archive.
- [MARCDataStreamState] set this to Active (A), Inactive(I) or Deleted (D).
- [DCFileName] the name of the Dublin Core file contained in the Archive
- [DCDataStreamState] set this to Active (A), Inactive(I) or Deleted (D).
- [MODSFileName] the name of the MODS file contained in the Archive
- [MODSDataStreamState] set this to Active (A), Inactive(I) or Deleted (D).
- [URLforNonXmlDataStreams]

  If non xml data streams exist in the archive (PDF or full text), a URL is required to access them. If you use python simple server use **http://localhost:8000** or if you are using an existing server enter a URL path to the existing server where the non xml data streams can be made available during ingest.

  If no pdf files or fulltext datastreams exist, set to FALSE

  Note: Remove any trailing slashes (created by tab completion) from arguments before executing script

Example:

```
python archive_to_foxml.py dspaceArchive 0 vital
```

```
exportedItems Imported_Items A marc.xml A dublin_core.xml A
mods.xml I

http://servername/directoryname
```

7.2 Insert namespace in to foxml objects

This script ensures that the correct marc namespace appears within the foxml object. Insertion is completed at this point because to do so earlier in the process would change formatting and mean that the marc viewed within VITAL repository was not validating correctly.

Script structure:

python insert_xmlns_xsi.py [outputDirectory]

Argument definitions:

- [outputDirectory] the name of the directory for storing the foxml objects

Example:

```
python insert_xmlns_xsi.py exportedItems
```

8 VITAL2.1 Ingest

To ingest the items into VITAL2.1 complete the following procedure:

1. Copy the FOXML object files onto the server that is running VITAL

2. Ensure that the FOXML object files are accessible via the **dbadmin** user

3. Change to the **dbadmin** user

4. Navigate to the following directory on the server

   ```
   /usr/vtls/vital/fedora/client/bin
   ```

5. Ensure the FEDORA_HOME and JAVA_HOME shell variables exist. If they do not exist, sample commands are outlined below

   ```
   export FEDORA_HOME=/usr/vtls/vital/fedora

   export JAVA_HOME=/usr/vtls/vital/java
   ```

6. Invoke the following command to start the fedora-ingest utility, where **[files]** is the location of the FOXML files and **[password]** is the fedoraAdmin password

   Note: This may take some time to complete

   ```
   ./fedora-ingest d [files] foxml1.0 O localhost:8080
   fedoraAdmin [password]
   ```

7. Further information on the Fedora ingest utilities is available at the following URL:

   http://www.fedora.info/download/2.1.1/userdocs/client/cmd-line/index.html#ingest

8. Once the ingest is complete, check the XML log file, as specified by the output of the program, for any errors

9. If the new objects are to made available via the VITAL portal, ensure sufficient time has elapsed to allow the VITAL indexer to become aware of the additional objects