python-icat

A Library for Writing ICAT Clients in Python

Rolf Krahl

ICAT Meeting, Dublin, Mar 2014

Introduction

- SOAP is used as the access protocol for ICAT.
- Clients exist for different programming languages, including Java and Python.
- The most popular SOAP library for Python is Suds.
- python-icat aims to make writing ICAT clients with Python simpler.

Design Goals

python-icat is build on top of Suds.

Goals

- Keep the general structure and flexibility of Suds.
- Simplify things where possible.
- Try to remove annoying details.
- Make use object oriented design.

A typical python-icat program might be mistaken for a generic Suds program at first glance. It's just somewhat simpler.

Example: Add a Datafile

Using plain Suds

```
dataset = client.service.search(sessionId,
                "Dataset[name='e201215']")[0]
format = client.service.search(sessionId,
                "DatafileFormat[name='NeXus']")[0]
datafile = client.factory.create("datafile")
datafile.dataset = dataset
datafile.datafileFormat = format
datafile.name = "e201215-7.nxs"
datafile.id = client.service.create(sessionId, datafile)
```

Example: Add a Datafile

Using plain Suds

```
dataset = client.service.search(sessionId,
              "Dataset[name='e201215']")[0]
format = client.service.search(sessionId,
              "DatafileFormat[name='NeXus']")[0]
datafile = client.factory.create("datafile")
datafile.dataset = dataset
datafile.datafileFormat = format
datafile.name = "e201215-7.nxs"
datafile.id = client.service.create(sessionId, datafile)
```

Using python-icat

```
dataset = client.search("Dataset[name='e201215']")[0]
format = client.search("DatafileFormat[name='NeXus']")[0]
datafile = client.new("datafile")
datafile.dataset = dataset
datafile.datafileFormat = format
datafile.name = "e201215-7.nxs"
datafile.create()
```

Example: Add a Datafile

Or even:

Using python-icat

```
dataset = client.search("Dataset[name='e201215']")[0]
format = client.search("DatafileFormat[name='NeXus']")[0]
client.new("datafile",
           dataset=dataset,
           datafileFormat=format,
           name="e201215-7.nxs").create()
```

Notes

- `client.new(...)` creates a new ICAT entity objects. Use this in place of `client.factory.create(...)`.
- `client.new(...)` optionally accepts keyword/value arguments to set attributes.
- ICAT API methods are defined as methods in the python-icat client. Replace `client.service.<method>(...)` by `client.<method>(...)`.
- Don't care about the session id, the python-icat client remembers it and adds it to the ICAT method calls as needed.
- ICAT entity objects have their own methods: e.g. `datafile.create()`.

Example: Add Keywords to an Investigation

Using plain Suds

```
investigation = client.service.search(sessionId,
        "Investigation[name='2010-E2-0489-1']")[0]
keywords = []
for k in ["Foo", "Bar", "Baz"]:
    keyword = client.factory.create("keyword")
    keyword.name = k
    keyword.investigation = investigation
    keywords.append(keyword)
client.service.createMany(sessionId, keywords)
```

Example: Add Keywords to an Investigation

Using plain Suds

```
investigation = client.service.search(sessionId,
        "Investigation[name='2010-E2-0489-1']")[0]
keywords = []
for k in ["Foo", "Bar", "Baz"]:
    keyword = client.factory.create("keyword")
    keyword.name = k
    keyword.investigation = investigation
    keywords.append(keyword)
client.service.createMany(sessionId, keywords)
```

Using python-icat

```
investigation = client.search(
        "Investigation[name='2010-E2-0489-1']")[0]
investigation.addKeywords(["Foo", "Bar", "Baz"])
```

Example: Create a Group

Using plain Suds

```
users = [ jbotu , jdoe , nbour ]
group = client . factory . create ( "group" )
group . name = "investigation_42_reader"
group . id = client . service . create ( sessionId , group )
ugs = []
for u in users :
    ug = client . factory . create ( "userGroup" )
    ug . user = u
    ug . group = group
    ugs . append ( ug )
client . service . createMany ( sessionId , ugs )
```

Example: Create a Group

Using plain Suds

```
users = [ jbotu , jdoe , nbour ]
group = client . factory . create ("group")
group . name = "investigation_42_reader"
group . id = client . service . create ( sessionId , group )
ugs = []
for u in users :
    ug = client . factory . create ("userGroup")
    ug . user = u
    ug . group = group
    ugs . append ( ug )
client . service . createMany ( sessionId , ugs )
```

Using python-icat

```
users = [ jbotu , jdoe , nbour ]
group = client . createGroup ("investigation_42_reader", users )
```

Example: Login

Using plain Suds

```
client = suds.client.Client(url)
credentials = client.factory.create("credentials")
credentials.entry.append(
        [ { 'key':'username', 'value':username },
          { 'key':'password', 'value':password } ])
sessionId = client.service.login(auth, credentials)

# ...

client.service.logout(sessionId)
```

Example: Login

Using plain Suds

```
client = suds.client.Client(url)
credentials = client.factory.create("credentials")
credentials.entry.append(
        [ { 'key':'username', 'value':username },
          { 'key':'password', 'value':password } ])
sessionId = client.service.login(auth, credentials)

# ...

client.service.logout(sessionId)
```

Using python-icat

```
client = icat.client.Client(url)
credentials = { 'username':username, 'password':password }
client.login(auth, credentials)
```

ICAT Version

- Drawback of python-icat: it depends on the ICAT version.
- When the ICAT API changes, the library needs to get adapted to the new version.
- Currently supported: 4.2.* and 4.3.*, the API version is checked automatically.
- A module `icat.icatcheck` tests compatibility helps to adapt the library to new versions.
- Advantage: some incompatibities between ICAT versions are handled by python-icat and hidden from the application.

Example: Add Instrument to an Investigation

Using plain Suds

```
investigation = client.service.search(sessionId,
        "Investigation INCLUDE 1 [name='2010-E2-0489-1']")[0]
instrument = client.service.search(sessionId,
        "Instrument[name='HIKE']")[0]
if client.service.getApiVersion() < '4.3.0':
    investigation.instrument = instrument
    client.service.update(sessionId, investigation)
else:
    ii = client.factory.create('investigationInstrument')
    ii.investigation = investigation
    ii.instrument = instrument
    client.service.create(sessionId, ii)
```

Example: Add Instrument to an Investigation

Using plain Suds

```
investigation = client.service.search(sessionId,
        "Investigation INCLUDE 1 [name='2010-E2-0489-1']")[0]
instrument = client.service.search(sessionId,
        "Instrument[name='HIKE']")[0]
if client.service.getApiVersion() < '4.3.0':
    investigation.instrument = instrument
    client.service.update(sessionId, investigation)
else:
    ii = client.factory.create('investigationInstrument')
    ii.investigation = investigation
    ii.instrument = instrument
    client.service.create(sessionId, ii)
```

Using python-icat

```
investigation = client.search(
        "Investigation[name='2010-E2-0489-1']")[0]
instrument = client.search("Instrument[name='HIKE']")[0]
investigation.addInstrument(instrument)
```

Config

- A typical ICAT client always needs the same set of command line arguments: URL of the ICAT service, authentication plugin name, username, and password.
- A module `icat.config` takes care of this: it defines the command line arguments.
- Configuration options may be set via command line arguments, environment variables, configuration files, and default values (in this order, first match wins). The password may also be read from interactive keyboard input.
- Of course, a program may define additional custom arguments.

Example: Config

Using plain Suds

```
url = "https://" + sys.argv[1] + ":" + sys.argv[2] \
    + "/ICATService/ICAT?wsdl"
auth = sys.argv[3]
username = sys.argv[5]
password = sys.argv[7]
client = suds.client.Client(url)
credentials = client.factory.create("credentials")
credentials.entry.append(
        [ { 'key':'username', 'value':username },
          { 'key':'password', 'value':password } ])
sessionId = client.service.login(auth, credentials)
```

Example: Config

Using plain Suds

```
url = "https://" + sys.argv[1] + ":" + sys.argv[2] \
    + "/ICATService/ICAT?wsdl"
auth = sys.argv[3]
username = sys.argv[5]
password = sys.argv[7]
client = suds.client.Client(url)
credentials = client.factory.create("credentials")
credentials.entry.append(
        [ { 'key':'username', 'value':username },
          { 'key':'password', 'value':password } ])
sessionId = client.service.login(auth, credentials)
```

Using python-icat

```
config = icat.config.Config()
conf = config.getconfig()
client = icat.Client(conf.url, **conf.client_kwargs)
client.login(conf.auth, conf.credentials)
```

Config

Default Command Line Arguments

```
usage: login-icat-config.py [options]

optional arguments:
  -h, --help            show this help message and exit
  -c CONFIGFILE, --configfile CONFIGFILE
                        config file
  -s SECTION, --configsection SECTION
                        section in the config file
  -w URL, --url URL     URL to the web service description
  --http-proxy HTTP_PROXY
                        proxy to use for http requests
  --https-proxy HTTPS_PROXY
                        proxy to use for https requests
  -a AUTH, --auth AUTH  authentication plugin
  -u USERNAME, --user USERNAME
                        username
  -p PASSWORD, --pass PASSWORD
                        password
  -P, --prompt-pass     prompt for the password
```

Example: Exception Handling

Using plain Suds

```
try:
    sessionId = client.service.login(auth, credentials)
except suds.WebFault as e:
    if e.fault.detail.IcatException.type == 'SESSION':
        print "Login failed: %s" % e
    else:
        raise
```

Example: Exception Handling

Using plain Suds

```
try:
    sessionId = client.service.login(auth, credentials)
except suds.WebFault as e:
    if e.fault.detail.IcatException.type == 'SESSION':
        print "Login failed: %s" % e
    else:
        raise
```

Using python-icat

```
try:
    client.login(conf.auth, conf.credentials)
except ICATSessionError as e:
    print "Login failed: %s" % e
```

Example: Searching

Using plain Suds

```
searchres = client.service.search(sessionId, "Facility")
if len(searchres) != 1:
    raise RuntimeError("Expected to find one facility")
else:
    facility = searchres[0]
```

Example: Searching

Using plain Suds

```
searchres = client.service.search(sessionId, "Facility")
if len(searchres) != 1:
    raise RuntimeError("Expected to find one facility")
else:
    facility = searchres[0]
```

Using python-icat

```
facility = client.assertedSearch("Facility")[0]
```

Example: Searching

Using plain Suds

```
searchres = client.service.search(sessionId, "Facility")
if len(searchres) != 1:
    raise RuntimeError("Expected to find one facility")
else:
    facility = searchres[0]
```

Using python-icat

```
facility = client.assertedSearch("Facility")[0]
```

Using python-icat (more)

```
# Assert there is at least one Investigation
investigation = client.assertedSearch("Investigation",
                                      assertmax=None)[0]
# Assert there is at most one Instrument
res = client.assertedSearch("Instrument", assertmin=0)
```

Misc

- A module `icat.cgi` helps writing CGI scripts. It does session management: the ICAT session Id is set as a cookie in the user's browser.
- Methods `Entity.getUniqueKey()` and `Client.searchUniqueKey()` to create a unique object identifier and to search for the object corresponding to an identifier respectively.
- Example scripts `icatdump.py` and `icatrestore.py` that dump the whole content of an ICAT to a file (YAML) and restore it from the dump file respectively.

Internals

- `icat.client.Client` is a `suds.client.Client`. Everything you can do with a Suds client, you can do with a python-icat client.
- The ICAT entity objects created by `client.new(...)` or returned by a search live in a hierarchy of classes based on `icat.entity.Entity`.
- The ICAT entity objects mimic very closely the behavior of corresponding Suds objects. They are converted transparently from and to Suds objects as appropriate.

System Requirements and Download

System Requirements

- Python 2.6 or newer (Python 2.6 requires a patch).
- Suds, either 0.4 or jurko fork, the latter is recommended.
- argparse (in system library in Python 2.7 or newer).
- The example scripts use PyYAML, but this is not needed to use the library itself.

Download

- python-icat 0.4.0 available at `http://code.google.com/p/icatproject/wiki/PythonIcat`
- BSD license.

System Requirements and Download

System Requirements

- Python 2.6 or newer (Python 2.6 requires a patch).
- Suds, either 0.4 or jurko fork, the latter is recommended.
- argparse (in system library in Python 2.7 or newer).
- The example scripts use PyYAML, but this is not needed to use the library itself.

Download

- python-icat 0.4.0 available at `http://code.google.com/p/icatproject/wiki/PythonIcat`
- BSD license.

Thank you for your attention! Questions?