

# Callista to Valet

# Table of Contents

1 Introduction.....	1
2 Background information.....	2
3 Downloading the Python Scripts.....	3
3.1 Script dependencies .....	3
4 Converting the csv file to xml.....	4
5 Split XML Into Archive.....	5
5.1 Name of the script.....	5
5.2 Location of script.....	5
5.3 Purpose of script.....	5
5.4 Parameters/Arguments.....	5
5.5 Syntax.....	6
5.6 Example.....	6
6 XSL Transform (temp.xml to valet.xml).....	7
6.1 Name of the script.....	7
6.2 Location of script.....	7
6.3 Purpose of script.....	7
6.4 Parameters/Arguments.....	7
6.5 Syntax.....	7
6.6 Example.....	8
7 Obtain files for valet archive.....	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.....	9
8 Place items into VALET.....	10





# 1 Introduction

## Author

Bron Dye, RUBRIC Technical Officer

Tim McCallum, RUBRIC Technical Officer

## Purpose

This document follows the steps to migration of a set of Callista records stored as a comma separated value file into VALET xml

## Audience

RUBRIC Project Partners and other users of VALET as an ingest tool.

## Requirements

Callista export file (csv)

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 VALET for ingest

## References

Official VALET 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](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/](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 Converting the csv file to xml

1. Open csv file in Microsoft Excel.
2. Go to **Data > XML > XML Source**
3. In **XML Source** frame, Click on **XML Maps...** then **Add** map
4. Locate pre-made xml file with 2 empty records, formatted ready for input data. (see Map at the end of this documentation).
5. To apply xml tags to worksheet, right-click on record node in tag list of **XML Source** frame. Choose **Map Elements** and click **OK**
6. Set Map options to untick **My Data has Headings**
7. Export XML: **Data > XML > Export** to create xml file.

## 5 Split XML Into Archive

### 5.1 Name of the script

split\_xml\_into\_archive.py

### 5.2 Location of script

migration\_toolkit/split\_xml\_into\_archive.py

### 5.3 Purpose of script

To convert a large xml file (containing multiple records) into a dspace archive format. The dspace archive format consists of one top level directory that houses multiple subdirectories. Each subdirectory represents one record from the large xml file. After this script has been executed you will find two files with in each subdirectory. A temp XML File containing metadata for one individual record (extracted from the large xml file) and a basic text file called contents. This file lists the contents of the subdirectory in which it is contained.

### 5.4 Parameters/Arguments

#### **dataFileName**

Path to the large input xml file that will be split.

#### **archiveName**

Name of the dspaceArchive that will be created (this will create the dspaceArchive in current directory)

OR

Full path to location where you would prefer the dspaceArchive to be created, must specify name of the dspaceArchive after full path when using this option (directory does not have to exist it will be created when script is run)

#### **xpath**

An xpath to match the root node for each individual record in the large xml file. For example `//*[local-name()='record']`

#### **tempXmlFileName**

Filename for the temp XML file within each subdirectory containing metadata for each record.

#### **templateFile**

The filepath to a wrapper file if needed. This parameter can be used if it is necessary to wrap the temp XML File file located within the subdirectory with extra data, for example extra namespace.

## 5.5 Syntax

```
python split_xml_into_archive.py dataFileName archiveName xpath  
tempXmlFileName templateFile
```

## 5.6 Example

```
python split_xml_into_archive.py large_xml_file.xml dspaceArchive  
"//record" temp.xml False
```

## 6 XSL Transform (temp.xml to valet.xml)

### 6.1 Name of the script

xsl\_transform.py

### 6.2 Location of script

migration\_toolkit/xsl\_transform.py

### 6.3 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.

### 6.4 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

### 6.5 Syntax

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

## 6.6 Example

```
python xsl_transform.py temp.xml  
xsl/acv_to_valet.xsl valet.xml dspaceArchive False
```

## 7 Obtain files for valet archive

### 7.1 Name of the script

`obtain_files_for_valet_archive.py`

### 7.2 Location of script

`migration_toolkit/obtain_files_for_valet_archive.py`

### 7.3 Purpose of script

The **`obtain_files_for_valet_archive.py`** script iterates through the archive downloading non xml datastreams. For example if a valet.xml (in an item of the archive) has a link to a pdf file in it, the script will go to the link, download the pdf (placing it in the directory for that item) and update the valet xml (so that VALET will know about the downloaded pdf file on ingest)

### 7.4 Parameters/Arguments

#### **dspaceArchive**

Full path to dspaceArchive

#### **fileName**

VALET file inside the dspaceArchive item containing links

#### **pathToDeadLinksFile**

Location where report file will be created

### 7.5 Syntax

```
python obtain_files_for_valet_archive.py
```

### 7.6 Example

```
python obtain_files_for_valet_archive.py dspaceArchive valet.xml /home/myuser
```

## 8 Place items into VALET

The VALET files can now be copied to the system running VALET. In this instance items were put into `/opt/vtls/valet/resource1/storage/stage1`

**Note:** This path will depend on the Step element. If this element contains 2, then place the items in `/opt/vtls/valet/resource1/storage/stage2`.