pyOLS

Python Ontology Lookup Service

PyOLS is a simple XML-RPC based ontology storage and lookup service.

What does that mean?

- XML-RPC is a protocol for making remote procedure calls – it uses HTTP and XML to call functions on remote computers.

- An ontology is a set of related terms and the relationships which bind them.

And what can PyOLS do with these ontologies (relationships, keywords and associations between keywords and “real things”)?

- Store them
- Graph them
- Search them

Installing pyOLS is simple:

```
$ tar -zxvf pyols.tar.gz
$ cd pyols/
$ python setup.py install
$ pyols -c env
$ pyols env
18:19:42 INFO Starting standalone server on port 8000
```

Using pyOLS is just as simple:

```
>>> from xmlrpclib import ServerProxy
>>> s = ServerProxy("http://localhost:8000/example")
>>> s.addKeyword('INTERCAL')
{'associations': [], 'description': '',
 'disambiguation': '', 'name': 'INTERCAL', ... }
>>> s.addKeyword('Befunge')
>>> s.addKeyword('programming language')
>>> s.addRelation('an esoteric', 0.75)
>>> s.addKeywordRelationship('INTERCAL',
...                'an esoteric', 'programming language')
>>> s.addKeywordRelationship('Befunge',
...                'an esoteric', 'programming language')
```

(where the path (in this case, /EXAMPLE) is the namespace that will be used)

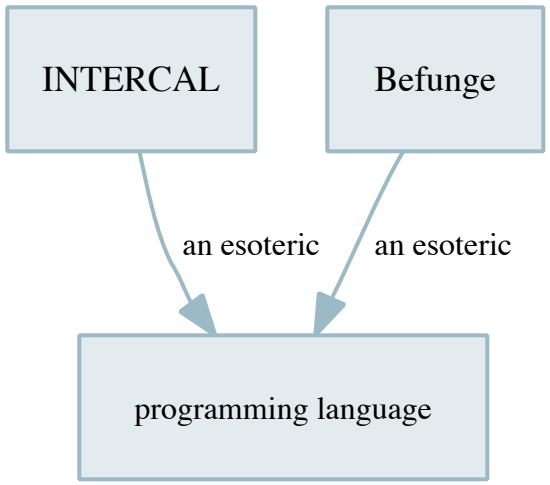
Now that there is a bit of data, it can be searched:

```
>>> s.getRelatedKeywords('Befunge')
{'Befunge': 1, 'INTERCAL': 0.5625,
 'programming language': 0.75}
```

And Dot source can also be generated:

```
>>> s.getDotSource()
digraph G {
...
"INTERCAL" [fontsize="9", tooltip=""];
"Befunge" [fontsize="9", tooltip=""];
"Befunge" -> "programming language" [label="an esoteric"];
"INTERCAL" -> "programming language" [label="an esoteric"];
}
```

Which, when graphed with Graphviz, yields a graph of keywords and their relationships within the ontology:



Ok, great. But how can you do anything useful with this ontology?

In the same way a class must be instantiated, **keywords** must be **associated** with **instances.**

There are two ways this can be accomplished in pyOLS:

- Using the built in KEYWORDASSOCIATION type

- Using a custom relationship type

First, using the KEYWORDASSOCIATION class:

```
>>> s.methodHelp('addKeywordAssociation')
Add a KeywordAssociation to the ontology.  If an instance
already exist, it will be updated.  The new instance is
returned.
addKeywordAssociation(keyword, path, description=u'')
>>> s.addKeywordAssociation('INTERCAL',
...     '/user/Wolever', 'DO :1 <- #0$#256')
```

(Where the PATH can be any Unicode string up to 512 characters long)

KEYWORDASSOCIATIONS can, like the other data storage types can also be queried:

```
>>> s.queryKeywordAssociations(None, '/user/Wolever')
[{'path':'/user/Wolever', 'description':'DO :1 <- #0$#256',
  'keyword': {'name': 'INTERCAL' ... }}]
>>>
```

(Note that NONE is used as the first argument of QUERYKEYWORDASSOCIATIONS because we do not care about the value of KEYWORD)

```
>>> s.addRelation('likes')
>>> s.addKeyword('/user/Wolever')
>>> s.addRelation('/user/Wolever', 'likes', 'INTERCAL')
```

As above, relations of this type can also be queried:

```
>>> s.queryKeywords('/user/Wolever')
[{'name': '/user/Wolever', left_relations = [{
    'relation': 'likes', 'right': 'INTERCAL'...
>>>
```

	KEYWORDASSOCIATION	Custom relation type
Advantages	Simple, easy to query	More expressive, possible to relate instances to each other
Disadvantages	Hard to create complex relationships	Slightly more difficult to create, currently there is no way to distinguish between keywords and instances pretending to be keywords

As mentioned briefly, NAMESPACES are another feature built into PyOLS: they allow one instance of PyOLS to store many different, separate, ontologies. This means, for example, the relationship "SYNONYMOF" could have a weight of 0.5 in one namespace, and a weight of 0.75 in a different namespace.

When a connection to PyOLS is established, the namespace is specified by the path that is requested (for example, connecting to http://server/EXAMPLE would use the "example" namespace).

Finally, there are also a few 'standard' XML-RPC functions that PyOLS supports which may be worth knowing:

```
>>> s = ServerProxy("http://localhost:8000/example")
>>> s.system.listMethods()
['addKeyword', 'addKeywordAssociation',
...
'queryRelations', 'system.listMethods',
'system.methodHelp', 'system.methodSignature',
'system.multicall']
>>> print s.system.methodHelp('addKeyword')
Add a Keyword to the ontology.  If an instance already
exist, it will be updated.  The new instance is returned.
addKeyword(name, disambiguation=u'', description=u'',
associations=None, left_relations=None,
right_relations=None)
```

!! BIG DISCLAIMER !!

PyOLS is in it's very first version, and is not complete. There is useful functionality that has been left out, and probably useless functionality that has been built in.

If there is something useful that is missing, or something useless which has been included, don't hesitate to report it (or submit a patch).