

ADT to VITAL

Table of Contents

1 Introduction.....	1
2 Background information.....	3
3 Downloading the Python Scripts.....	4
3.1 Script dependencies	4
4 Get ADT HTML using single URL.....	5
4.1 Name of the script.....	5
4.2 Location of script.....	5
4.3 Purpose of script.....	5
4.4 Parameters/Arguments.....	5
4.5 Syntax.....	5
4.6 Example.....	5
5 XSL Transform (temp.xml to MARC.xml).....	6
5.1 Name of the script.....	6
5.2 Location of script.....	6
5.3 Purpose of script.....	6
5.4 Parameters/Arguments.....	6
5.5 Syntax.....	6
5.6 Example.....	7
6 Create MARC controlfield tag 008.....	8
6.1 Name of the script.....	8
6.2 Location of script.....	8
6.3 Purpose of script.....	8
6.4 Parameters/Arguments.....	8
6.5 Syntax.....	8
6.6 Example.....	8
7 XSL Transform (marc.xml to dublin_core.xml).....	9
7.1 Name of the script.....	9
7.2 Location of script.....	9
7.3 Purpose of script.....	9
7.4 Parameters/Arguments.....	9
7.5 Syntax.....	9
7.6 Example.....	10
8 PDF to full text.....	11
8.1 Name of the script.....	11
8.2 Location of script.....	11
8.3 Purpose of script.....	11
8.4 Parameters/Arguments.....	11
8.5 Syntax.....	11
8.6 Example.....	11
9 Archive To FOXML.....	12
9.1 Name of the script.....	12
9.2 Location of script.....	12
9.3 Purpose of script.....	12
9.4 Additional Information.....	12
9.5 Parameters/Arguments.....	13
9.6 Syntax.....	13
9.7 Example using external web server to host non XML data streams.....	14
9.8 Example using Python simple server to host non XML data streams.....	14
10 Insert XMLNS XSI.....	15
10.1 Name of the script.....	15
10.2 Location of script.....	15
10.3 Purpose of script.....	15

10.4 Parameters/Arguments.....	15
10.5 Syntax.....	15
10.6 Example.....	15
11 Preparing FOXML data for Fedora Ingest.....	16
12 Preparing non XML data for Fedora ingest.....	16
12.1 Using the Python Web Server	16
12.2 Using an Existing Web Server.....	16
13 Fedora Ingest for VITAL 3.....	17
13.1 Ingest procedure.....	17

1 Introduction

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 VITAL/Fedora 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 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/~reese/marcedit/software/development/MarcEdit50_Setup.exe

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. However 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 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 Downloading the Python Scripts

All of the data migration scripts, and associated code libraries, modules and files, are made available via a publicly accessible website.

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/code/migration_toolkit/

3.1 Script dependencies

The structure of the toolkit is important; file dependencies are relative to the scripts used. These dependencies include

dspace_archive directory

A directory containing Python modules that provide utilities for the creation of dspace archive objects.

foxml_class directory

A directory containing Python modules that provide utilities for the creation of FOXML objects.

utils directory

A directory containing Python modules that provide general utilities almost all scripts within the migration toolkit.

4 Get ADT HTML using single URL

4.1 Name of the script

get_adt_html.py

4.2 Location of script

migration_toolkit/adt/python/get_adt_html.py

4.3 Purpose of script

The **get_adt_html.py** script is the Python script that harvests all of the items in an ADT repository. The script captures the metadata 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, note you must issue this command from within the directory that contains the **get_adt_html.py** script

This script creates a temporary DSpace simple 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.

Each item is represented by one item directory within **dspaceArchive**. These files are numbered consecutively; first directory is called **0**, **1** and so on.

The datastreams for each object are stored in the corresponding numbered directory. For example the datastreams in the **0** directory contain three non XML 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 a file named **contents** and a temporary xml file, **temp.xml**, containing basic dublin core metadata..

4.4 Parameters/Arguments

adtUrl

This is the link to the ADT repository that you will be harvesting

4.5 Syntax

```
python get_adt_html.py adtUrl
```

4.6 Example

```
python get_adt_html.py http://www.an_adt_repository.com.au
```

5 XSL Transform (temp.xml to MARC.xml)

5.1 Name of the script

xsl_transform.py

5.2 Location of script

migration_toolkit/xsl_transform.py

5.3 Purpose of script

The **xsl_transform.py** script is a Python script that iterates through a dspace archive. It carries out an XSL transformations on one XML file per item within the archive. A new XML file is created as a result of each transformation. This file is stored along side the original XML file in the item.

5.4 Parameters/Arguments

InputFile

Filename of the XML file in the archive to be converted.

XslFilePath

File path to the stylesheet used for the XSL transformation

OutputFile

Filename of new XML file that will be generated as part of conversion process

ArchiveName

Name of the archive to be accessed

RemoveInputFile

Remove the original XML file file? - set to False

5.5 Syntax

```
python xsl_transform.py InputFile  
XslFilePath OutputFile ArchiveName RemoveInputFile
```

5.6 Example

```
python xsl_transform.py temp.xml  
xsl/xml_to_marc.xsl marc.xml dspaceArchive False
```

6 Create MARC controlfield tag 008

6.1 Name of the script

`create_marc_controlfield_tag_008.py`

6.2 Location of script

`migration_toolkit/create_marc_controlfield_tag_008.py`

6.3 Purpose of script

The basic metadata does not have a MARC controlfield tag that is applicable to the contents of each record. This script iterates through the archive and inserts the correct controlfield tag into the marc.xml file.

6.4 Parameters/Arguments

ArchiveName

name of the archive to be accessed.

6.5 Syntax

```
python create_marc_controlfield_tag_008.py ArchiveName
```

6.6 Example

```
python create_marc_controlfield_tag_008.py dspaceArchive
```

7 XSL Transform (marc.xml to dublin_core.xml)

7.1 Name of the script

xsl_transform.py

7.2 Location of script

migration_toolkit/xsl_transform.py

7.3 Purpose of script

The **xsl_transform.py** script is a Python script that iterates through a dspace archive. It carries out an XSL transformations on one XML file per item within the archive. A new XML file is created as a result of each transformation. This file is stored along side the original XML file in the item.

7.4 Parameters/Arguments

InputFile

Filename of the XML file in the archive to be converted.

XslFilePath

File path to the stylesheet used for the XSL transformation

OutputFile

Filename of new XML file that will be generated as part of conversion process

ArchiveName

Name of the archive to be accessed

RemoveInputFile

Remove the original XML file file? - set to False

7.5 Syntax

```
python xsl_transform.py InputFile  
XslFilePath OutputFile ArchiveName RemoveInputFile
```

7.6 Example

```
python xsl_transform.py marc.xml  
xsl/marc_dc.xsl dublin_core.xml dspaceArchive False
```

8 PDF to full text

8.1 Name of the script

pdf_to_full_text.py

8.2 Location of script

migration_toolkit/pdf_to_full_text.py

8.3 Purpose of script

The **pdf_to_full_text.py** script is the Python script that iterates through a dspaceArchive and converts the harvested pdf files to **fulltext**.

8.4 Parameters/Arguments

ArchiveName

Name of the dspaceArchive that contains the pdf files

8.5 Syntax

```
python pdf_to_full_text.py ArchiveName
```

8.6 Example

```
python pdf_to_full_text.py dspaceArchive
```

9 Archive To FOXML

9.1 Name of the script

archive_to_foxml.py (if you are building FOXML for VITAL 3.X.X system)

archive_to_foxml_vital_2.py (if you are building FOXML for VITAL 2.X.X system)

9.2 Location of script

migration_toolkit/archive_to_foxml.py

migration_toolkit/archive_to_foxml_vital_2.py

9.3 Purpose of script

The **archive_to_foxml.py** script is the Python script that iterates through a dspace archive. The script builds a FOXML file for each item in the dspace archive using the MARC, Dublin Core and MODS metadata contained within each item. All FOXML files created during this process are stored in a single directory (specified as an argument)

9.4 Additional Information

It is important at this step to ascertain whether you have any non XML data streams as part of your migration.

How do I tell if I have non XML data streams?

There are a couple of ways to determine this. Firstly, the script `obtain_files_for_archive.py` is designed to fetch non XML data streams. If you have run this script then you would have non XML data streams as part of your ingest. Secondly, you can list the contents of a few items in your dspace archive and see if there are any non XML data streams present such as PDF files or full text files.

If non XML data streams exist in your archive (PDF, full text etc), you are required to provide a URL where these data streams can be served during ingest. You can serve these files on an independent web server or you can use a Python simple server on the same machine that VITAL is running on.

If you will be using the python simple server as part of your ingest into Fedora then use `http://localhost:8000` for the `URLforNonXmlDataStreams` argument below. If you will be using an existing independent web server during the Fedora ingest then enter the full URL to where the non XML data streams will be served during ingest.

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

9.5 Parameters/Arguments

archiveName

Full path to name of the archive to be accessed (write down the value that you put for this argument, it is needed during the ingest into VITAL/Fedora)

startNum

Starting number for the Fedora PID increment

PIDPrefix

Name of the Fedora 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 Dublin Core file contained in the Archive. Set to False if MODS file is not present.

MODSDataStreamState

Set this to Active (A), Inactive(I) or Deleted (D). or False if MODS file is not present

URLforNonXmlDataStreams

Set this to the full URL where non XML data streams can be served during ingest

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

9.6 Syntax

```
python archive_to_foxml.py archiveName startNum PIDPrefix
outputDirectory labelPrefix foxmlObjectState MARCFileName
```

```
MARCDataStreamState DCFileName DCDataStreamState MODSFileName  
MODSObjectState URLforNonXmlDataStreams
```

9.7 Example using external web server to host non XML data streams

```
python archive_to_foxml.py dspaceArchive 0 vital foxml_items  
Imported_Items A marc.xml A dublin_core.xml A False False  
http://servername/directoryname
```

9.8 Example using Python simple server to host non XML data streams

```
python archive_to_foxml.py dspaceArchive 0 vital foxml_items  
Imported_Items A marc.xml A dublin_core.xml A False False  
http://localhost:8000
```

10 Insert XMLNS XSI

10.1 Name of the script

`insert_xmlns_xsi.py`

10.2 Location of script

`migration_toolkit/insert_xmlns_xsi.py`

10.3 Purpose of script

This script iterates through FOXML items in a single directory and sets the correct namespace declaration for the MARC XML section of the FOXML

10.4 Parameters/Arguments

outputDirectory

The name of the directory where the FOXML files are located

10.5 Syntax

```
python insert_xmlns_xsi.py outputDirectory
```

10.6 Example

```
python insert_xmlns_xsi.py foxml_items
```

11 Preparing FOXML data for Fedora Ingest

1. Log into the machine that VITAL/Fedora is running on as the dbadmin user.
2. Copy the directory containing all of the FOXML items created during this migration to the /home/dbadmin directory on the VITAL/Fedora server, ensure that dbadmin has ownership of this directory after it has been copied.

12 Preparing non XML data for Fedora ingest

12.1 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/Fedora 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 dspace archive directory (created during this migration) to the /home/dbadmin directory on the VITAL server. Make sure that the name of the dspace archive directory is not changed in any way during the copying process.
4. Execute the following command from within the /home/dbadmin directory

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

This command will invoke Python and start the SimpleHTTPServer. Allowing Fedora to fetch the items (this is done based on a URL in the FOXML. This URL was constructed during the archive_to_foxml.py script).

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 data streams are on the same server as the VITAL repository.

12.2 Using an Existing Web Server

1. Copy the dspace archive directory (created during this migration) to the directory specified as the URLForNonXmlDataStream. This was an argument in the archive_to_foxml.py script. Go to this directory on the webserver using a web browser and make sure that it is resolving and files are being served.

13 Fedora Ingest for VITAL 3

13.1 Ingest procedure

To ingest the items into VITAL complete the following procedure:

1. Log into the VITAL server as the dbadmin user
2. Navigate to the following directory on the server

```
/opt/vtls/vital/applications/fedora/client/bin
```

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

```
export FEDORA_HOME=/opt/vtls/vital/applications/fedora
export JAVA_HOME=/opt/vtls/java
```

1. Invoke the following command to start the fedora-ingest utility, where outputDirectory is the directory containing all of the FOXML items and password is the fedoraAdmin password

Note: This may take some time to complete

```
./fedora-ingest.sh d outputDirectory foxml1.0 O localhost:8080
fedoraAdmin password http ""
```

note the single O before the work localhost in the above command is the capital letter O not zero

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>

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

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

