**XNAT scripting tutorial**

In the previous part of the tutorial you were introduced to the XNAT web interface. This is useful to inspect data and perform simple operations. However, when the size of a study increases this might become cumbersome. In that case, XNAT allows users to interface via a REST API.

The XNAT REST API allows users to work with xnat via scripts. The REST API is an interface that is language independent and is build on top of HTTP. Operations are carried out by HTTP requests with one of the verbs **GET**, **PUT**, **POST** or **DELETE**. The **GET** request is generally used for retrieving data, whereas the **PUT**, **POST**, and **DELETE** are used for modifying data.

A simple **GET** request can be send by simply putting the target url in a web browser and looking at the result. For a sending more complex HTTP requests, you can for example use **curl** (a command-line tool for linux), **postman** (an extension for the chrome browser), or the **requests** package for Python. In this tutorial we will use **xnatpy**: a Python package that is build on top of **requests**.

**Create a connection**

Start up ipython and create a connection, it will prompt you to enter the password for the user test:
Once a connection is established it is possible to browse the projects. This can achieved by simply looking at the `projects` attribute of the session:

```python
In [3]: session.projects
Out[3]: <XNATListing (brainimages, Brain Image Analysis): <ProjectData Brain Image Analysis (brainimages)>, (fastrtutorial, Fastr Tutorial): <ProjectData Fastr Tutorial (fastrtutorial)>>
```

We can select a project by simply indexing the project listing using other the id or name of the project:
In [4]: project = session.projects['brainimages']

In a similar fashion one can explore and selection subjects (and experiments) from the project:

In [5]: project.subjects
Out[5]: <XNATListing (demo_S00081, ANONYMIZ): <SubjectData ANONYMIZ (demo_S00081)>>

**Importing data into XNAT**

In the earlier part of the tutorial you uploaded data to XNAT and used the prearchive. This functionality is also exposed using `xnatpy`. It is both possible to upload data straight into the archive and to upload via the prearchive with more controlled archiving.

For the uploading we use an import service. This service automatically sorts the DICOM data in an zip archive into scans. Uploading the data we used earlier straight into the archive under the subject and experiment `ANONYMIZ2` is one simple command:

```
# # Import directly into archive:
In [6]: experiment = session.services.import_('human/hachterberg/temp/ANONYMIZ.zip',
project='brainimages', subject='ANONYMIZ2', experiment='ANONYMIZ2')
```

As it is dangerous to add data straight into the archive due to lack of reviewing, it is possible to also upload the data to the prearchive first. This can be achieved by adding the `destination` argument as follows:

```
# # Import via prearchive:
In [7]: prearchive_session =
session.services.import_('human/hachterberg/temp/ANONYMIZ.zip', project='brainimages',
destination='/prearchive')

In [8]: prearchive_session
Out[8]: <PrearchiveSession brainimages/20161107_114859342/ANONYMIZ>
```

Once the data is uploaded (either via `xnatpy` or other means) it is possible to query the prearchive and process the scans in it. To get a list of `sessions` waiting for archiving use the following:

```
In [9]: session.prearchive.sessions()
Out[9]: [<PrearchiveSession brainimages/20161107_114859342/ANONYMIZ>]
```
Once the data in the prearchive is located it can be archived as follows:

```python
In [10]: prearchive_session = session.prearchive.sessions()[0]
In [11]: experiment = prearchive_session.archive(subject='ANONYMIZ3',
   experiment='ANONYMIZ3')
In [11]: experiment
Out[11]: <MrSessionData ANONYMIZ3 (demo_E00092)>
```

⚠️ Note

It is worth noting that it is possible to inspect the scan before archiving: one can look at the status, move it between projects, list the scans and files contained in the scans.

**Download data**

It is possible to list the scans contained in an experiment and explore them further:

```python
In [12]: experiment.scans
Out[12]: <XNATListing (1001-MR2, FLAIR): <MrScanData FLAIR (1001-MR2)>, (1001-MR3, T1): <MrScanData T1 (1001-MR3)>, (1001-MR1, PD): <MrScanData PD (1001-MR1)>>
In [13]: experiment.scans['T1']
Out[13]: <MrScanData T1 (1001-MR3)>
```

In some cases you might want to download an individual scan to inspect/process locally. This is using:

```python
In [14]: experiment.scans['T1'].download('/home/hachterberg/temp/T1.zip')
Downloading http://145.100.58.186/xnat/data/experiments/demo_E00091/scans/1001-MR3/files?format=zip:
13035 kb
Saved as /home/hachterberg/temp/T1.zip...
```

As you can see, the scan is downloaded as a zip archive that contains all the DICOM files.

If you are interested in downloading all data of an entire subject, it is possible to use a helper function that downloads the data and extracts it in the target directory. This will create a data structure similar to that of XNAT on your local disk:
In [15]: subject = experiment.subject

In [16]: subject.download_dir('/home/hachterberg/temp/')
Download http://145.100.58.186/xnat/data/experiments/demo_E00091/scans/ALL/files?format=zip:
23736 kb
Downloaded image session to /home/hachterberg/temp/ANONYMIZ3
Downloaded subject to /home/hachterberg/temp/ANONYMIZ3

To see what is downloaded, we can use the linux command find from ipython:

In [17]: !find /home/hachterberg/temp/ANONYMIZ3
/home/hachterberg/temp/ANONYMIZ3
/home/hachterberg/temp/ANONYMIZ3/ANONYMIZ3
/home/hachterberg/temp/ANONYMIZ3/ANONYMIZ3/scans
...

Custom variables

The custom variables are exposed as a \texttt{dict}-like object in \texttt{xnatpy}. They are located in the \texttt{field} attribute under the objects that can have custom variables:

In [18]: experiment = project.subjects['ANONYMIZ'].experiments['ANONYMIZ']

In [19]: experiment.fields
Out[19]: <VariableMap {u'brain_volume': u'0'}>

In [20]: experiment.fields['brain_volume']
Out[20]: u'0'

In [21]: experiment.fields['brain_volume'] = 42.0

In [22]: experiment.fields
Out[22]: <VariableMap {u'brain_volume': u'42.0'}>

In [27]: experiment.fields['brain_volume']
Out[27]: u'42.0'