SpredNonDicomUpload Documentation

Introduction

This is some brief documentation for the work I did for the Mouse Imaging Centre (MICe) and the Ontario Brain Institute (OBI). The OBI is working on a big data solution for neuroscience data, called Brain-CODE, and it was my job to upload approximately 50 GB of mouse brain imaging data from MICe into the imaging side of Brain-CODE. The imaging side of Brain-CODE uses the Stroke Patient Recovery Research Database (SPReD).

The upload process uses an R script to generate the subject metadata, then uses a Python script to map the subject metadata to the XNAT XML schema, to extract other metadata, to package files together for an upload, then to structure HTTP PUT web service calls to the XNAT REST API.

To learn how to use these scripts, consult the Workflow below and make sure you have everything installed listed in the Dependencies section. Development Notes / Rationales contains information for developers who might be interested in extending this work (maybe writing proper classes instead of one long upload script).

Workflow

1. Near the top of the `GenerateSubjectMetadata.R` script, supply the paths to the distortion corrected folder and processed folder for each strain you want to upload along with specifying the gf file and the strain label to use for each strain (label used as a prefix for all the data files – e.g. NLGN) in the `GenerateSubjectMetadata.R` script. Run the script. This will create a file called `SubjectMetadata.csv` in your working directory.

2. Move the .csv file that is produced to the same directory where you will run the Python script. Open up the .csv file and add values to the columns for any additional metadata you might have (e.g. age, gender, handedness, race, weight, height). Race is the mouse background (e.g. C57BL/6), age is an integer representing days old, gender is M or F.

3. Open up the `SpredNonDicomUpload.py` script and scroll down to the function called `init_project_constants()`. It is here that project and upload-wide constants are defined (e.g. project name, site code, SPReD url, etc.). You'll want to ensure that these are correct for the given project and upload.

4. Run the Python script on the command line with: `python SpredNonDicomUpload.py`. Supply your username and password, and it will upload subjects, sessions, scans, and any associated files to the project and site at the specified SPReD url. By default, the script is interactive to help ensure that the proper files are uploaded. To disable interactivity and instead run the script automatically, use the command: `python SpredNonDicomUpload.py -a`

5. Log on to SPReD and verify that the subjects, sessions, scans, and associated files were created successfully. Also, a log file will have been created in a `logs` subdirectory of the folder containing `SpredNonDicomUpload.py`.

Dependencies

- mincinfo from minc-tools (https://github.com/BIC-MNI/minc-tools)
- python packages (datetime, jeffs_utilities, math, numpy, os, pandas, pdb, requests, subprocess, sys, zipfile)
  - jeffs_utilities can be found at: https://github.com/jeffbruce/SpredNonDicomUpload

Development Notes / Rationales

Incorrect Field Mappings This section describes some MINC fields that were difficult to map to the allowable XNAT XML elements and attributes.

- the difference between `study:start_time` and `vnmr:time_submitted` in the MINC headers is unknown
  - it is unclear whether these fields would map to SPReD session and scan, respectively
  - for MICe's purposes, there will only ever be 1 scan per session, so it doesn't matter for the current implementation
- `fov` (x and y) in XNAT is of type integer, yet these variables are of type float in the MINC header
  - to convert to mm, simply multiplied by 10
- `fov` and `matrix` fields in XNAT are two dimensional (only x and y), however, all images at MICe are 3 dimensional without slices
  - what is the best way of handling this behaviour? possibly supply z dimension in the `frames` XNAT XML element
- duration goes up to a maximum of around 16 hours
  - there is a bug in XNAT for handling `xs:duration` values
- it's not clear what fields to use for mouse background, strain, and genotype

Rationales

1. Upload process is interactive by default.
   - any time large amounts of data are downloaded/uploaded someone should probably oversee the process
   - there is a command line argument to disable interactivity
     - this is safe because the script will automatically shut off if it fails during one step of the process, and it will notify the user what subject and action (create subject, session, scan, resource, file) that it failed on
2. If a strain already exists in the SPReD project to which data is being uploaded and that strain is selected to be uploaded, the existing subjects for that strain in the SPReD project will be deleted and replaced with the ones being uploaded. This allows one to go back and reupload a strain, maintaining the original strain code for that strain.
3. Extending this module to work with any non-DICOM data would be difficult.
   - however, this script is still useful skeleton code for people with their own non-DICOM data
   - the python script hard codes a number of things:
     - there is only 1 project, 1 session, 1 scan, and 2 resources (1 for distortion corrected image, 1 for registrations) associated with a subject
     - metadata is generated with a separate R script, and must conform to a specific schema in order to qualify as a valid upload
   - all of the QA is specific to MICe
4. A log file is produced in a subdirectory called `logs` every time the script is run, allowing one to quickly see the actual files that were uploaded.
5. The bottleneck of this script is the actual uploading of the files, because some files can be as large as 250 MB.
   - currently it takes about 10-15 seconds to upload a 75 MB file

TO DO

- verify automatically that the upload was successful (both metadata and actual file should be accurate)
  - probably more work than it's worth
  - how to check that metadata are accurate?
    - manually verify the metadata
    - GET the data and check it against what was added
  - how to check that the MINC file is accurate?
    - is there information contained in the HTTP response body that I can use?
    - use a checksum somehow?
    - performing a GET and checking would be somewhat expensive
    - manually spot check certain files with a trained eye?