
XNAT Python Client Documentation

Release 0.3.13

Hakim Achterberg

Feb 05, 2019

Contents

| | | |
|----------|--------------------------------------|-----------|
| 1 | XNAT Client Documentation | 3 |
| 1.1 | Introduction | 3 |
| 1.1.1 | Getting started | 3 |
| 1.1.2 | Credentials | 3 |
| 1.1.3 | Status | 4 |
| 1.2 | XNATpy Tutorial | 4 |
| 1.2.1 | XNAT REST API | 4 |
| 1.2.2 | Installation | 4 |
| 1.2.3 | Connecting to a server | 4 |
| 1.2.4 | Exploring your xnat server | 5 |
| 1.2.5 | Looping over data | 6 |
| 1.2.6 | Downloading data | 6 |
| 1.2.7 | Importing data into XNAT | 7 |
| 1.2.8 | Prearchive | 7 |
| 1.2.9 | Example scripts | 7 |
| 1.3 | XNAT scripting tutorial | 8 |
| 1.3.1 | Create a connection | 8 |
| 1.3.2 | Importing data into XNAT | 9 |
| 1.3.3 | Download data | 10 |
| 1.3.4 | Inspect DICOM tags | 11 |
| 1.3.5 | Custom variables | 11 |
| 1.4 | Code reference | 12 |
| 1.4.1 | xnat Package | 12 |
| 1.4.2 | session Module | 13 |
| 1.4.3 | inspect Module | 17 |
| 1.4.4 | prearchive Module | 18 |
| 1.4.5 | services Module | 20 |
| 1.4.6 | users Module | 21 |
| 2 | Indices and tables | 23 |
| | Python Module Index | 25 |

A new XNAT client that exposes XNAT objects/functions as python objects/functions.

Warning: This is NOT pyxnat, but a new module which is not as mature but uses a different philosophy for the user interface. Pyxnat is located at: <https://pythonhosted.org/pyxnat/>

The XNAT Python client is open-source (licensed under the Apache 2.0 license) and hosted on bitbucket at https://bitbucket.org/bigr_erasmusmc/xnatpy

To get yourself a copy:

```
hg clone https://<yourusername>@bitbucket.org/bigr_erasmusmc/xnatpy
```

or if you have a ssh key pair:

```
hg clone ssh://hg@bitbucket.org/bigr_erasmusmc/xnatpy
```

The official documentation can be found at xnat.readthedocs.org

1.1 Introduction

A new XNAT client that exposes XNAT objects/functions as python objects/functions.

1.1.1 Getting started

To install just use the setup.py normally:

```
python setup.py install
```

To get started, create a connection and start querying:

```
>>> import xnat
>>> session = xnat.connect('https://central.xnat.org', user="", password="")
>>> session.projects['Sample_DICOM'].subjects
>>> session.disconnect()
```

To see all options for creating connections see the `xnat.connect()`.

The `XNAT session` is the main class for interacting with XNAT. It contains the main communication functions.

When using IPython most functionality can be figured out by looking at the available attributes/methods of the returned objects.

1.1.2 Credentials

To store credentials this module uses the `.netrc` file. This file contains login information and should be accessible ONLY by the user (if not, the module will throw an error to let you know the file is unsafe).

1.1.3 Status

Currently we do not support the creation of Projects, Subjects, Experiments, etc via code. You can create resources and uploads files to them. Also it is possible to import data via the import service (upload a zip file). There is also support for working with the prearchive (reading, moving, deleting and archiving).

There is virtually no documentation or testing, this is a known limitation.

1.2 XNATpy Tutorial

1.2.1 XNAT 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 (on top of which this package as well as `pyxnat` is build)

To get an idea of how the XNAT REST API works it is helpful to visit the following URLs in your browser:

- <https://central.xnat.org/data/archive/projects>
- <https://central.xnat.org/data/archive/projects?format=xml>
- <https://central.xnat.org/data/archive/projects?format=json>

The first URL give you a table with an overview of all projects you can access on XNAT central. The second and third URL give the same information, but in different machine readable formats (XML and JSON respectively). This is extremely useful when creating scripts to automatically retrieve or store data from XNAT.

1.2.2 Installation

The easiest way to install xnat is via to python package index via pip:

```
pip install xnat
```

However, if you do not have pip or want to install from source just use the `setup.py` normally:

```
python setup.py install
```

1.2.3 Connecting to a server

To get started, create a connection:

```
>>> import xnat
>>> session = xnat.connect('https://central.xnat.org')
```

To see all options for creating connections see the `xnat.connect()`. The connection holds your login information, the server information and a session. It will also send a heartbeat every 14 minutes to keep the connection alive.

When working with a session it is always important to disconnect when done:


```
>>> session.disconnect()
```

Credentials

It is possible to pass your credentials for the session when connecting. This would look like:

```
>>> session = xnat.connect('http://my.xnat.server', user='admin', password='secret')
```

This would work and log in fine, but your password might be visible in your source code, command history or just on your screen. If you only give a user, but not a password xnatpy will prompt you for your password. This is fine for interactive use, but for automated scripts this is useless.

To store credentials this xnatpy uses the .netrc file. On linux the file is located in ~/.netrc. This file contains login information and should be accessible ONLY by the user (if not, the module will throw an error to let you know the file is unsafe). For example:

```
echo "machine images.xnat.org
> login admin
> password admin" > ~/.netrc
chmod 600 ~/.netrc
```

This will create the netrc file with the correct contents and set the permission correct.

Self-closing sessions

When in a script where there is a possibility for unforeseen errors it is safest to use a context operator in Python. This can be achieved by using the following:

```
>>> with xnat.connect('http://my.xnat.server') as session:
...     print session.projects
```

As soon as the scope of the with exists (even if because of an exception thrown!) the session will be disconnected automatically.

1.2.4 Exploring your xnat server

When a session is established, it is fairly easy to explore the data on the XNAT server. The data structure of XNAT is mimicked as Python objects. The connection gives access to a listing of all projects, subjects, and experiments on the server.

```
>>> import xnat
>>> session = xnat.connect('http://images.xnat.org', user='admin', password='admin')
[INFO] Found an 1.7 version (1.7.0)
[INFO] Retrieving schema from http://images.xnat.org/xapi/schemas/security
[INFO] Retrieving schema from http://images.xnat.org/xapi/schemas/birn/birnprov
[INFO] Retrieving schema from http://images.xnat.org/xapi/schemas/screening/
↪screeningAssessment
[INFO] Retrieving schema from http://images.xnat.org/xapi/schemas/catalog
[INFO] Retrieving schema from http://images.xnat.org/xapi/schemas/pipeline/workflow
[INFO] Retrieving schema from http://images.xnat.org/xapi/schemas/pipeline/build
[INFO] Retrieving schema from http://images.xnat.org/xapi/schemas/pipeline/repository
[INFO] Retrieving schema from http://images.xnat.org/xapi/schemas/validation/
↪protocolValidation
```

(continues on next page)

(continued from previous page)

```
[INFO] Retrieving schema from http://images.xnat.org/xapi/schemas/assessments
[INFO] Retrieving schema from http://images.xnat.org/xapi/schemas/xdat
[INFO] Retrieving schema from http://images.xnat.org/xapi/schemas/xdat/instance
[INFO] Retrieving schema from http://images.xnat.org/xapi/schemas/xdat/PlexiViewer
[INFO] Retrieving schema from http://images.xnat.org/xapi/schemas/xdat/display
[INFO] Retrieving schema from http://images.xnat.org/xapi/schemas/xnat
[INFO] Retrieving schema from http://images.xnat.org/xapi/schemas/project
>>> session.projects
<XNATListing (sandbox, sandbox project): <ProjectData sandbox project (sandbox)>>
```

The XNATListing is a special type of mapping in which you can access elements by a primary key (usually the *ID* or *Accession #*) and a secondary key (e.g. the label for a subject or experiment). Selection can be performed the same as a Python dict:

```
>>> sandbox_project = session.projects["sandbox"]
>>> sandbox_project.subjects
<XNATListing (XNAT_S00001, test001): <SubjectData test001 (XNAT_S00001)>>
```

You can browse the following levels on the XNAT server: projects, subjects, experiments, scans, resources, files. Also under experiments you have assessors which again can contain resources and files. This all following the same structure as XNAT.

Warning: Loading all subjects/experiments on a server can take very long if there is a lot of data. Going down through the project level is more efficient.

1.2.5 Looping over data

There are situations in which you want to perform an action for each subject or experiment. To do this, you can think of an XNATListing as a Python dict and most things will work naturally. For example:

```
>>> sandbox_project.subjects.keys()
[u'XNAT_S00001']
>>> sandbox_project.subjects.values()
[<SubjectData test001 (XNAT_S00001)>]
>>> len(sandbox_project.subjects)
1
>>> for subject in sandbox_project.subjects.values():
...     print(subject.label)
test001
```

1.2.6 Downloading data

The REST API allows for downloading of data from XNAT. The xnatpy package includes helper functions to make the downloading of data easier. For example, to download all experiments belonging to a subject:

```
>>> subject = sandbox_project.subjects['test001']
>>> subject.download_dir('./Downloads/test001')
```

This will download all the relevant experiments and unpack them in the target folder. Experiments, scans and resources can also be downloaded in a zip bundle using the `download_zip` method.

1.2.7 Importing data into XNAT

To add new data into XNAT it is possible to use the REST import service. It allows you to upload a zip file containing an experiment and XNAT will automatically try to store it in the correct place:

```
>>> session.services.import_('/path/to/archive.zip', project='sandbox', subject=
↳ 'test002')
```

Will upload the DICOM files in archive.zip and add them as scans under the subject *test002* in project *sandbox*. For more information on importing data see [import_](#)

1.2.8 Prearchive

When scans are sent to the XNAT they often end up in the prearchive pending review before adding them to the main archive. It is possible to view the prearchive via [xnatpy](#):

```
>>> session.prearchive.sessions()
[]
```

This gives a list of [PrearchiveSessions](#) in the archive. It is possible to archive, rebuild, more or remove the session using simple methods. For more information see [PrearchiveSession](#)

1.2.9 Example scripts

There is a number of example scripts located in the `examples` folder in the source code. The following code is a small command-line tool that prints all files for a given scan in the XNAT archive:

```
#!/usr/bin/env python

import xnat
import argparse
import re

def get_files(connection, project, subject, session, scan):
    xnat_project = connection.projects[project]
    xnat_subject = xnat_project.subjects[subject]
    xnat_experiment = xnat_subject.experiments[session]
    xnat_scan = xnat_experiment.scans[scan]
    files = xnat_scan.files.values()
    return files

def filter_files(xnat_files, regex):
    filtered_files = []
    regex = re.compile(regex)
    for file in xnat_files:
        found = regex.match(file.name)
        if found:
            filtered_files.append(file)
    return filtered_files

def main():
    parser = argparse.ArgumentParser(description='Prints all files from a certain_
↳ scan.')
```

(continues on next page)

(continued from previous page)

```

    parser.add_argument('--xnathost', type=unicode, required=True, help='xnat host_
↪name')
    parser.add_argument('--project', type=unicode, required=True, help='Project id')
    parser.add_argument('--subject', type=unicode, required=True, help='subject')
    parser.add_argument('--session', type=unicode, required=True, help='session')
    parser.add_argument('--scan', type=unicode, required=True, help='scan')
    parser.add_argument('--filter', type=unicode, required=False, default='.*', help=
↪'regex filter for file names')
    args = parser.parse_args()

    with xnat.connect(args.xnathost) as connection:
        xnat_files = get_files(connection, args.project, args.subject, args.session,
↪args.scan)
        xnat_files = filter_files(xnat_files, args.filter)
        for file in xnat_files:
            print('{}'.format(file.name))

if __name__ == '__main__':
    main()

```

1.3 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`.

1.3.1 Create a connection

Start up `ipython` and create a connection, it will prompt you to enter the password for the user test:

```

>> ipython
Python 2.7.12+ (default, Sep 1 2016, 20:27:38)
Type "copyright", "credits" or "license" for more information.

IPython 2.4.1 -- An enhanced Interactive Python.
?          -> Introduction and overview of IPython's features.
%quickref  -> Quick reference.
help       -> Python's own help system.
object?    -> Details about 'object', use 'object??' for extra details.

In [1]: import xnat

```

(continues on next page)

(continued from previous page)

```
In [2]: session = xnat.connect('http://145.100.58.186/xnat', user='test')
Please enter the password for user 'test':
[INFO] Found an 1.6 version (1.6.4)
[INFO] Retrieving schema from http://145.100.58.186/xnat/schemas/xnat/xnat.xsd
[INFO] Found additional schemas: ['http://145.100.58.186/xnat/schemas/pipeline/
↪workflow.xsd', 'http://145.100.58.186/xnat/schemas/catalog/catalog.xsd', 'http://
↪145.100.58.186/xnat/schemas/pipeline/repository.xsd', 'http://145.100.58.186/xnat/
↪schemas/screening/screeningAssessment.xsd', 'http://145.100.58.186/xnat/schemas/
↪project/project.xsd', 'http://145.100.58.186/xnat/schemas/validation/
↪protocolValidation.xsd', 'http://145.100.58.186/xnat/schemas/assessments/
↪assessments.xsd', 'http://145.100.58.186/xnat/schemas/birn/birnprov.xsd', 'http://
↪145.100.58.186/xnat/schemas/security/security.xsd']
[INFO] Retrieving schema from http://145.100.58.186/xnat/schemas/pipeline/workflow.xsd
[INFO] Retrieving schema from http://145.100.58.186/xnat/schemas/catalog/catalog.xsd
[INFO] Retrieving schema from http://145.100.58.186/xnat/schemas/pipeline/repository.
↪xsd
[ERROR] Could not parse schema from http://145.100.58.186/xnat/schemas/pipeline/
↪repository.xsd, not valid XML found
[INFO] Retrieving schema from http://145.100.58.186/xnat/schemas/screening/
↪screeningAssessment.xsd
[INFO] Retrieving schema from http://145.100.58.186/xnat/schemas/project/project.xsd
[INFO] Retrieving schema from http://145.100.58.186/xnat/schemas/validation/
↪protocolValidation.xsd
[INFO] Retrieving schema from http://145.100.58.186/xnat/schemas/assessments/
↪assessments.xsd
[INFO] Retrieving schema from http://145.100.58.186/xnat/schemas/birn/birnprov.xsd
[INFO] Retrieving schema from http://145.100.58.186/xnat/schemas/security/security.xsd
```

Once a connection is established it is possible to browse the projects. This can be achieved by simply looking at the projects attribute of the session:

```
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 either the id or name of the project:

```
In [4]: project = session.projects['brainimages']
```

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

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

1.3.2 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 a 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_('/home/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_('/home/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:

```
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.

1.3.3 Download data

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

```
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:

```
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/')
Downloading 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
/home/hachterberg/temp/ANONYMIZ3/ANONYMIZ3/scans/1001-MR2-FLAIR
/home/hachterberg/temp/ANONYMIZ3/ANONYMIZ3/scans/1001-MR2-FLAIR/resources
/home/hachterberg/temp/ANONYMIZ3/ANONYMIZ3/scans/1001-MR2-FLAIR/resources/DICOM
/home/hachterberg/temp/ANONYMIZ3/ANONYMIZ3/scans/1001-MR2-FLAIR/resources/DICOM/files
/home/hachterberg/temp/ANONYMIZ3/ANONYMIZ3/scans/1001-MR2-FLAIR/resources/DICOM/files/
↳IM2.dcm
/home/hachterberg/temp/ANONYMIZ3/ANONYMIZ3/scans/1001-MR2-FLAIR/resources/DICOM/files/
↳IM32.dcm
/home/hachterberg/temp/ANONYMIZ3/ANONYMIZ3/scans/1001-MR2-FLAIR/resources/DICOM/files/
↳IM11.dcm
...
```

1.3.4 Inspect DICOM tags

You can retrieve the dicom tags of a scan, in both the archive and the prearchive, using the `dicom_dump` method of a `Scan` or `PrearchiveScan` object.:

```
In [10]: experiment.scans['T1'].dicom_dump()
```

You can also filter on DICOM tags using the `fields` argument:

```
In [11]: experiment.scans['T1'].dicom_dump(fields="PatientID")
In [12]: experiment.scans['T1'].dicom_dump(fields=["PatientID", "PatientName"])
```

1.3.5 Custom variables

The custom variables are exposed as a dict-like object in `xnatpy`. They are located in the `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']
```

(continues on next page)

(continued from previous page)

```
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'
```

1.4 Code reference

1.4.1 xnat Package

This package contains the entire client. The connect function is the only function actually in the package. All following classes are created based on the <https://central.xnat.org/schema/xnat/xnat.xsd> schema and the xnatcore and xnatbase modules, using the convert_xsd.

`xnat.connect` (*server*, *user=None*, *password=None*, *verify=True*, *netrc_file=None*, *debug=False*, *extension_types=True*, *loglevel=None*, *logger=None*, *detect_redirect=True*)

Connect to a server and generate the correct classed based on the servers xnat.xsd This function returns an object that can be used as a context operator. It will call disconnect automatically when the context is left. If it is used as a function, then the user should call `.disconnect()` to destroy the session and temporary code file.

Parameters

- **server** (*str*) – uri of the server to connect to (including <http://> or <https://>)
- **user** (*str*) – username to use, leave empty to use netrc entry or anonymous login.
- **password** (*str*) – password to use with the username, leave empty when using netrc. If a username is given and no password, there will be a prompt on the console requesting the password.
- **verify** (*bool*) – verify the https certificates, if this is false the connection will be encrypted with ssl, but the certificates are not checked. This is potentially dangerous, but required for self-signed certificates.
- **netrc_file** (*str*) – alternative location to use for the netrc file (path pointing to a file following the netrc syntax)
- **bool** (*debug*) – Set debug information printing on and print extra debug information. This is meant for xnatpy developers and not for normal users. If you want to debug your code using xnatpy, just set the loglevel to DEBUG which will show you all requests being made, but spare you the xnatpy internals.
- **loglevel** (*str*) – Set the level of the logger to desired level
- **logger** (*logging.Logger*) – A logger to reuse instead of creating an own logger
- **detect_redirect** (*bool*) – Try to detect a redirect (via a 302 response) and short-cut for subsequent requests

Returns XNAT session object

Return type *XNATSession*

Preferred use:


```
>>> import xnat
>>> with xnat.connect('https://central.xnat.org') as session:
...     subjects = session.projects['Sample_DICOM'].subjects
...     print('Subjects in the SampleDICOM project: {}'.format(subjects))
Subjects in the SampleDICOM project: <XNATListing (CENTRAL_S01894, dcmtest1):
↳<SubjectData CENTRAL_S01894>, (CENTRAL_S00461, PACE_HF_SUPINE): <SubjectData_
↳CENTRAL_S00461>>
```

Alternative use:

```
>>> import xnat
>>> session = xnat.connect('https://central.xnat.org')
>>> subjects = session.projects['Sample_DICOM'].subjects
>>> print('Subjects in the SampleDICOM project: {}'.format(subjects))
Subjects in the SampleDICOM project: <XNATListing (CENTRAL_S01894, dcmtest1):
↳<SubjectData CENTRAL_S01894>, (CENTRAL_S00461, PACE_HF_SUPINE): <SubjectData_
↳CENTRAL_S00461>>
>>> session.disconnect()
```

1.4.2 session Module

class `xnat.session.XNATSession` (*server, logger, interface=None, user=None, password=None, keepalive=840, debug=False, original_uri=None*)

Bases: `object`

The main XNATSession session class. It keeps a connection to XNATSession alive and manages the main communication to XNATSession. To keep the connection alive there is a background thread that sends a heart-beat to avoid a time-out.

The main starting points for working with the XNATSession server are:

- `XNATSession.projects`
- `XNATSession.subjects`
- `XNATSession.experiments`
- `XNATSession.prearchive`
- `XNATSession.services`
- `XNATSession.users`

Note: Some methods create listing that are using the `xnat.XNATListing` class. They allow for indexing with both XNATSession ID and a secondary key (often the label). Also they support basic filtering and tabulation.

There are also methods for more low level communication. The main methods are `XNATSession.get`, `XNATSession.post`, `XNATSession.put`, and `XNATSession.delete`. The methods do not query URIs but instead query XNATSession REST paths as described in the [XNATSession 1.6 REST API Directory](#).

For an even lower level interfaces, the `XNATSession.interface` gives access to the underlying [requests](#) interface. This interface has the user credentials and benefits from the keep alive of this class.

Note: XNATSession Objects have a client-side cache. This is for efficiency, but might cause problems if the

server is being changed by a different client. It is possible to clear the current cache using `XNATSession.clearcache`. Turning off caching complete can be done by setting `XNATSession.caching`.

Warning: You should NOT try use this class directly, it should only be created by `xnat.connect`.

clearcache()

Clear the cache of the listings in the Session object

delete (*path*, *headers=None*, *accepted_status=None*, *query=None*, *timeout=None*)

Delete the content of a given REST directory.

Parameters

- **path** (*str*) – the path of the uri to retrieve (e.g. “/data/archive/projects”) the remained for the uri is constructed automatically
- **format** (*str*) – the format of the request, this will add the format= to the query string
- **query** (*dict*) – the values to be added to the query string in the uri
- **accepted_status** (*list*) – a list of the valid values for the return code, default [200]
- **timeout** (*float or tuple*) – timeout in seconds, float or (connection timeout, read timeout)

Returns the requests reponse

Return type requests.Response

download (*uri*, *target*, *format=None*, *verbose=True*, *timeout=None*)

Download uri to a target file

download_stream (*uri*, *target_stream*, *format=None*, *verbose=False*, *chunk_size=524288*, *update_func=None*, *timeout=None*)

Download the given uri to the given target_stream.

Parameters

- **uri** (*str*) – Path of the uri to retrieve.
- **target_stream** (*file*) – A writable file-like object to save the stream to.
- **format** (*str*) – Request format
- **verbose** (*bool*) – If True, and an update_func is not specified, a progress bar is shown on stdout.
- **chunk_size** (*int*) – Download this many bytes at a time
- **update_func** (*func*) – If provided, will be called every chunk_size bytes. Must accept three parameters:
 - the number of bytes downloaded so far
 - the total number of byte to be downloaded (might be None),
 - A boolean flag which is False during the download, and True when the download has completed (or failed)
- **timeout** (*float or tuple*) – timeout in seconds, float or (connection timeout, read timeout)

download_zip (*uri*, *target*, *verbose=True*, *timeout=None*)

Download uri to a target zip file

experiments

Listing of all experiments on the XNAT server

Cached using the caching decorator

get (*path*, *format=None*, *query=None*, *accepted_status=None*, *timeout=None*)

Retrieve the content of a given REST directory.

Parameters

- **path** (*str*) – the path of the uri to retrieve (e.g. “/data/archive/projects”) the remained for the uri is constructed automatically
- **format** (*str*) – the format of the request, this will add the format= to the query string
- **query** (*dict*) – the values to be added to the query string in the uri
- **accepted_status** (*list*) – a list of the valid values for the return code, default [200]
- **timeout** (*float or tuple*) – timeout in seconds, float or (connection timeout, read timeout)

Returns the requests reponse

Return type requests.Response

get_json (*uri*, *query=None*, *accepted_status=None*)

Helper function that perform a GET, but sets the format to JSON and parses the result as JSON

Parameters

- **uri** (*str*) – the path of the uri to retrieve (e.g. “/data/archive/projects”) the remained for the uri is constructed automatically
- **query** (*dict*) – the values to be added to the query string in the uri

head (*path*, *accepted_status=None*, *allow_redirects=False*, *timeout=None*)

Retrieve the header for a http request of a given REST directory.

Parameters

- **path** (*str*) – the path of the uri to retrieve (e.g. “/data/archive/projects”) the remained for the uri is constructed automatically
- **accepted_status** (*list*) – a list of the valid values for the return code, default [200]
- **allow_redirects** (*bool*) – allow you request to be redirected
- **timeout** (*float or tuple*) – timeout in seconds, float or (connection timeout, read timeout)

Returns the requests reponse

Return type requests.Response

interface

The underlying [requests](#) interface used.

post (*path*, *data=None*, *json=None*, *format=None*, *query=None*, *accepted_status=None*, *timeout=None*)

Post data to a given REST directory.

Parameters

- **path** (*str*) – the path of the uri to retrieve (e.g. “/data/archive/projects”) the remained for the uri is constructed automatically
- **data** – Dictionary, bytes, or file-like object to send in the body of the Request.
- **json** – json data to send in the body of the Request.
- **format** (*str*) – the format of the request, this will add the format= to the query string
- **query** (*dict*) – the values to be added to the query string in the uri
- **accepted_status** (*list*) – a list of the valid values for the return code, default [200, 201]
- **timeout** (*float or tuple*) – timeout in seconds, float or (connection timeout, read timeout)

Returns the requests reponse

Return type requests.Response

prearchive

Representation of the prearchive on the XNAT server, see [*xnat.prearchive*](#)

projects

Listing of all projects on the XNAT server

Cached using the caching decorator

put (*path, data=None, files=None, json=None, format=None, query=None, accepted_status=None, timeout=None*)

Put the content of a given REST directory.

Parameters

- **path** (*str*) – the path of the uri to retrieve (e.g. “/data/archive/projects”) the remained for the uri is constructed automatically
- **data** – Dictionary, bytes, or file-like object to send in the body of the Request.
- **json** – json data to send in the body of the Request.
- **files** – Dictionary of 'name': file-like-objects (or {'name': file-tuple}) for multipart encoding upload. file-tuple can be a 2-tuple ('filename', fileobj), 3-tuple ('filename', fileobj, 'content_type') or a 4-tuple ('filename', fileobj, 'content_type', custom_headers), where 'content-type' is a string defining the content type of the given file and custom_headers a dict-like object containing additional headers to add for the file.
- **format** (*str*) – the format of the request, this will add the format= to the query string
- **query** (*dict*) – the values to be added to the query string in the uri
- **accepted_status** (*list*) – a list of the valid values for the return code, default [200, 201]
- **timeout** (*float or tuple*) – timeout in seconds, float or (connection timeout, read timeout)

Returns the requests reponse

Return type requests.Response

scan_types

A list of scan types associated with this XNATSession instance

scanners

A list of scanners referenced in XNATSession

services

Collection of services, see `xnat.services`

subjects

Listing of all subjects on the XNAT server

Cached using the caching decorator

upload (*uri, file_, retries=1, query=None, content_type=None, method=u'put', overwrite=False, timeout=None*)

Upload data or a file to XNAT

Parameters

- **uri** (*str*) – uri to upload to
- **file** – the file handle, path to a file or a string of data (which should not be the path to an existing file!)
- **retries** (*int*) – amount of times xnatpy should retry in case of failure
- **query** (*dict*) – extra query string content
- **content_type** – the content type of the file, if not given it will default to application/octet-stream
- **method** (*str*) – either put (default) or post
- **overwrite** (*bool*) – indicate if previous data should be overwritten
- **timeout** (*float or tuple*) – timeout in seconds, float or (connection timeout, read timeout)

Returns**users**

Representation of the users registered on the XNAT server

xnat_version

The version of the XNAT server

Cached using the caching decorator

`xnat.session.default_update_func` (*total*)

Set up a default update function to be used by the `Session.download_stream` method. This function configures a `progressbar.ProgressBar` object which displays progress as a file is downloaded.

Parameters **total** (*int*) – Total number of bytes to be downloaded (might be None)

Returns A function to be used as the `update_func` by the `Session.download_stream` method.

1.4.3 inspect Module

class `xnat.inspect.Inspect` (*xnat_session*)

Bases: `object`

datafields (*datatype, pattern=u'*, prepend_type=True*)

datatypes (*pattern=u'*, fields_pattern=None*)

xnat_session

1.4.4 prearchive Module

class `xnat.prearchive.Prearchive` (*xnat_session*)

Bases: `object`

sessions (*project=None*)

Get the session in the prearchive, optionally filtered by project. This function is not cached and returns the results of a query at each call.

Parameters `project` (*str*) – the project to filter on

Returns list of prearchive session found

Return type list

xnat_session

class `xnat.prearchive.PrearchiveFile` (*uri, xnat_session, id_=None, datafields=None, parent=None, fieldname=None*)

Bases: `xnat.core.XNATBaseObject`

data

download (*path*)

Download the file

Parameters `path` (*str*) – the path to download to

Returns the path of the downloaded file

Return type *str*

fulldata

name

The name of the file

size

The size of the file

xpath

class `xnat.prearchive.PrearchiveScan` (*uri, xnat_session, id_=None, datafields=None, parent=None, fieldname=None*)

Bases: `xnat.core.XNATBaseObject`

data

dicom_dump (*fields=None*)

Retrieve a dicom dump as a JSON data structure See the XAPI documentation for more detailed information: [DICOM Dump Service](#)

Parameters `fields` (*list*) – Fields to filter for DICOM tags. It can either a tag name or tag number in the format GGGGEEEE (G = Group number, E = Element number)

Returns JSON object (dict) representation of DICOM header

Return type *dict*

download (*path*)

Download the scan as a zip

Parameters `path` (*str*) – the path to download to

Returns the path of the downloaded file

Return type *str*

files

List of files contained in the scan

fulldata

series_description

The series description of the scan

xpath

```
class xnat.prearchive.PrearchiveSession(uri=None, xnat_session=None, id_=None,
                                         datafields=None, parent=None, fieldname=None,
                                         overwrites=None, **kwargs)
```

Bases: `xnat.core.XNATBaseObject`

```
archive(overwrite=None, quarantine=None, trigger_pipelines=None, project=None, subject=None,
        experiment=None)
```

Method to archive this prearchive session to the main archive

Parameters

- **overwrite** (*str*) – how the handle existing data (none, append, delete)
- **quarantine** (*bool*) – flag to indicate session should be quarantined
- **trigger_pipelines** (*bool*) – indicate that archiving should trigger pipelines
- **project** (*str*) – the project in the archive to assign the session to
- **subject** (*str*) – the subject in the archive to assign the session to
- **experiment** (*str*) – the experiment in the archive to assign the session content to

Returns the newly created experiment

Return type ExperimentData

autoarchive

data

```
delete(asynchronous=None)
```

Delete the session from the prearchive

Parameters **asynchronous** (*bool*) – flag to delete asynchronously

Returns requests response

```
download(path)
```

Method to download the zip of the prearchive session

Parameters **path** (*str*) – path to download to

Returns path of the downloaded zip file

Return type *str*

folder_name

fulldata

id

A unique ID for the session in the prearchive :return:

label

lastmod

move (*new_project*, *asynchronous=None*)

Move the session to a different project in the prearchive

Parameters

- **new_project** (*str*) – the id of the project to move to
- **asynchronous** (*bool*) – flag to move asynchronously

Returns requests response

name

prevent_anon

prevent_auto_commit

project

rebuild (*asynchronous=None*)

Rebuilt the session in the prearchive

Parameters **asynchronous** (*bool*) – flag to rebuild asynchronously

Returns requests response

scan_date

scan_time

scans

List of scans in the prearchive session

status

subject

tag

timestamp

uploaded

Datetime when the session was uploaded

xpath

1.4.5 services Module

class `xnat.services.Services` (*xnat_session*)

Bases: `object`

The class representing all service functions in XNAT found in the /data/services REST directory

dicom_dump (*src*, *fields=None*)

Retrieve a dicom dump as a JSON data structure See the XAPI documentation for more detailed information: [DICOM Dump Service](#)

Parameters **fields** (*lst*) – Fields to filter for DICOM tags. It can either a tag name or tag number in the format GGGGEEEE (G = Group number, E = Element number)

Returns JSON object (dict) representation of DICOM header

Return type `dict`

import_ (*path*, *overwrite=None*, *quarantine=False*, *destination=None*, *trigger_pipelines=None*, *project=None*, *subject=None*, *experiment=None*, *content_type=None*)
 Import a file into XNAT using the import service. See the [XNAT wiki](#) for a detailed explanation.

Parameters

- **path** (*str*) – local path of the file to upload and import
- **overwrite** (*str*) – how the handle existing data (none, append, delete)
- **quarantine** (*bool*) – flag to indicate session should be quarantined
- **trigger_pipelines** (*bool*) – indicate that archiving should trigger pipelines
- **destination** (*str*) – the destination to upload the scan to
- **project** (*str*) – the project in the archive to assign the session to
- **subject** (*str*) – the subject in the archive to assign the session to
- **experiment** (*str*) – the experiment in the archive to assign the session content to
- **content_type** (*str*) – overwrite the content_type (by the mimetype will be guessed)

Returns

issue_token (*user=None*)

Issue a login token, by default for the current logged in user. If username is given, for that user. To issue tokens for other users you must be an admin.

Parameters **user** (*str*) – User to issue token for, default is current user

Returns Token in a named tuple (alias, secret)

xnat_session

class `xnat.services.TokenResult` (*alias, secret*)

Bases: `tuple`

alias

Alias for field number 0

secret

Alias for field number 1

1.4.6 users Module

class `xnat.users.User` (*data*)

Bases: `object`

Representation of a user on the connected XNAT system

data

email

The email of the user

first_name

The first name of the user

id

The id of the user

last_name

The last name of the user

login

The login name of the user

class `xnat.users.Users` (*xnat_session*)

Bases: `_abcoll.Mapping`

Listing of the users on the connected XNAT installation

data

Cached using the caching decorator

xnat_session

CHAPTER 2

Indices and tables

- `genindex`
- `modindex`
- `search`

X

- `xnat`, [12](#)
- `xnat.inspect`, [17](#)
- `xnat.prearchive`, [18](#)
- `xnat.services`, [20](#)
- `xnat.session`, [13](#)
- `xnat.users`, [21](#)

A

alias (xnat.services.TokenResult attribute), 21
archive() (xnat.prearchive.PrearchiveSession method), 19
autoarchive (xnat.prearchive.PrearchiveSession attribute), 19

C

clearcache() (xnat.session.XNATSession method), 14
connect() (in module xnat), 12

D

data (xnat.prearchive.PrearchiveFile attribute), 18
data (xnat.prearchive.PrearchiveScan attribute), 18
data (xnat.prearchive.PrearchiveSession attribute), 19
data (xnat.users.User attribute), 21
data (xnat.users.Users attribute), 22
datafields() (xnat.inspect.Inspect method), 17
datatypes() (xnat.inspect.Inspect method), 17
default_update_func() (in module xnat.session), 17
delete() (xnat.prearchive.PrearchiveSession method), 19
delete() (xnat.session.XNATSession method), 14
dicom_dump() (xnat.prearchive.PrearchiveScan method), 18
dicom_dump() (xnat.services.Services method), 20
download() (xnat.prearchive.PrearchiveFile method), 18
download() (xnat.prearchive.PrearchiveScan method), 18
download() (xnat.prearchive.PrearchiveSession method), 19
download() (xnat.session.XNATSession method), 14
download_stream() (xnat.session.XNATSession method), 14
download_zip() (xnat.session.XNATSession method), 14

E

email (xnat.users.User attribute), 21
experiments (xnat.session.XNATSession attribute), 15

F

files (xnat.prearchive.PrearchiveScan attribute), 18

first_name (xnat.users.User attribute), 21
folder_name (xnat.prearchive.PrearchiveSession attribute), 19
fulldata (xnat.prearchive.PrearchiveFile attribute), 18
fulldata (xnat.prearchive.PrearchiveScan attribute), 19
fulldata (xnat.prearchive.PrearchiveSession attribute), 19

G

get() (xnat.session.XNATSession method), 15
get_json() (xnat.session.XNATSession method), 15

H

head() (xnat.session.XNATSession method), 15

I

id (xnat.prearchive.PrearchiveSession attribute), 19
id (xnat.users.User attribute), 21
import_() (xnat.services.Services method), 20
Inspect (class in xnat.inspect), 17
interface (xnat.session.XNATSession attribute), 15
issue_token() (xnat.services.Services method), 21

L

label (xnat.prearchive.PrearchiveSession attribute), 19
last_name (xnat.users.User attribute), 21
lastmod (xnat.prearchive.PrearchiveSession attribute), 19
login (xnat.users.User attribute), 21

M

move() (xnat.prearchive.PrearchiveSession method), 19

N

name (xnat.prearchive.PrearchiveFile attribute), 18
name (xnat.prearchive.PrearchiveSession attribute), 20

P

post() (xnat.session.XNATSession method), 15
Prearchive (class in xnat.prearchive), 18
prearchive (xnat.session.XNATSession attribute), 16

PrearchiveFile (class in xnat.prearchive), 18
PrearchiveScan (class in xnat.prearchive), 18
PrearchiveSession (class in xnat.prearchive), 19
prevent_anon (xnat.prearchive.PrearchiveSession attribute), 20
prevent_auto_commit (xnat.prearchive.PrearchiveSession attribute), 20
project (xnat.prearchive.PrearchiveSession attribute), 20
projects (xnat.session.XNATSession attribute), 16
put() (xnat.session.XNATSession method), 16
xnat.users (module), 21
xnat_session (xnat.inspect.Inspect attribute), 17
xnat_session (xnat.prearchive.Prearchive attribute), 18
xnat_session (xnat.services.Services attribute), 21
xnat_session (xnat.users.Users attribute), 22
xnat_version (xnat.session.XNATSession attribute), 17
XNATSession (class in xnat.session), 13
xpath (xnat.prearchive.PrearchiveFile attribute), 18
xpath (xnat.prearchive.PrearchiveScan attribute), 19
xpath (xnat.prearchive.PrearchiveSession attribute), 20

R

rebuild() (xnat.prearchive.PrearchiveSession method), 20

S

scan_date (xnat.prearchive.PrearchiveSession attribute), 20
scan_time (xnat.prearchive.PrearchiveSession attribute), 20
scan_types (xnat.session.XNATSession attribute), 16
scanners (xnat.session.XNATSession attribute), 16
scans (xnat.prearchive.PrearchiveSession attribute), 20
secret (xnat.services.TokenResult attribute), 21
series_description (xnat.prearchive.PrearchiveScan attribute), 19
Services (class in xnat.services), 20
services (xnat.session.XNATSession attribute), 17
sessions() (xnat.prearchive.Prearchive method), 18
size (xnat.prearchive.PrearchiveFile attribute), 18
status (xnat.prearchive.PrearchiveSession attribute), 20
subject (xnat.prearchive.PrearchiveSession attribute), 20
subjects (xnat.session.XNATSession attribute), 17

T

tag (xnat.prearchive.PrearchiveSession attribute), 20
timestamp (xnat.prearchive.PrearchiveSession attribute), 20
TokenResult (class in xnat.services), 21

U

upload() (xnat.session.XNATSession method), 17
uploaded (xnat.prearchive.PrearchiveSession attribute), 20
User (class in xnat.users), 21
Users (class in xnat.users), 22
users (xnat.session.XNATSession attribute), 17

X

xnat (module), 12
xnat.inspect (module), 17
xnat.prearchive (module), 18
xnat.services (module), 20
xnat.session (module), 13