API Documentation

API Documentation

May 9, 2007

Contents

1 Package astrogrid

This is a functional python interface to the Astrogrid Client Runtime (ACR).

The following classes are provided by

```
>>> from astrogrid import *
```

- Applications - run remote applications
- Community - account login and logout
- ConeSearch - execute a cone search
- Configuration - configure AR access
- DSA - run a remote ADQL query
- MySpace - access to MySpace (i.e. list file, delete, etc.)
- Registry - registry search
- SiapSearch - execute a siap search
- StapSearch - execute a stap search

The following additional methods are available from within the acr class

```
>>> from astrogrid import acr
```

- acr.login - login
- acr.isLoggedIn - returns True/False depending on the user login status

In order to use of the class methods, use:

```
>>> c = Community()
>>> c.guiLogin()
```

See the additional information for each class for the methods that it contains.

Additionally all the ACR calls are available using the acr variable, e.g.,

```
>>> from astrogrid import acr
>>> acr.astrogrid.community.login('user','password','community')
```

Credits:

ACR is Noel Python interface is Eduardo, Noel and John

Version: 0.5

Date: $ $

1.1 Modules

- **myspace**: Python interface to MySpace
  (Section 6, p. 11)
- **plastic** *(Section 7, p. 13)*
- **registry**: Module to send queries to query the registry.
  (Section 8, p. 14)
- **siap**: Send queries to a siap server.
  (Section 9, p. 15)
- **stap**: Send queries to a stap server.
  (Section 10, p. 17)
- **system**: Module to send queries to query the registry.
  (Section 11, p. 19)
- **utils**: Several applications.
  (Section 12, p. 21)

1.2 Variables

Name	Description
__revision__	**Value:** `'$ $'`
acr	**Value:** `ACR()`

1.3 Class ACR

Connect to the Astrogrid ACR. Provides direct access to the XMLRPC methods as well as providing some convenience shortcuts.

Examples:

```
>>> from astrogrid import acr
>>> acr.login()
>>> print acr.system.apihelp.listMethods()
>>> print acr.methods
```

1.3.1 Methods

__init__(*self*)

Connects to the Astrogrid ACR

autologin(*self*)

askpass(*self*)

login(*self*, *key*=`None`, *username*=`None`, *password*=`None`, *community*=`None`)

Convenience method to login.

Parameters

`username:` User name *(type=str)*
`password:` Password *(type=str)*
`community:` Community *(type=str)*

isLoggedIn(*self*)

Convenience method to check if logged in

logout(*self*)

__getattr__(*self*, *attr*)

2 Module astrogrid.applications

Run server side applications.

2.1 Class Applications

This is the main class to send applications to be run remotely and asynchronously in the system.

Example:

```
>>> from astrogrid import Applications
>>> app = Applications('ivo://starlink.ac.uk/stilts1.3', 'tcopy')
... fill in app.inputs and app.outputs ...
>>> run1 = app.submit()
... fill in app.inputs and app.outputs with other values ...
>>> run2 = app.submit()
>>> run1.status()
COMPLETED
>>> run2.status()
RUNNING
```

2.1.1 Methods

__init__(*self*, *ivorn*=None, *interface*='default')

Parameters

`ivorn:` String containing the IVORN of the job to be submitted *(type=str)*

`interface:` If the application provides various interfaces, the specify the chosen one here, otherwise it will pick up the default. *(type=str)*

list(*self*)

Return a list of all applications

template(*self*, *ivorn*, *interface*='default')

submit(*self*, *server*=None, *validate*=True)

Submits a job.

Parameters

`server:` Server to send the job to. If not defined just picks one. *(type=str)*

`validate:` Validate document before submitting. Default: True *(type=bool)*

Return Value

ceapp An instance of CEApp allowing to check status and get results.

2.2 Class DSA

```
>>> from astrogrid import DSA
>>> mydsa = DSA('ivo://wfau.roe.ac.uk/twomass-dsa/ceaApplication')
>>> mydsa.query(adqlx)
```

2.2.1 Methods

__init__(*self*, *service*)

Parameters

`service:` IVORN of the service to be queried *(type=str)*

query(*self*, *sql*, *ofmt='votable'*, *saveAs=None*)

Parameters

`sql:` SQL query *(type=str)*

`ofmt:` Output format (votable|votbin|csv). Default: vot *(type=str)*

`saveAs:` File name to save the query to in MySpace. Default: None *(type=str)*

Return Value

ceapp An instance of CEApp allowing to check status and get results.

2.2.2 Instance Variables

Name	Description
doc	Documentation about the application
info	Details of the application

2.3 Class CEApp

Check the status of a CEA running application and return results. An instance of this class is returned by the Applications and DSA classes when submitting a job.

2.3.1 Methods

__init__(*self*, *execID*)

status(*self*)

results(*self*)

3 Module astrogrid.community

Module to login/logout and get user information.

3.1 Class Community

Community related tasks.

Example:

```
>>> from astrogrid import Community
>>> c = Community()
>>> c.isLoggedIn()
False
>>> c.login('me', 'mypass', 'mycommunity')
>>> c.isLoggedIn()
True
>>> c.logout()
```

3.1.1 Methods

__init__(*self*)

isLoggedIn(*self*)

Check if logged in.

Return Value

True or False

guiLogin(*self*)

Ask for credentials using the Workbench GUI

login(*self*, *username*, *password*, *community*)

Login with username and password. If not defined then use the ones defined in the configuration file.

Parameters

`username:` User name *(type=str)*
`password:` Password *(type=str)*
`community:` Community *(type=str)*

logout(*self*)

Log out

getUserInfo(*self*)

Returns user information

4 Module astrogrid.cone

Send a cone search query to a service.

A cone search is a query of a catalogue for objects around a position in sky. This module implements the basic cone search, given a service provider and the coordinates and radius of the area in sky, returns a list of objects.

4.1 Class ConeSearch

The following example sends a cone search query to NED and saves the resulting VOTable in the local disk.

```
>>> from astrogrid import ConeSearch
>>> cone = ConeSearch("ivo://ned.ipac/Basic_Data_Near_Position")
>>> print cone.info['content']['description']
>>> result = cone.execute(242.811, 54.596, 0.1)
>>> open("ned.vot",'w').write(result)
```

4.1.1 Methods

__init__(*self*, *service*)

Parameters

`service`: URI of service to be queried (e.g. "ivo://ned.ipac/Basic_Data_Near_Position") *(type=str)*

execute(*self*, *ra*, *dec*, *radius*, *saveAs*=`None`, *clobber*=`False`)

Execute the cone search.

Parameters

`ra`: R.A. in degrees *(type=float)*
`dec`: Dec in degrees *(type=float)*
`radius`: Radius in degrees *(type=float)*
`saveAs`: Saves the query to a file in MySpace. Default: None *(type=str)*
`clobber`: Overwrites file if it exists (takes priority over configuration file) *(type=bool)*

Return Value

res (**str**)
VOTable as a string or the name of the output file if `saveAs` was used.

4.1.2 Instance Variables

Name	Description
info	Information about the service

5 Module astrogrid.helpbrowser

5.1 Functions

open(*filename*)

aghelp()

6 Module astrogrid.myspace

Python interface to MySpace

6.1 Class MySpace

Perform operations in MySpace.

```
>>> from astrogrid import MySpace
>>> m = MySpace()
>>> m.ls()    # list contents
>>> m.rm('test8', recursive=True)  # recursively remove the test8 directory
```

Saving data to MySpace:

```
>>> m.savefile('image.fits', '#images/image.fits')
>>> m.savefile('http://www.example.com/image.fits', '#images/image.fits', clobber=True)
>>> m.savefile(open('image.fits').read(), '#images/image.fits')
```

6.1.1 Methods

__init__(*self*)

home(*self*)

Return the home IVORN

ls(*self*, *directory*=None, *pprint*=True)

List contents of directory

readfile(*self*, *ivorn*, *ofile*=None)

Read a file from MySpace and optionally saves the content to local disk.

Parameters

`ivorn`: File name in MySpace. It can be the full ivorn (i.e. 'ivo://ukidss.roe....') or the path from the root (i.e. '#sdss/dr3qso.vot') *(type=str)*

`ofile`: Output file in local disk. Useful for large files since they are not stored in memory prior to being saved. There is no checking that the file already exists. *(type=str)*

savefile(*self*, *data*, *ivorn*, *clobber*=`False`)

Saves data into MySpace.

Parameters

`data:` Data string to copy. It can be a http or ftp reference, a local file or the data itself.

`ivorn:` File name in MySpace. It can be the full ivorn (i.e. 'ivo://ukidss.roe....') or the path from the root (i.e. '#sdss/dr3qso.vot') *(type=str)*

`clobber:` Clobber exiting file. Default: False *(type=bool)*

Return Value

result (**bool**)

True if operation succeeded.

rm(*self*, *ivorn*, *recursive*=`False`)

Remove a file or directory from MySpace.

Parameters

`ivorn:` File or directory in MySpace. It can be the full ivorn (i.e. 'ivo://ukidss.roe....') or the path from the root (i.e. '#sdss/dr3qso.vot') *(type=str)*

`recursive:` If True then directories are removed recursively. *(type=bool)*

convertfile(*self*, *ifile*, *ofile*, *ifmt*=`'votable'`, *ofmt*=`'fits'`, *clobber*=`False`)

Convert one file in MySpace to another format (e.g. useful to convert from votable to fits and then download the fits).

Parameters

`ifile:` Input file *(type=str)*

`ofile:` Output file *(type=str)*

`ifmt:` Input file format (votable|fits|csv). Default: votable *(type=str)*

`ofmt:` Output file format (votable|fits|csv). Default: fits *(type=str)*

`clobber:` Clobber existing file. Default: False *(type=bool)*

Return Value

result (**bool**)

True if operation succeeded.

7 Module astrogrid.plastic

7.1 Functions

mkURI(*input*)

listPlasticApps()

getId(*app*)

broadcast(*vot*, *app*=`None`)

Sends a votable to all listening PLASTIC applications

Parameters

`vot:` table to broadcast. It can be a string containing the votable or a file in MySpace.

`app:` application to send to (e.g. topcat). Default sends to all listening applications.

Return Value

Returns True if succeeded, False otherwise.

8 Module astrogrid.registry

Module to send queries to query the registry.

8.1 Class Registry

Perform queries on the registry

8.1.1 Methods

__init__(*self*)

__getitem__(*self*, *id*)

endpoint(*self*)

Returns the IVORN of the registry endpoint

keywordSearch(*self*, *keywords*, *orValues*=`False`)

Performs a keyword search on the registry

search(*self*, *keywords*, *orValues*=`False`)

Performs a keyword search on the registry

searchCone(*self*, *key*=`None`)

Return all services which provide a cone interface

searchSiap(*self*, *key*=`None`)

Return all services which provide a siap interface

searchStap(*self*, *key*=`None`)

Return all services which provide a stap interface

9 Module astrogrid.siap

Send queries to a siap server.

9.1 Class SiapSearch

Execute SIAP searches.

The following example sends a cone search query to NED and saves the resulting VOTable in the local disk.

```
>>> from astrogrid import SiapSearch
>>> siap = SiapSearch('ivo://roe.ac.uk/services/SIAPDR4-images')
>>> print siap.info['content']['description']
>>> result = siap.execute(180.0, 2.0, 1.0)
>>> open('siapdr4.vot','w').write(result)
```

The following example saves the images from the query to a directory in MySpace. The execute query returns in this case a Thread instance that can be queried for completion.

```
>>> res, thread = siap.execute(180.0, 2.0, 0.1, saveDatasets='#sdss/images/')
>>> thread.isAlive()
True
>>> thread.isAlive()
False
```

9.1.1 Methods

__init__(*self*, *service*)

Parameters

`service:` URI of service to be queried (e.g. 'ivo://roe.ac.uk/services/SIAPDR4-images') *(type=str)*

`debug:` Print debugging information. Default = False *(type=bool)*

execute(*self*, *ra*, *dec*, *radius*, *format*=`'ALL'`, *saveAs*=`None`, *clobber*=`False`, *saveDatasets*=`None`)

Execute the cone search.

Parameters

`ra:` R.A. in degrees *(type=float)*

`dec:` Dec in degrees *(type=float)*

`radius:` Radius in degrees *(type=float)*

`saveAs:` Saves the query to a file in MySpace. Default: None *(type=str)*

`clobber:` Overwrites file if it exists (takes priority over configuration file) *(type=bool)*

`saveDatasets:` If the name of a directory in MySpace is specified saves asynchronously the files of the siap query. *(type=str)*

Return Value

res (**str**)
VOTable as a string or the name of the output file is `saveAs` was used.

thread If saveDatasets then returns the thread so it can be queried for completion.

9.1.2 Instance Variables

Name	Description
info	Information about the service

10 Module astrogrid.stap

Send queries to a stap server.

10.1 Class StapSearch

Execute STAP searches.

The following example sends a cone search query to NED and saves the resulting VOTable in the local disk.

```
>>> from astrogrid import StapSearch
>>> stap = StapSearch('ivo://mssl.ucl.ac.uk/stap/vso/eit')
>>> print stap.info['content']['description']
>>> result = stap.execute(180.0, 2.0, 1.0)
>>> open('stap_eit.vot','w').write(result)
```

The following example saves the images from the query to a directory in MySpace. The execute query returns in this case a Thread instance that can be queried for completion.

```
>>> res, thread = siap.execute(180.0, 2.0, 0.1, saveDatasets='#sdss/images/')
>>> thread.isAlive()
True
>>> thread.isAlive()
False
```

10.1.1 Methods

__init__(*self*, *service*)

Parameters

`service:` URI of service to be queried (e.g. 'ivo://mssl.ucl.ac.uk/stap/vso/eit') *(type=str)*

`debug:` Print debugging information. Default = False *(type=bool)*

execute(*self*, *start*, *end*, *format*=`'ALL'`, *saveAs*=`None`, *clobber*=`False`, *saveDatasets*=`None`)

Execute the cone search.

Parameters

`start:` Start date and time *(type=datetime)*

`end:` Start date and time *(type=datetime)*

`saveAs:` Saves the query to a file in MySpace. Default: None *(type=str)*

`clobber:` Overwrites file if it exists (takes priority over configuration file) *(type=bool)*

`saveDatasets:` If the name of a directory in MySpace is specified saves asynchronously the files of the siap query. *(type=str)*

Return Value

res (**str**)
VOTable as a string or the name of the output file is `saveAs` was used.

thread If saveDatasets then returns the thread so it can be queried for completion.

10.1.2 Instance Variables

Name	Description
info	Information about the service

11 Module astrogrid.system

Module to send queries to query the registry.

11.1 Class Configuration

11.1.1 Methods

__init__(*self*)
Overrides: UserDict.UserDict.__init__

__setitem__(*self*, *key*, *value*)

__cmp__(*self*, *dict*)

__contains__(*self*, *key*)

__delitem__(*self*, *key*)

__getitem__(*self*, *key*)

__len__(*self*)

__repr__(*self*)

__setitem__(*self*, *key*, *item*)

clear(*self*)

copy(*self*)

fromkeys(*cls*, *iterable*, *value*=`None`)

get(*self*, *key*, *failobj*=`None`)

has_key(*self*, *key*)

items(*self*)

iteritems(*self*)

iterkeys(*self*)

itervalues(*self*)

keys(*self*)

pop(*self*, *key*, **args*)

popitem(*self*)

setdefault(*self*, *key*, *failobj*=`None`)

update(*self*, *dict*=`None`, ***kwargs*)

values(*self*)

12 Module astrogrid.utils

Several applications.

12.1 Functions

mkURI(*input*)

read_votable(*table*, *ofmt*=`'votable'`)

Read a votable string or local file and returns a VOTable or FITS structure.

Parameters

`table:` string containing the votable or file in the local disk. If the file is in MySpace, first read the file and then use this routine. *(type=str)*

`ofmt:` Output format (votable|fits). Default: votable *(type=str)*

tput(*ifile*, *ofile*, *ifmt*=`'votable'`, *ofmt*=`'votable'`)

browserview(*vot*)

Displays a votable as an HTML document in the system browser

Parameters

`vot:` votable as a string

tmatch2(*in1*, *in2*, *out*, *matcher*=`'1and2'`)

Cross match two tables (stored in MySpace)

Parameters

`in1:` input table 1 *(type=str)*

`in2:` input table 2 *(type=str)*

`out:` output tble *(type=str)*

`matcher:` type of match *(type=str)*

12.2 Variables

Name	Description
has_pyfits	**Value:** `False`

Index