# USC - csv file data to VITAL2.1

## 1 About this document

**Author**

Tim McCallum and Bron Dye

**Purpose**

This document follows the steps to migration of a set of records stored as a comma separated value file into VITAL2.1

**Audience**

RUBRIC Project Partners and other users of Fedora repositories

**Requirements**

csv 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 VITAL2.1 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/

**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,

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)

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 generic xml 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 csv data 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.

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)

# 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.

**create_marc_controlfieldtag_008.py**

A python script to create a controlfield tag within each marc record to reflect the current date and  language used within the item.

**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.

**insert_xmlns_xsi.py**

A python script to insert a correct marc namespace into a completed foxml object.

## 3.1 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

## 3.2 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/migration_toolkit

## 4 Converting the csv file to xml

1. Open csv file in Microsoft Excel.
2. Establish the necessary marc meta-data requirements such as all datafield tags, indicators and subfield codes. Delete columns as necessary.
3. Set the conferenceYear column to text and manually change each field to the year.
4. Save file as **trimmed.csv**
5. Open Microsoft Excel
6. Go to **Data** > **XML** > **XML Source**
7. In **XML Source** frame, Click on **XML Maps...** then **Add** map
8. Locate pre-made xml file with 2 empty records, formatted ready for input data. (see marc_excel_map.xml)
9. To apply xml tags to worksheet, right-click on record node in tag list of **XML Source** frame. Choose **Map Elements** and click **OK**
10. Set Map options to untick **My Data has Headings**
11. Copy only data to be exported from csv file and paste into worksheet
12. Export XML: **Data** > **XML** > **Export** to create xml file.
13. Once extracted carry out find and replace on characters that are not displayed properly for example replace Õ with ' (comma) these characters are given to us in the original document so there is now way to display them correctly other than getting the correct characters in the original document or just replacing them manually.
14. Check for ", ," in the author tag. If this exists, it will cause a split on a comma to render incorrect author formats. Remove any additional commas.

**TIP:**

Check xml file in an xml editor to ensure that it is valid, if find-replace process has introduced unwanted whitespace the split_xml_into_archive.py script will stall.

# 5 Extract the xml

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 simple archive, in the current directory.

**Script structure:**

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

**Argument definitions:**

- [pathToDataFileName] filename of the input xml file
- [archiveName] name of the Archive
- [xpath] match the node for each item in the larger xml file
- [streamName] filename of the output xml file
- [templateFile] the filepath to a wrapping file (wrapper_file_for_USC) this wraps a <collection></collection> tag around each <record></record> set

**Example:**

```
python split_xml_into_archive.py outfile.xml dspaceArchive
//record temp.xml USC_research_system/wrapper_file_for_USC
```

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

```
csvArchive/

      0/

            contents

            dc_temp.xml

      1/

            contents

            dc_temp.xml
```

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)

# 6 Convert basic metadata to MARC

The **xsl_transform.py** script converts the **temp.xml** files in the individual item directories of the **csvArchive** directory to VITAL compliant **marc.xml**.

**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 duringe conversion.
- [ArchiveName] the name of the archive to be accessed
- [RemoveInputFile] Remove or retain the input file? - True or False

**Example:**

```
python  xsl_transform.py temp.xml
USC_research_system/xsl/recursive_generic_xml_to_marc.xsl
marc.xml dspaceArchive True
```

## 6.1 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 dspaceArchive
```

## 6.2 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] filename of the input xml file to be converted.
- [XslFilePath] file path to the stylesheet to be used for the conversion
- [OutputFile]  filename to be created during conversion. t
- [ArchiveName]  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 dspaceArchive False
```

# 7 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.

**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 to be used for the conversion
- [OutputFile] filename to be created following conversion.
- [ArchiveName] name of the archive to be accessed
- [RemoveInputFile] Remove or retain the input file? -  False

**Example:**

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

# 8 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] 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
```

## 8.1 Insert name space in to foxml objects

This script ensures that the correct marc namespace appears within the foxml object. We have inserted it at this point because insertion earlier in the process changed the formatting and meant that the marc when viewed within VITAL repository was not validating correctly.

**Script structure:**

python insert_xmlns_xsi.py [outputDirectory]

**Argument definition:**

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

**Example:**

```
python insert_xmlns_xsi.py exportedItems
```

## 8.2 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 [files] foxml1.0 O localhost:8080
   fedoraAdmin [password]
   ```
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

## 9 marc_excel_map.xml

```
<?xml version="1.0"?>
<collection>
<record>
<author></author>
<issn_isbn></issn_isbn>
<conferenceYear></conferenceYear>
<conferenceLocation></conferenceLocation>
<pages></pages>
<itemType></itemType>
<publication></publication>
<conferenceName></conferenceName>
<number></number>
<pageEnd></pageEnd>
<pageStart></pageStart>
<place></place>
<destCode></destCode>
<publisher></publisher>
<title></title>
<volume></volume>
<year></year>
</record>
<record>
<author></author>
<issn_isbn></issn_isbn>
<conferenceYear></conferenceYear>
<conferenceLocation></conferenceLocation>
<pages></pages>
<itemType></itemType>
```

<publication></publication>

<conferenceName></conferenceName>

<number></number>

<pageEnd></pageEnd>

<pageStart></pageStart>

<place></place>

<destCode></destCode>

<publisher></publisher>

<title></title>

<volume></volume>

<year></year>

</record>

</collection>