pyxnat

A Python interface for XNAT

30th June 2010

Yannick SCHWARTZ

yannick.schwartz@cea.fr
yannick.schwartz@gmail.com

Neurospin

Plan

- What and why?
- Getting started
- Gathering knowledge
- Reading data from XNAT
- Processing some data
- Writing some data on XNAT
- Work in progress

pyxnat?

- pyxnat is a Python module on top of the REST API
- Why a library on top of the REST API?
  - Takes care of all the responses parsing
  - Provides helper functions
  - Implements optimizations
  - Adds features (e.g. cache)
- Why Python?
  - We use Python at our lab
  - Growing neuroimaging Python community

Getting started

pyxnat 0.5 is the available version

http://packages.python.org/pyxnat/

- Installation
- Documentation

pyxnat 0.6 is the version in the presentation

Setup the connection

```
>>> from pyxnat import Interface
>>> imagen = Interface(server='https://imagen.cea.fr/imagen_database',
                       user='login', password='pass', cachedir='/tmp')
```

Sub interfaces

Data selection
Search operations
Server introspection

```
>>> imagen.select(...)
>>> imagen.search(...)
>>> imagen.inspect(...)
```

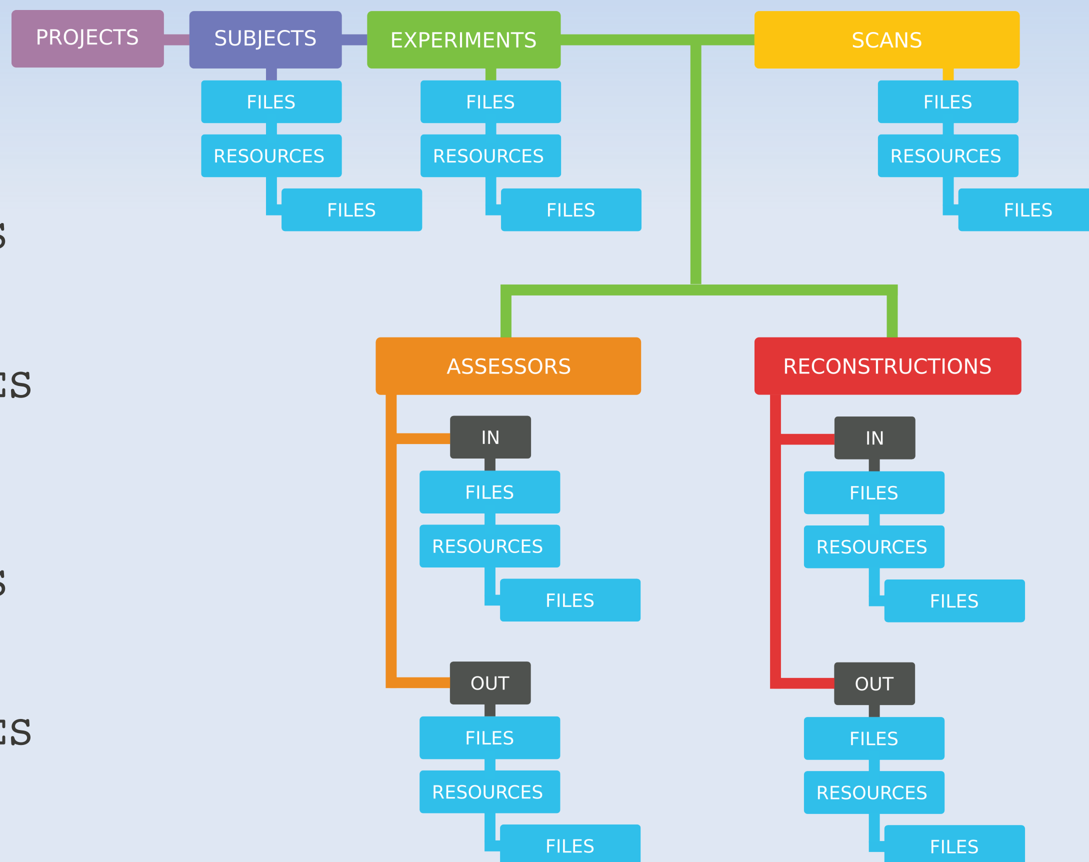
Gathering knowledge

- Prior knowledge is required to interact with the REST API
  - REST hierarchy and functions
  - Schema types and fields
  - Values of the various fields and REST resources for a project
- First time browsing a database using REST
  - XNAT wiki on REST
  - Web interface
  - Exploring the REST hierarchy to discover available resources
  - Iterating over a few subjects to make sure it is consistent

REST hierarchy

```
>>> imagen.inspect.rest_hierarchy()

- PROJECTS
        + SUBJECTS
            + EXPERIMENTS
                + ASSESSORS
                    + IN_FILES
                    + IN_RESOURCES
                        + FILES
                    + OUT_FILES
                    + OUT_RESOURCES
                        + FILES
                + RECONSTRUCTIONS
                    + IN_FILES
                    + IN_RESOURCES
                        + FILES
                    + OUT_FILES
                    + OUT_RESOURCES
                        + FILES
                + SCANS
                    + IN_FILES
                    + IN_RESOURCES
```



Schema types

Datatypes

```
>>> imagen.inspect.datatypes()
    ['psytool:ctsData','imagen:rawPackageData','spm:wamaskData',
     'psytool:niData','imagen:recruitmentInfosData',spm:wmeanaData', ...]
```

GET /REST/search/elements?format=json

Datafields

```
>>> imagen.inspect.datatypes('psytool:dotprobeData', '*')
    [...,
     'psytool:dotprobeData/TS_1','psytool:dotprobeData/TS_2',
     'psytool:dotprobeData/TS_3','psytool:dotprobeData/TS_4',
     ...]
```

GET /REST/search/elements/psytool:dotprobeData?format=json

Field values

```
>>> imagen.inspect.fieldvalues('psytool:dotprobeData/TS_1')
    ['', '1', '3', '2', '4', '7', '6', '8']
```

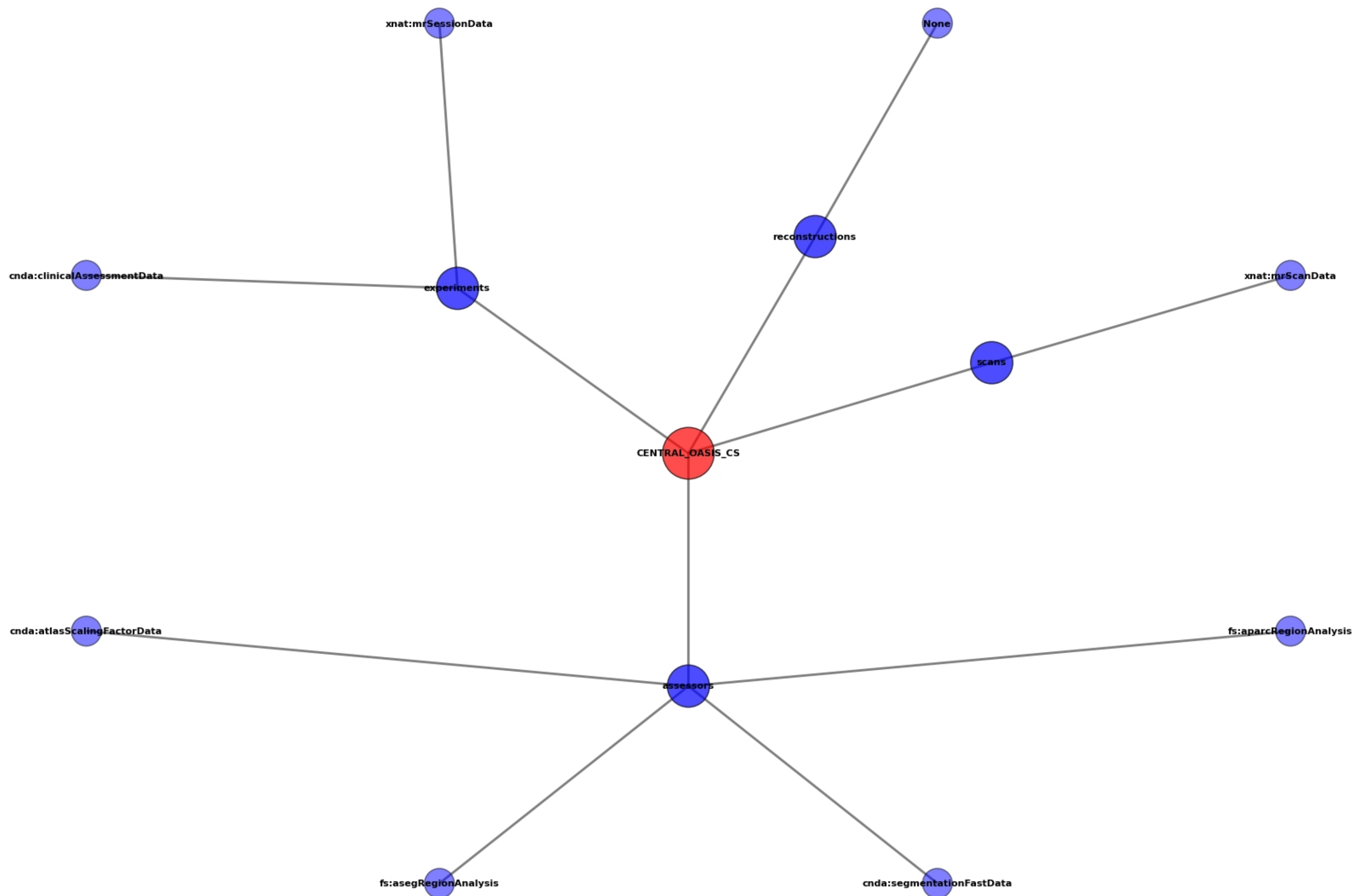
REST hierarchy & schema types

- There is no way to have a direct mapping between REST resources and datatypes.
- The mapping can be found by exploring a project with the **inspect** method.

```
>>>imagen.inspect(project='IMAGEN', subjects_nb=2)
```

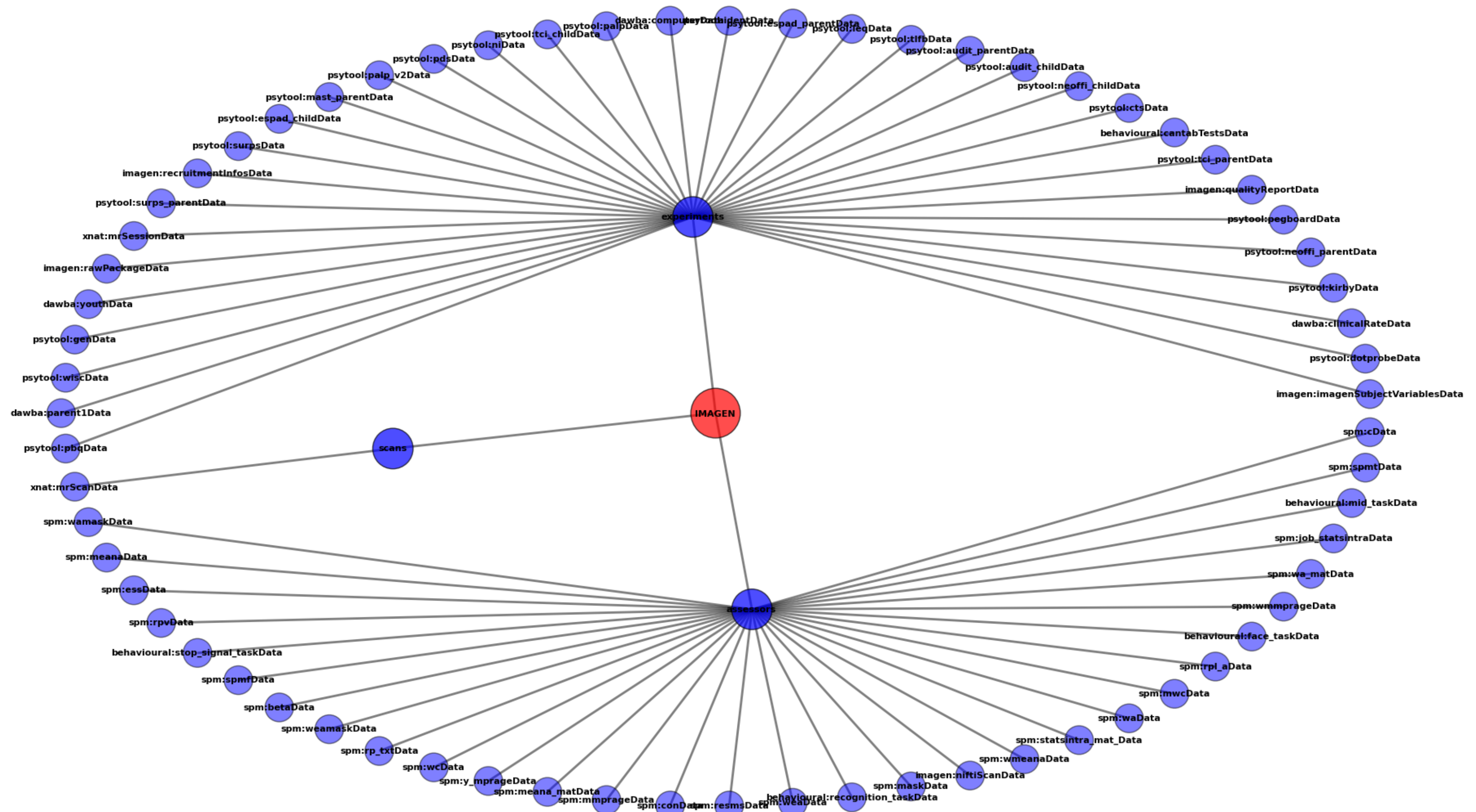
```
>>> imagen.inspect.rest_hierarchy()
- PROJECTS
    + SUBJECTS
        + EXPERIMENTS
          -----------
        - psytool:tci_childData
        - psytool:pbqData
        - dawba:computerData
        - psytool:palp_v2Data
        - psytool:tlfbData
        - xnat:mrSessionData
        - psytool:surpsData
        - psytool:niData
        - psytool:ctsData
        - psytool:kirbyData
        - psytool:tci_parentData
        - psytool:audit_childData
            + ASSESSORS
              ---------
            - spm:rp_txtData
            - spm:spmfData
            - spm:wamaskData
            - imagen:niftiScanData
            - spm:rpvData
```

REST hierarchy & schema types



REST hierarchy & schema types

IMAGEN



Naming conventions

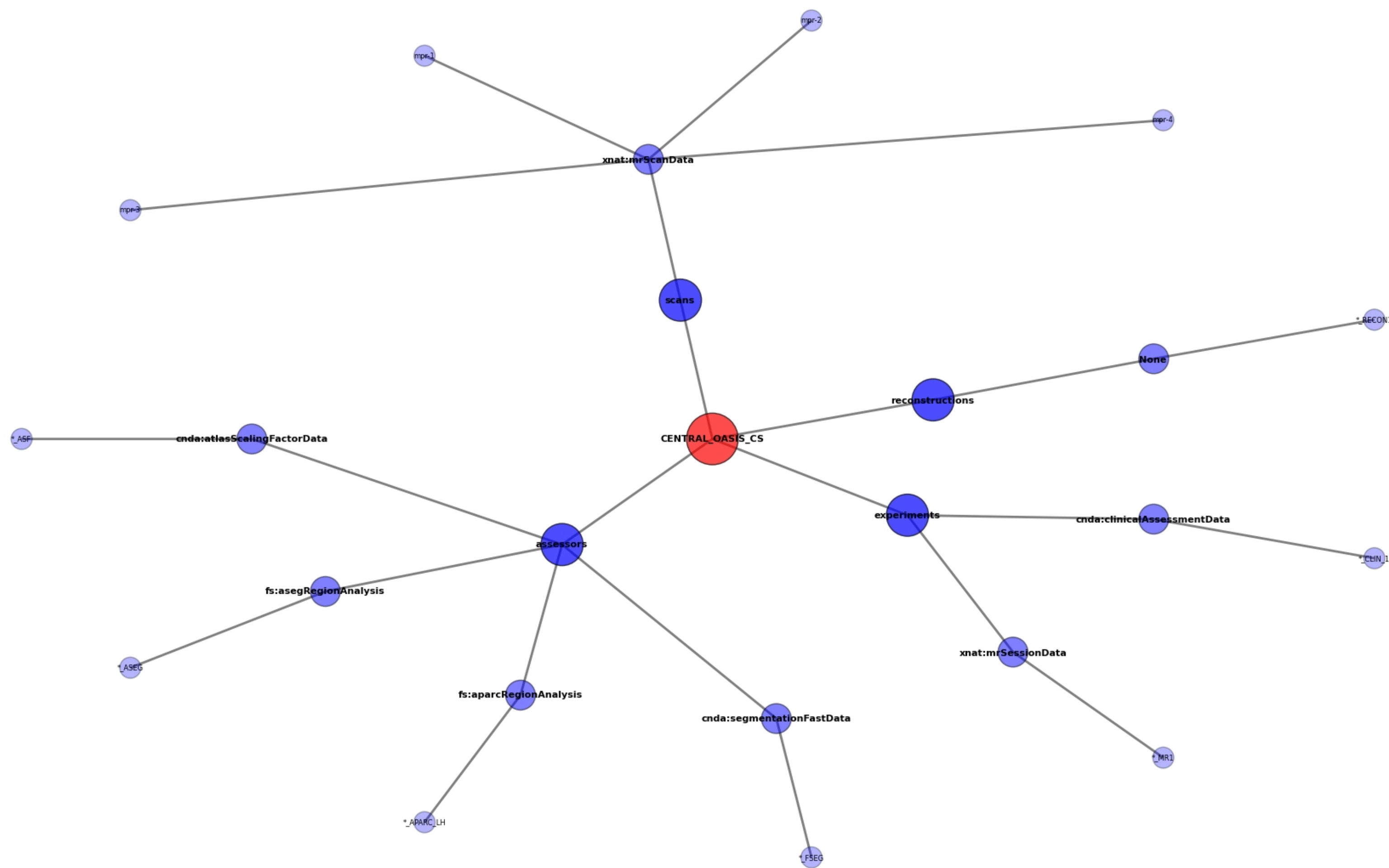
- Administrators are using a consistent vocabulary across projects, that maps to XNAT datatypes.

```
>>> imagen.inspect.naming_conventions()
{
 '/projects/IMAGEN/subjects/*/experiments/SessionA_*':
'xnat:mrSessionData',
 '/projects/IMAGEN/subjects/*/experiments/*/assessors/*_ADNI_MPRAGE':
'imagen:niftiScanData',
 '/projects/IMAGEN/subjects/*/experiments/*/assessors/*_DTI':
'imagen:niftiScanData',
 '/projects/IMAGEN/subjects/*/experiments/*/assessors/beta_0003_EPI_faces_*':
'spm:betaData',
...}
```

- The mapping is stored in a configuration file that can be edited.

Naming conventions

CENTRAL_OASIS_CS



Naming conventions

IMAGEN

IMAGEN
assessors
experiments
scans
spm:betaData
imagen:niftiScanData
spm:conData
spm:job_statsintraData
spm:spmfData
spm:maskData
spm:statsintra_mat_Data
behavioural:recognition_taskData
spm:wmeanaData
spm:wa_matData
spm:wamaskData
spm:waData
spm:cData
spm:meanaData
spm:essData
behavioural:mid_taskData
spm:spmtData
xnat:mrScanData
imagen:recruitmentInfosData
psytool:palp_v2Data
psytool:palpData
dawba:computerData
psytool:identData
dawba:childData
psytool:dotprobeData
imagen:imagenSubjectVariablesData
dawba:youthData
xnat:mrSessionData
psytool:surps_parentData

Browsing a database

Simple requests

```
>>> imagen.select.projects().get()
>>> imagen.select('/projects').get()
```

GET /REST/projects?format=json

Nested requests

```
>>> imagen.select.projects().subjects().get()
>>> imagen.select('/projects/*/subjects').get()
>>> imagen.select('/projects/subjects').get()
>>> imagen.select('//subjects').get()
['IMAGEN_000000001274', 'IMAGEN_000000075717', ...,'IMAGEN_000099954902']
```

GET /REST/projects?format=json
GET /REST/projects/IMAGEN/subjects?format=json
...

Filtered requests

```
>>> imagen.select.projects().subjects('*55*42').get()
>>> imagen.select('/projects/*/subjects/*55*42').get()
>>> imagen.select('//subjects/*55*42').get()
['IMAGEN_000055203542', 'IMAGEN_000055982442', 'IMAGEN_000097555742']
```

Search engine interface

Expressing queries

A single constraint is a tuple: ('field', 'operator', 'value')
A query is a list a constraints: [contraints, 'ANDorOR']
A query may have sub-queries: [contraints, sub-queries, 'ANDorOR']
Example: **[('psytool:tci_parentData/TCI122', '=', '1'), 'AND']**

Datatypes and datafields in: interface.inspect.datatypes(...)

Get REST resources

```
>>> imagen.select('//experiments'
        ).where([('psytool:tci_parentData/TCI122', '=', '1'), 'AND'])
>>> imagen.select('//assessors/*ADNI_MPRAGE*/out/resources/files'
        ).where([('psytool:tci_parentData/TCI122', '=', '1'), 'AND'])
```

Get table values

```
>>> imagen.select('xnat:subjectData',
                  ['xnat:subjectData/PROJECT',
                   'xnat:subjectData/SUBJECT_ID']
        ).where(...)
```

project,subject_id
IMAGEN,IMAGEN_000000001274
IMAGEN,IMAGEN_000000075717
IMAGEN,IMAGEN_000000106601
IMAGEN,IMAGEN_000000106871
IMAGEN,IMAGEN_000000112288
IMAGEN,IMAGEN_000000215284
IMAGEN,IMAGEN_000000240546
IMAGEN,IMAGEN_000000292802
...

Retrieving data

Lazy access to resources

```
>>> imagen.select('/project/IMAGEN/subject/s001/experiments/e001001/files')
    <Collection Object> 149779788
>>> imagen.select('//experiments/e001001/files').where(...)
    <Collection Object> 526777631
>>> imagen.select('//experiments/e001001/files').get()
    list_of_ids
>>> for exp_file in imagen.select('//experiments/e001001/files'):
>>>     print exp_file
```

GET /REST/projects?format=json
GET /REST/projects/IMAGEN/subjects?format=json
GET /REST/projects/IMAGEN/subjects/s001/experiments?format=json
...

Files download

```
>>> for exp_file in imagen.select('//experiments/e001001/files'):
>>>     print exp_file.get()
```

Resources attributes

Work in progress...

Processing some data

Running FSL BET brain extraction tool on 8 processors

```
>>> imagen = Interface('server', 'login', 'pass')

>>> def bet(in_image):
>>>     p, name = os.path.split(in_image)
>>>     in_image = os.path.join(p, name.rsplit('.')[0])
>>>     out_image = os.path.join(p, name.rsplit('.')[0]+'_brain')
>>>     Popen('bet2 %s %s -f 0.5 -g 0 '%(in_image, out_image),
            shell=True).communicate()
>>>     return out_image

>>> pool = Pool(processes=8)

>>> for adni_mprage in \
        imagen.select('//assessors/*ADNI*/out/resources/files'
             ).where('psytool:tci_parentData/TCI051 = 1 AND'):

>>>     bet(adni_mprage.get())
OR
>>>     pool.apply_async(bet, (adni_mprage.get(),))

>>> pool.close()
>>> pool.join()
```

Sequential execution

Parallel execution

Inserting new data

- Creating an element with REST requires its ancestors to exist
- But we often want to create multiple levels at once

Explicit typing

```
>>> imagen.select('/project/IMAGEN/subjects/s001'
                  '/experiments/e001001/assessors/new_assess'
                  ).create(experiments='xnat:petSessionData')
```

PUT subject_URI?xsiType=xnat:subjectData
PUT experiment_URI?xsiType=xnat:petSessionData
PUT assessor_URI?xsiType=xnat:imageAssessorData

Using default types

```
>>> imagen.select('/project/IMAGEN/subjects/s001/experiments/e001001'
                  '/assessors/new_assess').create()
```

PUT subject_URI?xsiType=xnat:subjectData
PUT experiment_URI?xsiType=xnat:mrSessionData
PUT assessor_URI?xsiType=xnat:imageAssessorData

Inserting new data

Using naming conventions

```
>>> imagen.select('/project/IMAGEN/subjects/IMAGEN_000000123456'
                  '/experiments/IMAGEN_000000123456_SessionA'
                  '/assessors/IMAGEN_000000123456_SessionA_ADNI_MPRAGE_nii'
                  ).create()
```

PUT subject_URI?xsiType=xnat:subjectData
PUT experiment_URI?xsiType=xnat:mrSessionData
PUT assessor_URI?xsiType=imagen:niftiScanData

Default types < naming conventions < explicit typing

Uploading files

```
>>> imagen.select('/project/IMAGEN/subjects/IMAGEN_000000123456'
                  '/experiments/IMAGEN_000000123456_SessionA'
                  '/assessors/IMAGEN_000000123456_SessionA_ADNI_MPRAGE_nii'
                  '/out/resources/32492/file/ADNI_MPRAGE.nii.gz'
                  ).put('/path_to_local_file/myT1.nii.gz', content='T1')
```

Cache system

- Using HTTP cache to avoid re-downloading files
- Queries and files and stored in a cache directory

- Cache modes:
  - Online: cache has a lifespan
  - Offline: cache is always used
  - Mixed: cache is used when it exists

Work in progress

- On pyxnat
  - A few performance improvements
  - Documentation / packaging
  - Source code made available
- With pyxnat
  - Virtual File System
  - Command Line Interface
  - nipype http://nipy.sourceforge.net/nipype/

Demo time

Thanks!

Questions?