Harvesting ADT data for Foxml objects

1 About this document

Author

Bron Dye, RUBRIC Technical Officer

Tim McCallum, RUBRIC Technical Officer

Purpose

This technical report outlines how to use the ADT Harvest scripts to harvest items from an ADT repository for import to various repositories.

Audience

RUBRIC Project Partners and other users of ADT

Requirements

Access to an ADT website

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

Ensure pdf2fulltext module is installed

An instance of VITAL for ingest

References

Official ADT website
http://adt.caul.edu.au/

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/

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 ADT harvesting 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 items from an Australian Digital Thesis (ADT) repository 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 ADT item directories each with a temporary xml file storing dublin core metadata, all files relevant to the ADT item and a file listing all relevant files attached to the ADT 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.

get_adt_html.py

The python script used to harvest items out of an ADT repository.

xsl_transform.py

A Python module that converts ADT metadata to dspace metadata and creates a dublin core .

dspace_archive.py

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

pdf_to_full_text.py

A Python module that converts all pdf files within each item directory to a single **fulltext** file within the individual item directory.

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.

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)

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

4.1 Download the Python Scripts via Subversion

If you have the subversion client installed you can download the Python scripts, test files, and other files used during development. The URL that you will need to check out is as follows:

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

5 How to Harvest the ADT Data

The following sections of this technical report outline the procedure for using the Python scripts to harvest items from the ADT repository.

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

5.1 Harvesting the Items in the ADT Repository

The **get_adt_html.py** script is the Python script that harvests all of the items in an ADT repository. The script captures the meta data between the head tags of the individual ADT items and converts the tags to lowercase. To invoke the script enter the following command in a terminal:

```
python get_adt_html.py [ADT URL]

eg:python get_adt_html.py http://adt.usq.edu.au/adt-QUSQ/public/index.html
```

This script creates a temporary DSpace archive, **dspaceArchive**, in the current directory. Any files the script is unable to locate, will not be created. The user is notified if the files are missing.

```
dspaceArchive/
      00/
            01front.pdf
            02whole.pdf
            contents
            dc_temp.xml

      01/
            01front.pdf
            02whole.pdf
            03appendix.pdf
            contents
            dc_temp.xml
```

Each item is represented by one item directory within **dspaceArchive**. These files are

numbered consecutively; first directory is called **00**, **01** and so on.

The datastreams for each object are stored in the corresponding numbered directory. For example the datastreams in the **00** directory contain three datastreams for this object. They are the two PDF files, **01front.pdf** and **02whole.pdf**, and a text file containing all of the text extracted from the two PDF files named **contents** and a temporary xml file, **dc_temp.xml**, containing basic dublin core metadata..

5.2 Convert basic metadata to Marc

The **xsl_transform.py** script is the Python script that will converts the harvested metadata.

Replace the following parameters:

- [InputFile] the filename of the input xml file to be converted, this 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
- [ArchiveName] the name of the archive to be accessed
- [RemoveInputFile] Remove or retain the input file? - True or False

```
python xsl_transform.py [InputFile][XslFilePath][OutputFile]

[ArchiveName][RemoveInputFile]

eg:python  xsl_transform.py dc_temp.xml
../xsl/adtHtml_to_marc.xsl

 marc.xml dspaceArchive True
```

This script converts the **dc_temp.xml** files in the individual Item directories of the **dspaceArchive** directory to VITAL compliant marc, **marc.xml**.

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 to Dublin Core stream

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

Note:

This script requires that the namespace of the marc xml be declared at the beginning of the file, not included in the individual tags inside the marc record.

Replace the following parameters:

- [InputFile] the filename of the 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
- [ArchiveName] the name of the archive to be accessed
- [RemoveInputFile] Remove or retain the input file? - True or False

```
python xsl_transform.py [InputFile][XslFilePath][OutputFile]

[ArchiveName][RemoveInputFile]

eg:python  xsl_transform.py marc.xml

../xsl/marc_to_oai_dc_minusnamespace.xsl dc.xml dspaceArchive
False
```

This script converts the **marc.xml** files in the individual item directories of the **dspaceArchive** directory to VITAL compliant dublin core, **dc.xml**.

5.5 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

```
python pdf_to_full_text.py [ArchiveName]

eg python pdf_to_full_text.py dspaceArchive
```

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/
      00/
            01front.pdf
            02whole.pdf
            contents
            dc.xml
            marc.xml
            fulltext
```

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] 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
```

6.1 Insert name space in to foxml objects

```
Python insert_xmlns_xsi.py archiveName
```

6.2 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.

6.2.1 Using the Python Web Server

To use the based 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.

6.2.2 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.

6.3 Ingesting the Items into VITAL

To ingest the items into VITAL complete the following procedure:

1. Ensure that the datastream files are downloadable, either via the Python SimpleHTTPServer or via an existing web server
2. Copy the FOXML object files onto the server that is running VITAL
3. Ensure that the FOXML object files are accessible via the **dbadmin** user
4. Change to the **dbadmin** user
5. Navigate to the following directory on the server

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

6. 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
```

7. 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  [outputDirectory](mentioned above)
foxml1.0 O localhost:8080 fedoraAdmin [password]



./fedora-ingest d dspaceArchive foxml1.0 O localhost:8080
fedoraAdmin fedoraAdmin
```

8. 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

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

10. 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