# FuzzyPhoto AHRC AH/J004367/1

## Work Package 3 Report: Batch Loader

Dr. Jethro Shell, Mr. David Croft

25th October, 2013

Status: Draft

Distribution Type: Public

Keywords: Batch Loader, MySQL, Python, Database, Web Service

# Contents

# 1. Introduction

It was identified within the project structure that alongside the incorporation of data from the partner organisations directly donated, to maintain sustainability there would be a need for a consistent system of data upload on a periodic basis. A review was carried out (See report Batch Loader Requirements Report v1.1) to summarise the basic needs of such a system. This report discusses the development of a batch loader to import additional data from three key partner organisations from within the interest group.

Outputs of this work package comprimises:

1. This report.
2. A Python based implementation for the automated extraction and uploading of data to the FuzzyPhoto database cluster.

The elapsed time for this work package was two months. The resources required to complete this work package were 24 person-days.

# 2. Outlined Batch Loader Structure

The prime goal of the project is to produce links between images maintained across partner websites and additionally sourced data. To sustain the relevance of this data, there was identified a need to update this information on a periodic basis. The raw data supplied produces issues regarding consistency and clarity. This is discussed in depth within the report for Work Package 3. Based upon the findings in Work Package 3, a decision was taken to pursue data sources that were persistently more consistent throughout. The chosen data is to come from sources at the Victoria and Albert Museum, University of St. Andrews, and The Library of Birmingham.

The outline batch loader structure consists of:

1. Uploading of data.
2. Identification of correct and relevant source.
3. Transfer to defined database format.
4. Upload to LIDO database.

This outline structure is shown in Figure 1. The overall structure of the batch loader follows that outlined in Batch Loader Requirements Report v1.1. In this report two approaches were proposed: an individualised approach and a generalised, modular approach. Following the analysis of the data, it was decided that a more robust process would result from the use of a individualised configuration. The individualised approach constructs different processes for each institution in order to upload, identify, transfer and convert the data to the final database. The speed of implementation is proportional to the quantity of institutions incorporated. As this is only three, this method can supply both a robust methodology with minimal development time. The static nature of the method does not, however, allow for changes to the partner institutions data structures. This needs to be taken into consideration. The following sections take each part of the batch loader structure in turn, providing a discussion of the process.

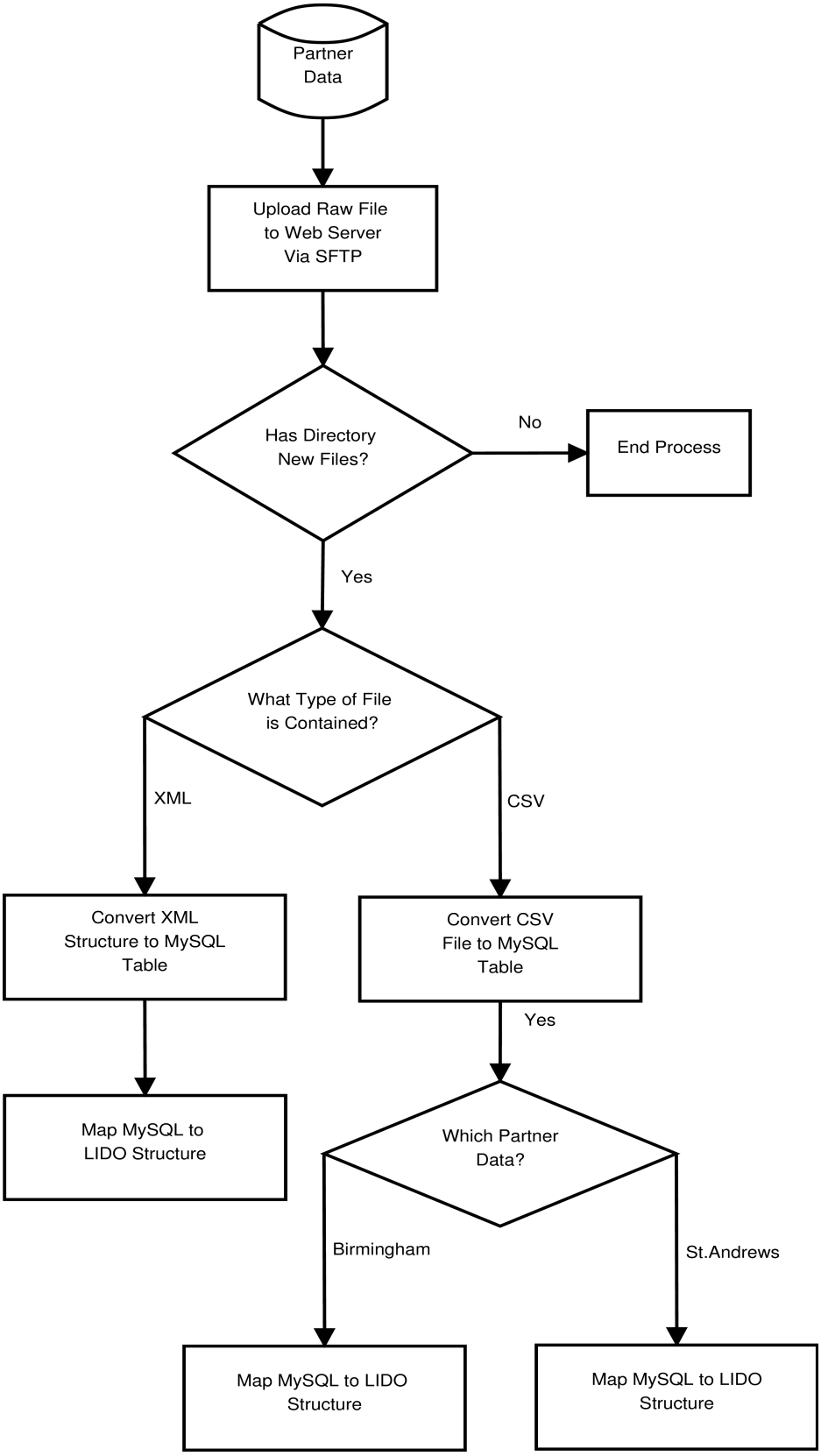


Figure 1. Outline of the Batch Loader Process.

## 2.1 Upload

In order for new data to be accessible to the batch loader for processing, it initially is loaded to a server through the use of Secure File Transfer Protocol (SFTP). SFTP allows the transferring of

files from a client (partner) to a host (Fuzzyphoto project server) over a connection with increased security. The SFTP is configured to use Public-key cryptography. Public-key cryptography uses two separate keys, one which is private and one which is public. The public key is used to encrypt plaintext or to verify a digital signature with the the private key being used to decrypt ciphertext or create a digital signature. This method needs to be established on both the client and host units.

SFTP also allows the files to be transferred with basic attributes such as timestamps, an important aspect of this process and an advantage over common File Transfer Protocol (FTP) which does not provision for this.

Each institution will add data on a periodic basis (currently standing at 6 months) to the server. The data is subsequently transferred to the data cluster to be processed.

## 2.2 Identification

The construction of the batch loader application used the Python programming language. A flexible, adaptable language, Python contains numerous libraries that allow for the processing of language and incorporation of SQL statements, both of which are integral to the batch loader. A brief discussion of how to run the application is given in the Appendix of this document.

The first stage of the main application is to identify the contents of the directories uploaded. The batch loader will periodically run to check the contents of the directories. This process looks to identify a number of key elements:

a) Is there data in the directories?

b) Is the file of the correct type for that institution?

c) Is the file uploaded younger than previous data uploaded?

### 2.2.1 Data Contained in Directories

To identify whether there are any contents within the directories, each subdirectory of the defined directory is scanned. Any files that are contained within the directories are recorded. If the directories are empty, the whole process is bypassed. This is accomplished through the use of the Python `os.path` and `directory` methods which allow for the navigation through directories. In the batch loader, the method `_Check__dirName` in class `upload` is used.

### 2.2.2 Correct File Type

If the directories are found to contain data, each file is processed to ascertain its type. Each institution has a predefined file type associated with its process I.E XML is associated with the Victoria and Albert Museum. This allows for a robust process to be employed. To gain the types, each file is examined using its extension. If the extension does not relate to the specified type, the file is not used. This processes uses the Python `os.path.` In the batch loader, the method `_Check__fileType` in class `upload` is used.

### 2.2.3 Compare Timestamps of Files

The comparison of the files within the directories has dual importance. Files may remain within the directory with the same name from previous uploads, also partner institutions may upload old files. Both cases need to be taken into account.

To accomplish this, firstly the timestamp of the uploaded file is gained using the `_Check__filename`, `_Check_getTimeStamp` and `_Check_timeStampUpload`

methods from the `upload` class. The filename is gained from the file without the inclusion of its extension. This is passed to the `_Check_timeStampUpload` method along with the timestamp, acquired using the `_Check_getTimeStamp` method. To compare previous files, a CSV file of timestamps mapped to the appropriate file name is contained within the configuration. The `_Check_timeStampUpload` compares the information supplied to data within the file. Based on this data, the method produces a `true` value if the uploaded file is younger.

Based on the output from the method, the `_Check__uploadTimes` confirms the file, and adds the new timestamp to the configuration file removing the previous timestamp. If the file is older than the data contained in the configuration file, the subsequent processes are skipped.

## 2.2 Upload

Each of the raw sources provided by the partner organisations offer unique problems in terms of processing and cleaning of data. For an in depth discussion of the problems encountered, and solutions provided to this process, see the Work Package Report 3. Due to the nature of the raw data provided, three of the partner organisations have been focussed on. The batch loader approached the uploading of each of the datasets in differing manners due to the nature of the raw data.

### 2.2.1 Library of Birmingham

The Library of Birmingham data is provided in a raw form as an excel sheet. To provide a consistent format, it will be requested that they provide a Comma-separated Values (CSV) file. The Birmingham data is formed from a single table that consists of eight columns. In line with previous work carried out in Work Package 3, a single MySQL table is constructed.

To be able to revert any changes made to the data and provide a MySQL structure to work from, the uploading process loads the raw data to MySQL interim tables. As the Birmingham data is a CSV file (currently Excel), the process to upload this format can use methods within the standard MySQL structure.

The partners chosen to provide continued data support were specifically sort due to the cleanliness of the source. Whilst the Birmingham data provided requires minimal cleaning, the data format was shown to require pre-processing before entering the LIDO format. This will be discussed further in later sections.

Each of the institutions data is loaded into a specific directory. As was previously discussed, this directory is processed to check for the correct file type. The Library of Birmingham data is to be supplied as a CSV file. The final structure of the Fuzzyphoto database is an entity relationship MySQL form. To transfer the raw data to this format, the batch loader goes through three steps:

1. Create database – To increase the robustness of the process, the batch loader ascertains if the database has been previously created. If not, it is formed using the database name supplied from the main task. This element of the process uses the `_Data_checkDatabase` method from the `database` class contained within the `_Data_uploadCSV` method.
2. Create table - Before adding the defined table, this is also checked. Using the MySQL query:

   ```
   "SELECT COUNT( * ) FROM information_schema.tables WHERE
   table_name = %s" % (convTable)
   ```

   where convTable is the table to added, a positive or negative result can be gained. Based upon this, a new table is created. The table names are required to be added to the table. To allow this to be an automated process, a new table is created and then altered using column names extracted from the CSV file. It is assumed that the first column of the supplied CSV

file contains the column names. Python supplies a CSV reading library that allows for the extraction of data from CSV files line by line. This is used to gain the necessary information.

3. Load data – As the supplied raw data is in a CSV file format, MySQL supplied a process for uploading the data directly through a single statement. The data is loaded into the constructed table using a `LOAD DATA LOCAL INFILE` statement. To incorporate the use of column names extracted from the raw data, the statement is generated automatically within the `_Data_uploadCSV` method.

### 2.2.2 University of St. Andrews

The University of St. Andrews raw data is provided as multiple CSV files. Each file holds a separate element of a single record. Again keeping in line with previous investigation, each of the CSV files is converted to a single table before being imported to the LIDO format.

The uploading of the University of St. Andrews data follows a similar process to the Library of Birmingham. Separation from this process only occurs due to the multiple nature of the file structure.

### 2.2.3 Victoria and Albert Museum

The Victoria and Albert Museum raw data is provided as a single Extensible Markup Language (XML) file. The complexity of the Victoria and Albert structure, coupled with the use of XML required the use of a different approach to the uploading of both the Library of Birmingham and University of St. Andrews data. To gain access to the XML structure, the Python programming language contains an ElementTree method. This allows for the storing of hierarchical data structures in memory. XML can be represented in this way, allowing for the reading of elements and sub-elements within the tree.

The process to upload the Victoria and Albert Museum data is:

1. Create the database – A method (`Data_checkDatabase)` is used to check if the database has been added previously.
2. The XML file is checked for elements that need to be expanded, those elements that have any sub-elements.
3. Checks are required to increase the automation and robustness of the approach. If the element has sub-elements, they are navigated to. Where '_' is found as a sub-element, the previous element is used as the table name. If '_' is the element and no text is contained within the sub-element, no table is formed.
4. If a table relating to the element, or sub-element has not been created, a new table is formed. A special case is made for the element sys_id. This table is formed with the addition of a primary key of `id`. All of the tables can be linked to this table.
5. Data is added to the table created based on the information in each element. This is carried out using the `Data_addData` method.
6. Large quantities of insertions into a MySQL database can reduces the speed of its execution. To optimise the process, the transactions are committed to the database after a counter reaches 1000, reducing the overhead required.

## 2.3 Backing Up the Database

To mitigate the changes that the batch loader introduces to the database, a back up is made of the database to a local directory (/backup/). The back up uses the standard `mysqldump` command. If there is a need to return the database to its previous state, a SQL command can be used combined with the file produced to return the database.

## 2.4 Mapping to LIDO

Following the uploading of the data to the MySQL tables, each table or set of tables is mapped to the LIDO structure. Each institution has a separate configuration based on the structure of the data and the mapping of fields between the raw data and the final LIDO format. The following sections will give a brief overview of the transfer of the data from the MySQL interim tables to the LIDO format.

### 2.4.1 Deletion of Previous Data

In order to add the new data to the database, all previous data relating to each of the institutions is removed from the database. To achieve this, the `internal_remark` column is used. This is outside of the LIDO schema but was added to allow an ease of navigation when dealing with each separate institution.

To facilitate the removal of the data, it is necessary to disable the use of foreign key checks. This is achieved using the MySQL statement:

```
SET FOREIGN_KEY_CHECKS=0;
```

Following this command, the data from each individual institution is deleted. Once this has been carried out, the foreign key checks are reactivated. To achieve this, the below statement is used:

```
SET FOREIGN_KEY_CHECKS=1;
```

### 2.4.2 Mapping the Library of Birmingham Data to LIDO

The basis for each of the LIDO configuration schemas was derived from the work carried out in the cleaning, processing and formation of the initial LIDO data structures. This can be referred to in the report for Work Package 3. Python, through the use of MySQLdb, affords the use of direct interaction with MySQL databases. This approach was used in the previous steps.

Each of the mappings is carried out using SQL commands. The following is a summary of the process:

1. Insert into the fuzzyphot.lido_actor(name) the Creator from the Birmingham data

   ```
   INSERT INTO fuzzyphoto.lido_actor (name) SELECT Creator FROM birmingham
   ```

2. Insert the photograph fields.

   ```
   INSERT INTO fuzzyphoto.lido_photography (title, object_description, object_published_id)  SELECT Title, Decscription, Reference FROM birmingham
   ```

3. Gain the LIDO record to insert into the event table (lidoId).

   ```
   SELECT lido_record_id FROM fuzzyphoto.lido_photography
   ```

4. Gain the items to insert for event.

```
SELECT Date FROM birmingham
```

5. The date is cleaned and split into a separate earliest and latest category using a dateHandler class. The date handler uses a combination of regular expressions and predefined rules to convert the data into a day-month-year format.
6. Gain the items to insert for format (formating).

```
SELECT Format FROM birmingham
```

7. Gain the items to insert for dimensions (dimensions).

```
SELECT Dimensions FROM birmingham
```

8. Insert each of the elements into the fuzzyphoto.lido_event table

```
 INSERT INTO fuzzyphoto.lido_event
(lido_photography_lido_record_id, earliest_date, latest_date,
material, object_measurement, event_type) VALUES (%s, %s, %s,
%s, %s, 'Shoot')" % (lidoId, earliest, latest, formating,
dimensions)
```

### 2.4.3 Mapping the St. Andrews Data to LIDO

The insertion of data into the LIDO schema from the University of St. Andrews tables is relatively simplistic. The process constitutes:

1. Insert information regarding the artist into the fuzzyphot.lido_actor table.

```
INSERT INTO fuzzyphoto.lido_actor (name, birth_date,
death_date) select NamCitedName,BioBirthDate, BioDeathDate
from ColArtis
```

2. Insert the data relating to the photograph into the fuzzyphoto.lido_photography table.

```
INSERT INTO fuzzyphoto.lido_photography (object_published_id,
object_type, title, object_description) SELECT
ColObjectNumber, PhoRecordLevel, ColMainTitle, ImaDescription
from ecatalog
```

### 2.4.4 Mapping the Victoria and Albert Data to LIDO

The structure of the Victoria and Albert museum data requires a more a more complex use of SQL statements to convert the data. The larger quantity of tables and the spread of information produces a need for the use of a SQL joins. The process is as follows:

1. Insert data into the fuzzyphoto.lido_photography table.

```
INSERT INTO fuzzyphoto.lido_photography(object_published_id,
object_description, link_resource, title) SELECT
vanda_museum_number.vanda_museum_number,
spec_physical_description.spec_physical_description, url.url,
spec_title_field FROM vanda_museum_number INNER JOIN
spec_physical_description ON vanda_museum_number.sysid =
spec_physical_description.sysid INNER JOIN url ON
vanda_museum_number.sysid = url.sysid INNER JOIN
spec_title_field ON vanda_museum_number.sysid =
```

```
spec_title_field.sysid WHERE vanda_museum_number.sysid =
(SELECT vanda_museum_number.sysid FROM vanda_museum_number)
```

2. Get the LIDO record number to insert defined as `LidoId`.
   ```
   SELECT lido_record_id FROM fuzzyphoto.lido_photography
   ```

3. Get the width dimension of the photograph.
   ```
   SELECT dimension_Height FROM dimension_Height WHERE
   dimension_Height.sysid = (SELECT vanda_museum_number.sysid
   FROM vanda_museum_number.sysid)
   ```

4. Get the height dimension of the photograph.
   ```
   SELECT dimension_Width FROM dimension_Width WHERE
   dimension_Width.sysid = (SELECT vanda_museum_number.sysid
   FROM vanda_museum_number ORDER BY vanda_museum_number.sysid)
   ```

5. Concatenate the width and height into a single field defined as `measurement`.
6. Insert the data into the fuzzyphot.lido_event table.
   ```
   INSERT INTO fuzzyphoto.lido_event(material, earliest_date,
   latest_date, object_measurement, technique,
   lido_photography_lido_record_id, event_type) SELECT
   mus_materials_techniques_note.mus_materials_techniques_note,
   earliest.earliest, latest.latest, %s,
   spec_title_field.spec_title_field, %s, 'Shoot' FROM
   mus_materials_techniques_note INNER JOIN earliest ON
   mus_materials_techniques_note.sysid = earliest.sysid INNER
   JOIN latest ON mus_materials_techniques_note.sysid =
   latest.sysid INNER JOIN spec_title_field ON
   mus_materials_techniques_note.sysid = spec_title_field.sysid
   WHERE mus_materials_techniques_note.sysid = (SELECT
   vanda_museum_number.sysid FROM vanda_museum_number)%
   (measurement, lidoId)
   ```

# 3. Conclusion

This report outlines the structure of the batch loader to be used in the Fuzzyphoto project. It supplies an insight into the operation of the process to maintain sustainability of the data for the construction of similarity links between meta-data relating to historical images.

# 4. Appendix

To run the included Python files, Python must be installed. The Python files are designed for a Linux based system. Most modern versions of Linux come with Python out of the box such as Ubuntu, Fedora, Redhat Enterprise (RHEL) and CentOS.

If the user is within the PartnerData structure, the python script can be run as:

```
python mainProcess.py -t <timestampFileLocation>
```

`-u <uploadedFilesDirectory>`

Running the batch loader script from the terminal requires two additional arguments. If theses are not passed, the default parameters will be used. The default are located as an relative `../upload/` directory within the PartnerData structure. Within the upload folder, the `timestamp_config.csv` file will be contained. This file holds the timestamps relating to the files uploaded previously.