gsf Documentation

Release 1.0.0

Eric Lefebvre

Jul 17, 2017

Contents

GSF Py provides a client Python package, named gsf, to run IDL and ENVI analytics provided by the Geospatial Services Framework. The Python package provides the ability to query for available tasks, retrieve task information, and submit jobs to the GSF server.

There is an additional client Python package available, named gsfarc, to provide the ability to run GSF analytics through ArcMap, ArcGIS Pro, and ArcGIS Server.

See http://www.harrisgeospatial.com/ for more details on product offerings.

CHAPTER 1

Usage

Demonstration of connecting to GSF Server and submitting a job.

Note: If not executing the Python client code on same system as the server, replace *localhost* with the server name.

To connect to GSF and list the available services, create a new instance of the GSF Server class with the URL to the server from the Python command line:

```
>>> from gsf import Server
>>> server = Server('localhost','9191')
>>> server.services()
```

To get a dictionary of service information including a list of tasks, use the `service()` method on the Server object:

```
>>> envi_service = server.service('ENVI')
>>> envi_service.tasks()
```

To get a GSF task object, use the `task()` method on the Service object:

```
>>> task = envi_service.task('SpectralIndex')
```

To get a list of task parameter information, use the `parameters` property on the Task object:

```
>>> task.parameters
```

To run a task asynchronously, use the `submit()` method on the Task object. A GSF Job object is returned after the job has been submitted.

Debug Tip: To see if GSF server received the job and check job status, go to http://localhost:9191/job-console/

```
>>> input_raster = dict(url='http://localhost:9191/ese/data/qb_boulder_msi',
                        factory='URLRaster')
>>> parameters = dict(INPUT_RASTER=input_raster,
                      INDEX='Normalized Difference Vegetation Index')
>>> job = task.submit(parameters)
>>> job.wait_for_done()
```

CHAPTER 2

API Documentation

GSF Server

The GSF server object is used to connect to the server and retrieve information about available services and jobs.

class gsf.server.**Server**(*server=None*, *port='9191'*)

The GSF server connection class.

Example

Import the modules for the example.

```
>>> from gsf import Server
```

Connect to the GSF server and print information.

```
>>> server = Server('localhost','9191')
>>> print(type(server))
<class 'gsf.ese.server.Server'>
>>> print(server.name, type(server.name))
('localhost', <type 'str'>)
>>> print(server.port, type(server.port))
('9191', <type 'str'>)
```

Investigate available services.

```
>>> services = server.services()
>>> print(services, type(services))
([u'IDL', u'ENVI'], <type 'list'>)
```

Investigate jobs on the GSF server.

```
>>> job = server.job(1)
>>> print(job.results)
```

job(*job_id*)

Returns the GSF Job object based on the job_id. See GSF Job for more information.

Parameters **job_id** – The job_id for which to retrieve job information.

Returns GSF Job object

name

Returns the server hostname.

Returns a string

port
Returns the server port number.

Returns a string

service(*service_name*)
Returns the GSF Service object based on the service_name. See GSF Service for example.

Parameters **service_name** – The service to connect to.

Returns GSF Service object

services()
Returns a list of available services.

Returns a list

GSF Service

The GSF service object connects to a GSF Service and its tasks.

class gsf.service.**Service**
The GSF Service connection class.

Example

Import the modules for the example.

```
>>> from gsf import Server
>>> from pprint import pprint
```

Connect to the GSF server and the retrieve the ENVI service.

```
>>> server = Server('localhost','9191')
>>> service = server.service('ENVI')
>>> print(type(service))
<class 'gsf.ese.service.Service'>
```

Investigate service information.

```
>>> print(service.description, type(service.description))
('ENVI processing routines', <type 'str'>)
>>> print(service.name, type(service.name))
('ENVI', <type 'str'>)
>>> tasks = service.tasks()
>>> pprint(tasks)
['AdditiveLeeAdaptiveFilter',
 'AdditiveMultiplicativeLeeAdaptiveFilter',
 'ApplyGainOffset',
     ...
```

description
Returns a description of the service

Returns a string

name
Returns the name of the service

Returns a string

task(*task_name*)
Returns a GSF task object. See GSF Task for example.

Param task_name: The name of the task to retrieve.

Returns a GSF Task object

tasks()
Returns a list of task names available on this service

Returns a list

GSF Task

The GSF task object provides task information and can submit a job to the GSF server with input.

class gsf.task.**Task**(*uri=None*)
The GSF Task object connects to a GSF Task and its parameters.

Example

Import the modules for the example.

```
>>> from gsf import Server
>>> from pprint import pprint
```

Connect to the GSF server, and then retrieve the SpectralIndex task.

```
>>> server = Server('localhost','9191')
>>> service = server.service('ENVI')
>>> task = service.task('SpectralIndex')
>>> print(type(task))
<class 'gsf.ese.task.Task'>
```

Investigate task information.

```
>>> print(task.uri, type(task.uri))
('http://localhost:9191/ese/services/ENVI/SpectralIndex', <type 'str'>)
>>> print(task.description, type(task.description))
('This task creates a spectral index raster from one pre-defined spectral
index. Spectral indices are combinations of surface reflectance at two
or more wavelengths that indicate relative abundance of features of
interest.', <type 'str'>)
>>> print(task.display_name, type(task.display_name))
('Spectral Index', <type 'str'>)
>>> task_parameters = task.parameters
>>> pprint(task_parameters)
```

Submit a job to the GSF Server. See GSF Job for details on waiting for job to complete.

```
>>> input_raster = dict(url='http://localhost:9191/ese/data/qb_boulder_msi',
                        factory='URLRaster')
>>> parameters = dict(INPUT_RASTER=input_raster,
                      INDEX='Normalized Difference Vegetation Index')
>>> job = task.submit(parameters)
>>> print(type(job))
<class 'gsf.ese.job.Job'>
```

description
The task description

Returns a string

display_name
The display name of the task

Returns a string

name
The name of the task

Returns a string

parameters
A list of the task parameter definitions. Each task parameter is a dictionary containing, but not limited to, the following keys:

Key	Data Type	Type	Description
name	string	Required	The name of the parameter
display_name	string	Required	The display name of the parameter
type	string	Required	The parameter data type
direction	string	Required	Can be *input* or *output*
description	string	Required	The parameter description
required	bool	Required	Indicates if the parameter is required on input when submitting a job
dimensions	string	Optional	Indicates if the parameter is an array if set. Dimesions is of the format *[dim1,dim2,...]*
choice_list	list	Optional	A list of available choices for the parameter input
min	type	Optional	The minimum value allowed for the parameter
max	type	Optional	The maximum value allowed for the parameter

Returns a list of parameter dictionaries

submit(*parameters*)
Submits a task to the server asynchronously

Parameters **parameters** – A dictionary of key-value pairs of parameter names and values. The dictionary serves as input to the job.

Returns GSF Job object

uri
The unique identifier of this task

Returns a string

GSF Job

A GSF job object is returned from the GSF Task submit method and is used to query job status and retrieve job results.

class gsf.job.**Job**

A GSF Job object connects to a GSF Job and its status.

Example

Import the modules for the example.

```
>>> from gsf import Server
>>> from pprint import pprint
```

Connect to the GSF server, retrieve the SpectralIndex task, and submit a job.

```
>>> server = Server('localhost','9191')
>>> service = server.service('ENVI')
>>> task = service.task('SpectralIndex')
>>> input_raster = dict(url='http://localhost:9191/ese/data/qb_boulder_msi',
                        factory='URLRaster')
>>> parameters = dict(INPUT_RASTER=input_raster,
                      INDEX='Normalized Difference Vegetation Index')
>>> job = task.submit(parameters)
```

Wait for job to be done.

```
>>> job.wait_for_done()
```

Investigate job information and results.

```
>>> print(job.job_id, type(job.job_id))
(42, <type 'int'>)
>>> print(job.status, type(job.status))
('Succeeded', <class 'str'>)
>>> print(job.results,type(job.results))
({ 'OUTPUT_RASTER': { 'auxiliary_url': [ 'http://localhost:9191/ese/jobs/42/envitempfileFriJul15
                  'factory': 'URLRaster',
                  'url': 'http://localhost:9191/ese/jobs/42/envitempfileFriJul151817462016242_1.d
```

Submit a job that will fail.

```
>>> input_raster = dict(url='http://localhost:9191/ese/data/doesnotexist',
                        factory='URLRaster')
>>> parameters = dict(INPUT_RASTER=input_raster,
                      INDEX='Normalized Difference Vegetation Index')
>>> job = task.submit(parameters)
```

Wait for job to be done.

```
>>> job.wait_for_done()
```

Investigate job information and results.

```
>>> print(job.job_id)
43
>>> print(job.status)
Failed
>>> print(job.results)
{}
```

```
>>> print(job.error_message)
'Invalid value for parameter: INPUT_RASTER. Error: File: ...
```

error_message
Returns the job error message

Returns a string

job_id
Returns the job_id

Returns an integer greater than 0

progress
Returns a percentage of job completion

Returns an integer between 0 and 100

progress_message
Returns the job progress message

Returns a string

results
Returns the output parameter results as a dictionary where each key is the output parameter name. The value depends on the parameter type.

Returns a dictionary

status
Returns the current job status. Status can be "Succeeded | Failed | Accepted | Started"

Returns a string

wait_for_done()
Blocks execution until the job status message is either Succeeded or Failed.

Returns None

GSF Errors

Defines exceptions for the GSF package.

exception gsf.error.**JobNotFoundError**
Exception gets raised when the user has passed in an incorrect job number.

Example

```
>>> from gsf import Server
>>> server = Server('localhost','9191')
>>> job = server.job(-1) # job_id must be integer greater than 0
>>> print(job.status)
# traceback information
gsf.error.JobNotFoundError: HTTP code: 404, Reason: Not Found
```

exception gsf.error.**ServerNotFoundError**
Exception gets raised when the user has passed in an incorrect net location (host:port).

Example

```
>>> from gsf import Server
>>> server = Server('doesnotexist','9191')
>>> print(server.services())
# traceback information
gsf.error.ServerNotFoundError: [Errno 11004] getaddrinfo failed
```

exception gsf.error.**ServiceNotFoundError**

Exception gets raised when the user has passed in an incorrect service name.

Example

```
>>> from gsf import Server
>>> server = Server('localhost','9191')
>>> service = server.service('doesnotexist')
>>> print(service.name)
# traceback information
gsf.error.ServiceNotFoundError: HTTP code 400, Reason: Bad Request
```

exception gsf.error.**TaskNotFoundError**

Exception gets raised when the user has passed in an incorrect task name.

Example

```
>>> from gsf import Server
>>> server = Server('localhost','9191')
>>> service = server.service('ENVI')
>>> task = service.task('doesnotexist')
>>> print(task.name)
# traceback information
gsf.error.TaskNotFoundError: HTTP code 404, Reason: Not Found
```

g