Voyager XML data to Foxml

1 About this document

Author

Bron Dye, RUBRIC Technical Officer

Tim McCallum, RUBRIC Technical Officer

Purpose

This technical report outlines how to Voyager library system xml can be used for ingest as foxml objects.

Audience

RUBRIC Project Partners and other users of Fedora repositories

Requirements

xml import file

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 VITAL for ingest

A working installation of “marc-edit 5.0”

References

Official 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

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

Official libxml2 website
http://xmlsoft.org/python.html

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

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

Marc-edit site .exe file

http://oregonstate.edu/~reeset/marcedit/software/development/MarcEdit50_Setup.exe

RUBRIC is supported by the Systemic Infrastructure Initiative as part of the Commonwealth Government's Backing Australia's Ability - An Innovative Action Plan for the Future (http://backingaus.innovation.gov.au)

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.

RUBRIC is supported by the Systemic Infrastructure Initiative as part of the Commonwealth Government's Backing Australia's Ability - An Innovative Action Plan for the Future (http://backingaus.innovation.gov.au)

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 exported files produced from local 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 exported item.

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.

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_marc_metadata.py

A Python script that checks if date and title metadata in the input file have extra trailing characters. It removes these from 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_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 pdf and loads it into the archive.

pdf_to_full_text

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

wrap_namespace_around_file.py

This python script is used to convert the bare split record to marc.xml.

remove_xml_node.py

A python script that removes an entire xml node and its contents.

4 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 at the following URL.

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

5 Preparing the Voyager data for ingest

The following sections of this technical report outline the procedure converting the Voyager output file to marcxml and then using the Python scripts to extract the xml objects from a larger xml file.

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

5.1 Creating the marcxml

1. Download and open marc edit.
2. Select marc maker
3. Set the input file as the Voyager file
4. Set the output file
5. Click on marc to marc21 in the marc functions
6. Click execute to create the MarcXML
7. Open MarcXML in a text editor.
8. Remove the marc namespaces from it, Replace <marc: with < and replace </marc: with </

5.2 Extracting the XML items from the larger XML file

To split the large xml file into individual items, the Python script **split_xml_into_archive.py** is used. This script creates a temporary DSpace archive, in the current directory. Replace the parameters as follows:

Script structure:

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

Argument definitions:

- [dataFileName] filename of the large input xml file created using marcedit
- [archiveName] name of the Archive that will 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 output xml file that will be created
- [templateFile] (opt) the filepath to a wrapping file if a namespace is required. If there is NO namespace requirement, use "False" If unsure set this to "False"

Example:

```
python split_xml_into_archive.py temp_data/ethesis.xml
voyagerArchive //record temp.xml False
```

This script will create the following simple Dspace archive, where each item is represented by one item directory. These files are numbered consecutively; first directory is called **0**, **1** and so on.

```
dspaceArchive/
      0/
            contents
            temp.xml
      1/
            contents
            temp.xml
```

5.3 Clean temp.xml

The **clean_marc_metadata.py** script is used to check if date and title metadata have extra trailing characters. For example: A date field may have a trailing "." or "]". A title field may have a trailing ":". The script removes the extra characters and updates the temp.xml.

Script structure:

python clean_marc_metadata.py [archiveName][tempFileName][recordType]

Argument definitions:

Replace the following parameters:

- [archiveName] name of the archive (eg voyagerArchive)
- [tempFileName] filename of the temp.xml file that will be cleaned
- [recordType] the type of record Ethesis(use E) or Brunner (use B) Library Publications (use L) Working Papers (use W)

Example:

```
python clean_marc_metadata.py voyagerArchive temp.xml E
```

5.4 Obtain pdf files for archive

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

The obtain_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 fiiles are then downloaded by the script and put into the archive. Use optional protocol, username and password arguments when files to be harvested require authentication.

Script structure:

python obtain_files_for_archive.py [archiveName][targetFileName][fileType][protocol][username][password]

Argument definitions

- [archiveName] name of the archive to be accessed
- [targetFileName] name of the file in each archive to be accessed
- [fileType] file extension of external file to be accessed and downloaded
- [protocol] protocol used to access file(optional)
- [username] username required to access file(optional)
- [password] password required to access file(optional)

```
python obtain_files_for_archive.py  voyagerArchive temp.xml pdf
http:// username password
```

5.5 Convert temp.xml into marc.xml

The **wrap_namespace_around_file.py** script will be used to convert the bare split record to marc.xml.

Script structure:

python wrap_namespace_around_file.py [fileName][wrapperFile][archiveName] [newFile]

Argument definitions:

- [fileName] name of the file inside the archive that needs wrapping
- [wrapperFile] the full path to the wrapper file (eg wrapper_file_for_marc_file)
- [archiveName] name of the archive (eg voyagerArchive)
- [newFile] the name of the output file ie: marc.xml

Example:

```
python wrap_namespace_around_file.py temp.xml
wrapper_file_for_marc_file voyagerArchive marc.xml
```

5.6 Create a full text version of the pdf files

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

Script structure:

python pdf_to_full_text.py [ArchiveName]

Argument definitions:

- [ArchiveName] name of the archive (eg voyagerArchive)

Example:

```
python pdf_to_full_text.py [ArchiveName]

eg python pdf_to_full_text.py voyagerArchive
```

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.

```
dspaceArchive/

      0/

            01front.pdf

            02whole.pdf

            contents

            dc.xml

            marc.xml

            fulltext
```

5.7 Remove 856 marc tags from marc.xml (optional)

The **remove_xml_node.py** script is the Python script that removes unwanted nodes, in this case from the marc.xml.

Script structure:

python remove_xml_node.py [archiveName] [fileName][xpath]

Argument definitions:

- [archiveName] name of the archive to be accessed
- [fileName] name of the file from which the node is removed, in this case marc.xml

- [xpath] the xpath to locate the unwanted node

Example:

```
python remove_xml_node.py dspaceArchive marc.xml //*[local-
name()='datafield'][@tag='856']
```

5.8 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
- [XslFilePath] file path to the stylesheet used for the conversion
- [OutputFile] filename to be created by conversion – mods.xml, in this case
- [ArchiveName] name of the archive to be accessed
- [RemoveInputFile] Remove the input file? - set to False

Example:

```
python  xsl_transform.py marc.xml

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

5.9 Convert split record (marc.xml) to Dublin Core stream

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

Script structure:

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

Argument definitions:

- [InputFile] filename of the input xml file to be converted.
- [XslFilePath] file path to the stylesheet used for the conversion
- [OutputFile] filename created by conversion – dublin_core.xml in this case
- [ArchiveName] name of the archive to be accessed

- [RemoveInputFile] Remove the input file? - set to False

Example:

```
python  xsl_transform.py marc.xml

xsl/marc_dc_voyager.xsl dublin_core.xml voyagerArchive False
```

6 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] full path to name of the archive to be accessed
- [startNum] starting number for the PID increment
- [PIDPrefix] name of the PID
- [outputDirectory] 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] name of the MARC xml file contained in the Archive.
- [MARCDataStreamState] set this to Active (A), Inactive(I) or Deleted (D).
- [DCFileName] name of the Dublin Core file contained in the Archive
- [DCDataStreamState] set this to Active (A), Inactive(I) or Deleted (D).
- [MODSFileName] 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 this toFALSE

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

Example:

```
python archive_to_foxml.py voyagerArchive 0 vital

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

http://servername/directoryname
```

6.1 Insert name space 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 voyagerItems
```

7 Ingesting the Items into VITAL

7.1 Making the Datastreams Available via a Web Server

By design the Fedora ingest applications expect to retrieve non XML datastreams via HTTP from a web server. XML datastreams can be incorporated into the XML that represents the FOXML object. It is anticipated that in a future release of the Fedora software it will be possible to encode binary data, such as PDF files, and include them in the FOXML object.

There are two mechanisms available to make the datastreams available via a web server. The first is to the use the basic HTTP server that comes with Python. The other is to use an existing web server.

7.1.1 Using the Python Web Server

To use the simple HTTP server that comes with Python follow these steps:

1. Log into the machine that VITAL is running on
2. Start a new terminal session (or place the & after the command to make the server run in the background)
3. Copy the contents of **[archiveName]** directory (created in archive to foxml procedure above) into the **[outputDirectory]** (named in archive to foxml procedure above) If these directories are currently on another machine carrying out the harvest, ie development machine then carry out this step on the other machine and secure copy the [**outputDirectory]** over in one step. A good place copy it to is the /usr/vtls/vital/fedora/client/bin directory on the machine running VITAL
4. Change to the directory one level above the **[outputDirectory]** **(**/usr/vtls/vital/fedora/client/bin)
5. Execute the following command

```
python -c "import SimpleHTTPServer;SimpleHTTPServer.test()"&
```

This command will invoke Python and start the SimpleHTTPServer. The server will be able to provide access to files from the current directory, and any sub directories. The port used by the web server is 8000.

Please note that the SimpleHTTPServer will not be able to service requests other than those from the local machine, and therefore this process will only work when the datastreams are on the same server as the VITAL repository.

7.1.2 Using an Existing Web Server

Using an existing web server is possible. The **archive_to_foxml.py** script contains the http://localhost:8000 default server, simply change this path to the location on the server where you will be uploading the datastream files. The Python script will append the directory and file names to this path.

When uploading the datastream files onto the server ensure the directory structure is preserved, including the names of each file.

RUBRIC is supported by the Systemic Infrastructure Initiative as part of the Commonwealth Government's Backing Australia's Ability - An Innovative Action Plan for the Future (http://backingaus.innovation.gov.au)

8 Ingest procedure

To ingest the items into VITAL complete the following procedure:

1. Copy the **[outputDirectory]** onto the server that is running VITAL
2. Ensure that the contents of the **[outputDirectory]** 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 **[outputDirectory]**is the location of the FOXML files and **[password]** is the fedoraAdmin password

   Note: This may take some time to complete

```
./fedora-ingest d [outputDirectory] 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