ConnectomeDB, pyxnat, and the OHBM Hackathon

One of the datasets available for the OHBM Hackathon is the Q1 public data release from the Hum an Connectome Project. In addition to the imaging data, which are mirrored on S3 for easy access from AWS, a great deal of imaging metadata and associated non-imaging data is accessible through ConnectomeDB, a web application built on the XNAT imaging informatics platform.

pyxnat is a library that provides a Python language API to XNAT's RESTful web services. In this tutorial, we'll use pyxnat to access behavioral measures stored in ConnectomeDB and to view and download the imaging data. Even if you're not a Pythonista, read on, as the underlying XNAT REST API can be accessed from just about any language. I have small examples of code using the REST API in bash, Java and Clojure, and I'd probably find it amusing to cook up an example in your favorite language; send me mail if you'd like details.

Now that OHBM and the hackathon are past, most of this tutorial has been moved to a generic tutorial on pyxnat and ConnectomeDB. A few topics, mostly related to AWS and the S3 mirror of the Q1 data, remain here.

Unknown macro: 'html'

Table of Contents

Unknown macro: 'html'

Accessing imaging data

Before trying to access the data, it's important to understand what's in the Q1 release. The Q1 data release documentation describes the session structure and file layout in detail. The Q1 imaging data are mirrored on Amazon's S3, which is particularly useful for copying data into an EC2 instance. The Python library boto provides an interface for many of Amazon's web services, including S3. If you installed the HCP-customized pyxnat, you already have boto.

I'm working to extend pyxnat to translate between the internal storage paths on ConnectomeDB and the (differently organized) copy of the data on S3. You really don't need to wait for this, though: the example searches above produce subject labels, which is enough information to point you to the right data directories on S3 – for example, subject 100307's data can be found at s3://hcp.aws.amazon.com/q1/100307/ . Consult the hackathon HCP data release announcement fo r more details.

Browsing the imaging data

In order to start exploring and using the S3 Q1 mirror, you'll need to set up your AWS account and get access to the Amazon-hosted data. This process will get you an access key and a secret key, which you'll use to authenticate against S3. From a Python command line, you can browse the Q1 data by getting a handle to the "bucket" where the data are stored:

```
>>> from boto.s3.connection import S3Connection
>>> s3 =
S3Connection('your-access-key','your-secret-key
')
>>> bucket =
s3.get_bucket('hcp.aws.amazon.com')
```

S3 is a key-value-oriented store, rather than a hierarchical filesystem, but the HCP Q1 data is stored with keys that echo a regular file system. boto's interface to S3 makes it easy to pretend you're walking a file tree. There is a single root element q1, and each subject is a child of that root:

```
>>> [k.name for k in bucket.list('q1/','/')]
[u'q1/', u'q1/100307/', u'q1/103515/',
u'q1/103818/', u'q1/111312/', u'q1/114924/',
u'q1/117122/', u'q1/118932/', u'q1/119833/',
u'q1/120212/', u'q1/125525/', u'q1/128632/',
u'q1/130013/', u'q1/131621/', u'q1/137128/',
u'q1/138231/', u'q1/142828/', u'q1/143325/',
u'q1/144226/', u'q1/149337/', u'q1/150423/',
u'q1/153429/', u'q1/156637/', u'q1/159239/',
u'q1/161731/', u'q1/162329/', u'q1/167743/',
u'q1/172332/', u'q1/182739/', u'q1/191437/',
u'q1/192439/', u'q1/192540/', u'q1/194140/',
u'q1/197550/', u'q1/199150/', u'q1/199251/',
u'q1/200614/', u'q1/201111/', u'q1/210617/',
u'q1/217429/', u'q1/249947/', u'q1/250427/',
u'q1/255639/', u'q1/304020/', u'q1/307127/',
u'q1/329440/', u'q1/355542/', u'q1/499566/',
u'q1/530635/', u'q1/559053/', u'q1/585862/',
u'q1/611231/', u'q1/638049/', u'q1/665254/',
u'q1/672756/', u'q1/685058/', u'q1/729557/',
u'q1/732243/', u'q1/792564/', u'q1/826353/',
u'q1/856766/', u'q1/859671/', u'q1/861456/',
u'q1/865363/', u'q1/877168/', u'q1/889579/',
u'q1/894673/', u'q1/896778/', u'q1/896879/',
u'q1/901139/', u'q1/917255/', u'q1/937160/']
```

Each subject is organized as described in the Q1 data release documentation.

```
>>> [k.name for k in
bucket.list('q1/100307/','/')]
[u'q1/100307/.xdlm/', u'q1/100307/Diffusion/',
u'q1/100307/MNINonLinear/', u'q1/100307/T1w/',
u'q1/100307/release-notes/',
u'q1/100307/unprocessed/']
```

The three key fragments associated with the "minimally preprocessed" data are `Diffusion`, `MNINonLinear`, and `T1w`. The key fragment `.xdlm` marks file manifests for download integrity checking, including checksums; while `unprocessed` marks unprocessed data.

Downloading files from S3

Now let's copy all files for the motor task with left-to-right phase encoding from S3 to a local disk.

```
>>> import errno,os,os.path
>>> for k in
bucket.list('q1/100307/MNINonLinear/Results/tfM
RI_MOTOR_LR/'):
...   dir = os.path.dirname(k.name)
...   try: os.makedirs(dir)
...   except OSError as exc:
...     if exc.errno == errno.EEXIST and
os.path.isdir(dir): pass
...     else: raise
...   with open(k.name, 'w') as f:
...     k.get_contents_to_file(f)
...
>>>
```

Note that we use `bucket.list(...)`a little differently here: with one argument, it returns all keys starting with the provided text, which is comparable to a full recursive listing in a hierarchical file system.

Accessing S3 data through the Subject object

The instructions above handle ConnectomeDB and S3 as different worlds; there is some limited support for bridging this gap. Pyxnat can maintain a local mirror of the ConnectomeDB data, with contents downloaded on request. The first piece you'll need is an object representing both sides of the mirror (the local file space and the S3 bucket):

```
>>> from pyxnat.core.mirror import S3Mirror
>>> mirror =
S3Mirror.open('hcp.aws.amazon.com','your-access
-key','your-secret-key','/path/to/local/mirror'
)
```

Next, you'll need to hand this to the pyxnat Interface:

```
>>> cdb =
pyxnat.Interface('https://db.humanconnectomem.o
rg','username','password',data_mirror=mirror)
>>>
```

You can now access local copies of the data files through the subject object:

```
>>>
cdb.project('HCP_Q1').subject('100307').files('
MNINonLinear/Results/tfMRI_MOTOR_LR')
[u'/path/to/local/mirror/q1/100307/MNINonLinear
/Results/tfMRI_MOTOR_LR/...]
```

The return value from this call is a list of local file paths for the requested data. If the files are already on your disk, this will return quickly; if not, the files are downloaded before the files() call returns. You can request all of a subject's data by skipping the root path argument:

```
>>>
cdb.project('HCP_Q1').subject('100307').files()
```

Note that this will most likely take a really long time, because a single subject is close to 20 GB, unless you have a very fat pipe to S3 -- say, from an EC2 instance. (This is why I'm not even showing the return value. It's a long list of files. You probably should find think about whether you really need all those files, and find a better way to get them if you do.)