giosystemcore Documentation

Release 0.1

Ricardo Silva

November 28, 2016

Contents

giosystem is the name of the software system used for managing the processing lines and related functionality of the IPMA Copernicus Global Land project. It consists of serveral components:

giosystemcore giosystemcore is a package that facilitates working with files and algorithms used in the IPMA GIO-GL processing lines.

giosystem-django-base A django project that forms a base for installing the various web applications that are part of giosystem. It includes a django-pywps, which is a WPS server used to process IPMA Copernicus GL products on demand.

Introduction to giosystem-settings-django A web application that is used to store settings for the processing lines. These settings include hosts, file paths and patterns, processing packages, processing line execution dependencies, etc.

It provides a web accessible API that can be used to query the system's settings programatically. This API is used by *giosystemcore* for creating all of the specialized classes.

`giosystemdjangobase/downloads` A web application that provides individual products, quicklooks, etc to web clients on demand. It responds to HTTP requests, delivering the products to the requesting clients.

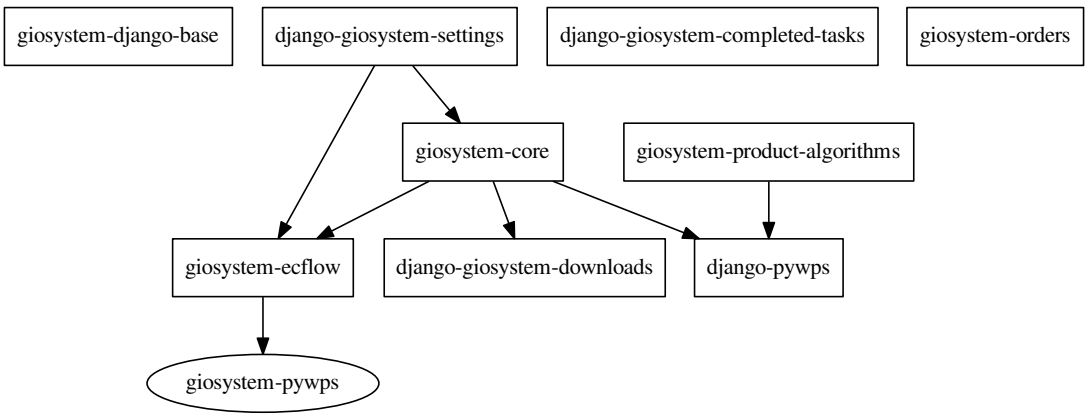
`giosystemdjangobase/operations` A web application that stores information on the various tasks performed by the processing lines. It stores information on the execution status and results of the tasks defined in the processing lines.

Giosystem processing A group of algorithms implemented in Fortran and C++ that perform the actual product generation. These algorithms are available as individual projects. The installation of these algorithms into the giosystem framework is managed and configured in the giosystem-processing component. Each algorithm is managed individually in its own repository on bitbucket.

giosystem-ecflow ecFlow processing lines. This package interfaces with the ecflow software, managing its configuration for the processing lines of the giosystem.

Giosystem-orders An ordering server (OGC OSEO) that responds to order requests. It interfaces with the external GIO-GL portal and is responsible for managing and processing order and subscription requests for the products generated at IPMA's processing lines.

These components interact with each other, forming the whole giosystem big picture. The following diagram provides a general overview on this interaction.



Contents:

CHAPTER 1

giosystemcore

giosystemcore is a package that facilitates working with files and algorithms used in the IPMA GIO-GL processing lines. It is usable by itself, but the real advantage of this package is when it is used together with the *django-giosystem-settings* web application.

It is a set of python modules that abstract the details of product generation, making it easy to generate IPMA GIO-GL products.

For example, if we want to fetch all the inputs to the 'preprocess_goes_lrit' package into a working directory on our local filesystem:

```
import giosystemcore.packages.packagefactory as factory
from giosystemcore.settings import get_settings

# only works inside IPMA's network
get_settings('http://gio-gl.meteo.pt/giosystem/settings/api/v1/')
package = factory.get_package('preprocess_goes_lrit', '201402011000')
fetched = package.fetch_inputs()
print(fetched)
```

1.1 Installation

Detailed installation instructions are available at giosystemcore's source code repository at http://bitbucket.org/ipmagio/giosystem-core

1.2 Core modules

1.2.1 settings

The *giosystemcore.settings* module interfaces with the django-giosystem-settings web application fetching the settings needed to configure the various objects.

When using giosystemcore, the first step will typically be the calling of *giosystemcore.settings.get_settings()* function, in order to establish a connection to the settings that we want to use.

```
from giosystemcore.settings import get_settings
get_settings('http://gio-gl.meteo.pt/giosystem/settings/api/v1/')
# now we are ready to rock!
```

1.2.2 files

A `giosystemcore.files.GioFile` represents a file used in the processing lines. Files are used by packages as inputs and outputs. They are present in hosts. A file has a *search_pattern* and a *search_path* which can be used to look for it in the defined hosts. Files can be configured in the *giosystem-settings-django* administration backend. The operational settings are available at

http://gio-gl.meteo.pt/giosystem/settings/admin/coresettings/files

The `giosystemcore.files` module provides an abstraction to work with the different file types used and generated in the giosystem.

It defines the `giosystemcore.files.GioFile` class, that has properties and methods useful for finding, fetching and sending files to/from multiple hosts.

Creating instances of *GioFile* should be done through the `giosystemcore.files.get_file()` factory function.

```
import giosystemcore.files
f = giosystemcore.files.get_file('', '201402071300')
```

1.2.3 packages

Giosystem relies on a set of specialized code routines for generating GIO products. These routines are defined as giosystem packages.

Packages represent execution units in the processing lines. They are used to generate products. Packages usually have inputs and outputs, which are GioFile instances. Packages have several methods for fetching their inputs, executing the product generation code and moving the outputs to designated hosts.

Packages should be instantiated by using the `giosystemcore.packages.packagefactory.get_package()` function. It will create the package using the settings defined in the giosystem-settings-django application.

The following example demonstrates creating a package for processing the *process_goes_lst* package for the *2014-03-15 09:00:00* timeslot:

```
import giosystemcore.settings
import giosystemcore.packages.packagefactory as pf

settings_url = 'http://gio-gl.meteo.pt/giosystem/coresettings/api/v1/'
giosystemcore.settings.get_settings(settings_url)
p = pf.get_package('process_goes_lst', '201403150900')
p.use_io_buffer_for_searching = False
p.use_archive_for_searching = False
p.copy_outputs_to_io_buffer = False
p.copy_outputs_to_archive = False
result, details = p.run()
```

1.2.4 hosts

A `giosystemcore.hosts.hosts` represents a host machine used in the giosystem processing lines. Hosts have methods for interacting with their respective filesystems, including ways to fetch files.

host_roles

Some hosts have special roles in the context of the giosystem processing lines. There is an *archive* host that holds the archive of IPMA's GIO-GL products. There is also a *web_server* host that has the public giosystem webservices installed.

The *giosystemcore.hosts* package provides several host classes. These are an abstraction of the real host machines in use by the giosystem and provide a number of useful methods for dealing with file copy and editing.

1.3 API

1.3.1 giosystemcore.settings

A module for reading settings stored in an online REST API.

This module provides a global interface for accessing giosystem settings stored in the giosystem-settings-django app, which is another component of giosystem.

Usually all that is necessary is to instantiate a *SettingsManager* object, providing it with the correct URL for the giosystem-settings-django app. This is achieved by calling the module's *get_settings()* function.

```
import giosystemcore.settings

settings_url = 'http://geo2.meteo.pt/giosystem/settings/api/v1/'
giosystemcore.settings.get_settings(settings_url)
```

class giosystemcore.settings.**SettingsManager**(*url*, *initialize_logging=True*)

Read the settings stored online in the django app.

Settings are fetched when needed instead of all in one go.

get_all_areas_settings()

Return the settings for all of the defined areas

This method will only query the online REST API if its cache is empty.

get_all_ecflow_servers_settings()

Return the settings for all of the defined ecflow servers

This method will only query the online REST API if its cache is empty.

get_all_files_settings()

Return the settings for all of the defined GioFiles

This method will only query the online REST API if its cache is empty.

get_all_hosts_settings()

Return the settings for all of the defined hosts

This method will only query the online REST API if its cache is empty.

get_all_organizations_settings()

Return the settings for all of the defined organizations

This method will only query the online REST API if its cache is empty.

get_all_packages_settings()

Return the settings for all of the defined GioPackages

This method will only query the online REST API if its cache is empty.

get_all_palettes_settings()
Return the settings for all of the defined palettes

This method will only query the online REST API if its cache is empty.

get_all_products_settings()
Return the settings for all of the defined products

This method will only query the online REST API if its cache is empty.

get_all_sources_settings()
Return the settings for all of the defined sources

This method will only query the online REST API if its cache is empty.

get_archive_settings()
Get the specific settings for the archive host.

get_data_receiver_settings()
Get the specific settings for the data receiver host.

get_file_settings(*name*)
Return the settings for the file.

get_ftp_server_settings()
Get the specific settings for the ftp server host.

get_host_settings(*name*)
Return the settings for the host.

get_io_buffer_settings()
Get the specific settings for the io buffer host.

get_order_server_settings()
Get the specific settings for the ordering server host.

get_package_settings(*name*)
Return the settings for the package.

get_product_settings_by_name(*short_name*)
Get product settings given its short name.

get_source_settings_by_name(*name*)
Get source settings given its name.

get_suite_settings_by_name(*name*)
Get a suite's settings given its name.

get_web_server_settings()
Get the specific settings for the web server host.

giosystemcore.settings.**get_settings**(*base_url=None*, *initialize_logging=True*)
Return the settings manager.

If the settings manager has already been created by a previous call, it will be returned. If not, the base_url argument must be provided so that a new settings manager can be created from the settings present at the URL. This ensures that multiple calls to this function will not trigger unneeded network traffic.

Parameters **base_url** (*str*) – URL to query the REST API and get the defined settings. A value of None will assume that the previously cached settings should be returned instead.

CHAPTER 2

giosystem-django-base

The giosystem includes several web applications:

- django-giosystem-settings
- django-giosystem-completed-tasks
- django-giosystem-web
- django-pywps

As indicated by their names, all these applications are django applications. As such, they must be installed inside a django project. Giosystem-django-base is a predefined django project suitable for installing all these apps.

It can be installed in each of the machines that are part of the giosystem network and configured to include the necessary apps.

2.1 Installation

Detailed installation instructions are available at the projects source code repository:

http://bitbucket.org/ipmagio/giosystem-django-base

CHAPTER 3

giosystem-ecflow

Giosystem-ecflow is a python package that manages the creation of ecflow servers and suites.

ecflow is a software that allows defining complex workflows, grouping them in suites, families and tasks and scheduling them for execution according to predefined rules.

The giosystem-ecflow library hooks up with the online giosystem-settings-django app and uses the information on the defined servers and suites to build the ecflow structure for the processing lines

3.1 Installation

Detailed installation instructions are available at the projects source code repository

3.2 Defining ecflow servers and suites

Ecflow servers and suites are defined in the giosystem-settings-django app. The settings are available at:

http://<operational_server>/giosystem/admin

There are three sections of the settings administration area that are used to configure suites and ecfflow servers:

1. packages - Packages define some settings that are used inside suites:
2. ecflow_servers
3. suites

3.3 Custom suite variables

Giosystem generated suites have some specific variables that are automatically generated by the giosystemecflow code.

- COMPRESS_OUTPUTS
- COLLABORATORS_TO_EMAIL
- COPY_OUTPUTS_TO_ARCHIVE
- COPY_OUTPUTS_TO_IO_BUFFER
- ECF_MICRO
- EXECUTION_HOST

- EXTRA_HOSTS_TO_COPY_OUTPUTS
- OPS_DATABASE_API_KEY
- OPS_DATABASE_URL
- OPS_DATABASE_USER
- PYTHON_COMMAND
- PYTHON_PATH
- REMOVE_PACKAGE_WORKING_DIR
- SERVER_NAME
- SETTINGS_URL
- URLBASE
- USE_ARCHIVE_FOR_SEARCHING
- USE_IO_BUFFER_FOR_SEARCHING
- PAKAGE

3.4 Custom suite families and tasks

Giosystemecflow generated servers and suites automatically add some families and tasks to each server or suite.

- /server_maintenance
- /<suite_name>/maintenance
- /<suite_name>/archive
- /<suite_name>/clean

3.5 Starting suites

Execute the start_ecflow_servers.py script that is available after installation of the giosystemecflow package. This script will take care of starting the predefined servers and suites. When using the script you will have to provide the URL to the settings that you wish to use.

```
~/dev/venv/bin/python ~/dev/venv/bin/start_giosystem_servers.py <settings_url>
```

The script starts the server and loads the suites but it **does not start them**. Use the *ecflowview* GUI app to access the server and manually start the suite.

3.6 Editing suites

Edit the online settings, save them and then run the *reload_suite* task in the affected suite. It will be reloaded and fetch the new settings.

3.7 Monitoring suites

Use the *ecflowview* GUI app to monitor suites execution in realtime.

CHAPTER 4

Introduction to giosystem-settings-django

Giosystem-settings-django is a django application that stores the settings for the giosystem. It has an administration web interface that can be used to edit the settings. It also exposes a REST API that can be queried programatically in order to retrieve the settings from any host. This makes it possible to use the same settings for all the giosystem hosts.

The settings app can be installed in any host. This provides a convenient way to both having:

- A set of operational settings, available on a secure host. These settings are used by all hosts that execute product generating packages in the main processing lines. They are rarely edited and access controlled, providing a stable and secure environment for production.
- Other settings instances, installed in development machines, that can be used for testing

Therefore, when working with the giosystem code, we must decide on what settings app will be used and provide the URL to its REST API to the `giosystemcore.settings.get_settings()` function.

4.1 Operational settings host

Currently the operational settings are available at the gio-gl host (only accessible from within IPMA's private network).

> http://gio-gl.meteo.pt/giosystem/settings

In order to access the administration interface, a specific user account must be used. Authentication credentials are available internally at GIO-GL's web based collaboration tool

4.2 Installation

Detailed installation instructions are available at the projects source code repository

4.3 Editing the settings

The settings can be inspected and edited using the online administration interface. It is available at http://gio-gl.meteo.pt/giosystem/settings/admin/

4.4 REST API

The settings can be queried by using the API provided at http://gio-gl.meteo.pt/giosystem/settings/api/v1/ This API is generated using the django tastypie project.

CHAPTER 5

Giosystem processing

The giosystemprocessing package handles remotely calling a giosystemcore package through the OGC Web Processing Service (WPS).

It is also responsible for installing the external Fortran and C/C++ code algorithms that implement the product generation logic.

For WPS, it uses pywps as a backend.

5.1 Installation

Detailed installation instructions are available at the package's source code repository at http://bitbucket.org/ipmagio/giosystem-processing

5.2 Installing the external code

Use the supplied scripts

5.3 Usage

After installation, the server should be accessible at the url http://<server_address>/giosystem/processing/wps

5.4 Web Processing Service

Web Processing Service is a standard for the execution of geospatial processing services. It defines an interface that a client can use to communicate with a processing server over the Internet and ask for a specific process to be executed.

5.5 Processes

The following processes are implemented by giosystem-processing:

5.5.1 gio_process

This process executes a package defined in giosystem-settings-django.

It accepts as inputs:

- **package name**
- **settings uri**
- **timeslot**
- compress outputs
- copy outputs to archive
- copy outputs to io buffer
- remove package working dir
- use archive for searching
- use io_buffer for searching
- extra hosts to copy outputs

And returns as outputs:

- output result
- output details

5.5.2 gio_archive_process

This process executes the archiving part of a package defined in giosystem-settings-django.

Note: add a process for the cleaning up of package outputs

CHAPTER 6

Giosystem-orders

Giosystem includes an ordering service that processes order requests for products of the processing lines.

The ordering service uses the following components:

- An *OGC OSEO server*
- An *Order database* where order information is stored
- A *User authentication module* that implements custom auth
- An *Order processing queue* that is used to manage order requests
- An *Order processing module* module, that uses giosystemcore

6.1 OGC OSEO server

The ordering service implements a subset of the OGC OSEO specification, which defines a standard for ordering earth observation products.

The giosystem-orders server is based on the pyoseo project, while incorporating some custom logic in order to work with the giosystem processing lines.

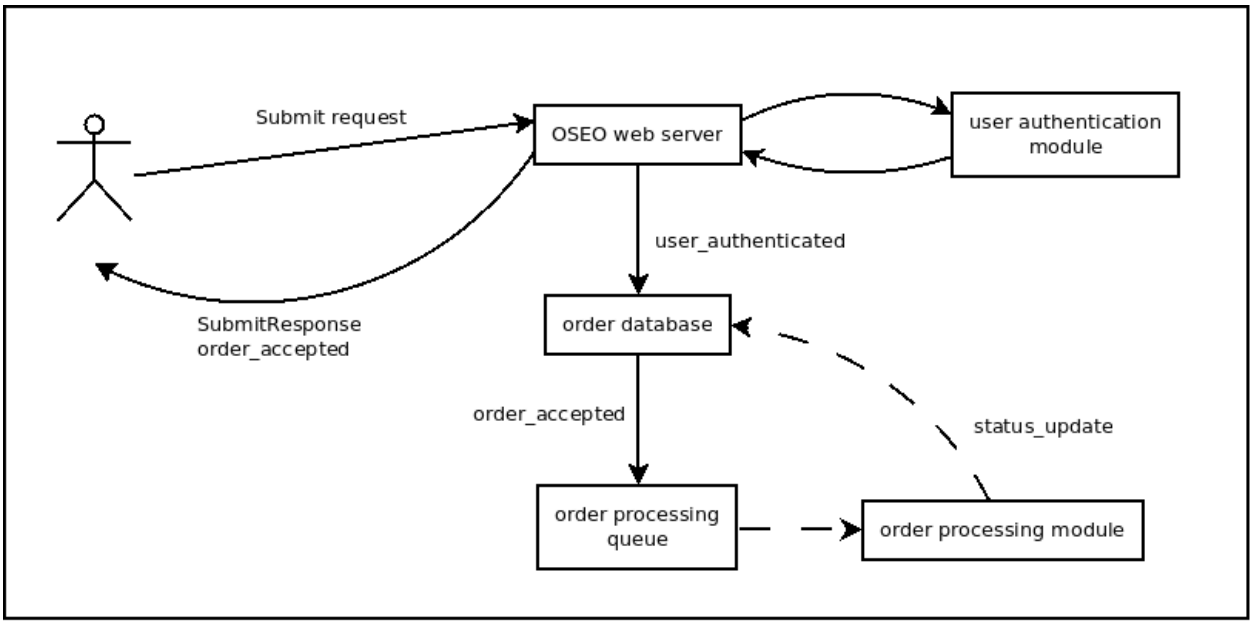
Note: The OSEO server has the following restrictions:

- Implements only the following OSEO operations:
  - Submit
  - GetStatus
  - DescribeResultAccess
  - GetOptions
- Does not implement the asynchronous interface for the Submit operation
- Does not support FTP push as a delivery method

6.1.1 Supported OSEO operations

Submit

The Submit operation allows a client to place an order on the server. Upon having been successfully authenticated, the order is placed in the order database and is sent to the processing queue, where it will be processed in due time. The following image provides a rough workflow of the Submit operation:

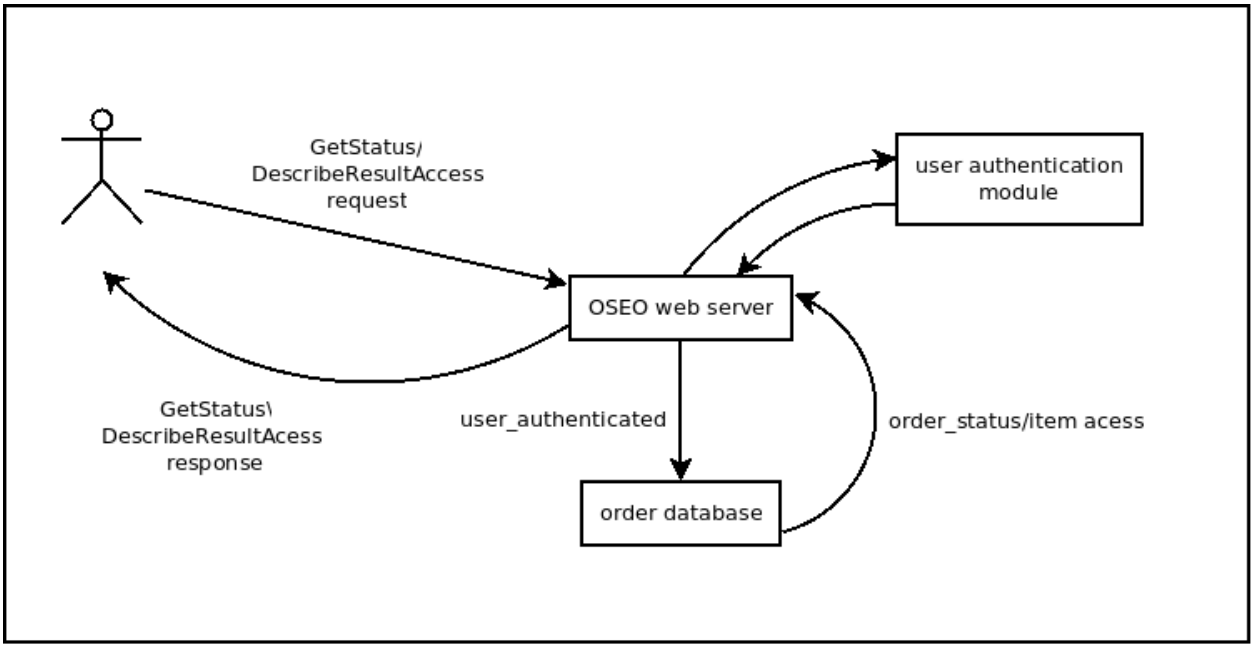


Since the server does not support the OSEO asynchronous interface, the response to a Submit request is sent only once, immediately after the order has been requested. Further status changes must be explicitly asked for by the user, by issuing Getstatus requests.

Example requests and further information on the Submit operation is available at http://pyoseo.readthedocs.org/en/latest/submit.html

GetStatus

The Getstatus operation allows a client to ask the server what is the status of an order. The server queries its order database and returns the status information back to the client.



As orders get processed by the queue, their status is dynamically updated. After an order has been successfully downloaded, its status is also updated.

Example requests and further information on the GetStatus operation is available at http://pyoseo.readthedocs.org/en/latest/getstatus.html

DescribeResultAccess

When an order has finished processing, a client can issue a DescribeResultAccess request in order to discover the URLs where the ordered items are available. The OSEO server supports both HTTP and FTP pull as delivery methods for an order.

Generally, completed orders will remain available for download for a period of **10 days**.

GetOptions

6.1.2 Server configuration

Being based on django, the server is configured by editing the *settings.py* and *settings_local.py* files. Besides the main django configuration, these files define the following settings, specific to the OSEO server:

- GIOSYSTEM_SETTINGS_URL - The full URL to the API for the giosystem settings web application;
- OSEOSERVER_ORDER_DELETION_THRESHOLD - An integer specifying the number of days that ordered items will remain on the server before being deleted;
- OSEOSERVER_ONLINE_DATA_ACCESS_HTTP_PROTOCOL_ROOT_DIR - The path on the file system to the root directory for placing ordered items which are to be available via HTTP;
- OSEOSERVER_ONLINE_DATA_ACCESS_FTP_PROTOCOL_ROOT_DIR - The path on the file system to the root directory for placing ordered items which are available via FTP;
- OSEOSERVER_AUTHENTICATION_CLASS - The Python import path to the custom authentication class that is used when authenticating incoming requests;

- OSEOSERVER_PROCESSING_CLASS - The Python import path to the custom module that handles the actual processing of orders
- OSEOSERVER_ADMIN_MAILS - a list of e-mail addresses that will be notified whenever an order fails
- OSEOSERVER_MAIL_SERVER - name of the server that is used to send e-mail notifications to the server's administrators whenever an order fails
- OSEOSERVER_MAIL_SERVER_PORT - port of the mail server
- OSEOSERVER_MAIL_DOMAIN - domain of the mail server
- OSEOSERVER_MAIL_ACCOUNT - user account that is to be used when sending e-mails
- OSEOSERVER_MAIL_ACCOUNT_PASSWORD - password of the account that is to be used when sending e-mails
- OSEOSERVER_AUTH_VITO_ATTRIBUTE - Custom setting used in the authentication module;
- OSEOSERVER_AUTH_VITO_TOKEN - Custom setting used in the authentication module;
- OSEOSERVER_AUTH_VITO_PASSWORD - Custom setting used in the authentication module.

The *settings_local.py* file holds site specific settings that override the ones in *settings.py*.

6.2 Order database

The giosystem ordering service is backed by a django application that includes a database. The database stores information on each order's contents, such as:

- status
- requesting user
- ordered items
- delivery options

The stored information is sufiicciently detailed in order to reconstruct an order request and monitor the complete order status life cycle.

The database is dynamically updated as orders are being processed. It is controlled by the OSEO server. Additionally, there is also a web based client used to inspect and manage the database. It can be accessed at

http://geoland2.meteo.pt/giosystem/ordering/admin

6.3 User authentication module

The authentication module runs whenever there is a request for ordering. Currently, the implemented authentication scheme uses the SOAP headers to supply authentication information between the ordering client and the server.

The authentication scheme agreed between IPMA and VITO operates as follows:

- ordering requests must send in a *wsse:Security/wsse:Password* element with the correct value.
- ordering requests must send in a *wsse:Security/wsse:Username* element with the username of the user that made the order on the PDF portal. The user's password is never sent to the ordering server. When the user tries to access the FTP server to download the data, the FTP server communicates with an LDAP server stored at VITO and authenticates the user using its username and the user supplied password.

6.4 Order processing queue

The giosystem ordering service is backed by a celery task queue. This enables the service to manage task execution processes asynchronously. The queue is populated with *specialized order processing tasks*, which are responsible for doing the actual processing of orders

There are two daemon processes responsible for processing orders and maintenance stuff. They are:

6.4.1 pyoseo-worker

The pyoseo-worker daemon is a celery queue that is responsible for processing orders.

6.4.2 pyoseo-beat

The pyoseo-beat daemon is responsible for performing periodic tasks, such as monitoring the number of FTP downloads for each order item, removing old orders, etc.

Both these daemons are configured to start automatically at boot, so if there ever is a need to reboot the machine, they will be back online as soon as it reboots.

6.4.3 Managing the services

The daemons are managed using SystemV-style init scripts. The scripts are located in the */etc/init.d* directory and can be used like this:

```
sudo service pyoseo-worker status # check if the service is running
sudo service pyoseo-worker start # start the service
sudo service pyoseo-worker stop # stop the service
sudo service pyoseo-worker restart # stop the service and immediately start it again
```

The previous commands can be used with both daemons. Both init scripts are exact copies of the files supplied by the celery development team. The original can be found at celery's github repository.

Each service can be customized by editing the files in:

```
/etc/default/pyoseo-worker
/etc/default/pyoseo-beat
```

For more information on the options for each service, consult celery's documentation.

6.4.4 Log files

The log files for these services are located in */var/log/celery*. They can be monitored in real time by issuing the command:

```
tail -f /var/log/celery/worker1.log # for the pyoseo-worker daemon
tail -f /var/log/celery/celeryd.log # for the pyoseo-beat daemon
```

6.5 Order processing module

The order processing module performs the job of actually fetching the ordered items and placing them in the correct directories on the web or the FTP server, making them available for download. This module makes use of the helper functions defined in the `giosystemcore`.

Order processing consists in:

- retrieving the ordered items and options
- fetching the products from the giosystem host network
- apply some post-processing routines
- placing the ordered items in their web accessible destination

Note: Update this documentation to reflect the current status of the ordering server.

In order to handle order requests that specify delivery as HTTP GET requests, the django-giosystem-downloads app includes a *get_ordered_file* view that will fetch the files from disk. In order for it to work correctly, you must install the xsendfile apache module.

```
sudo apt-get install libapache2-mod-xsendfile
```

6.6 Debugging failed orders

Orders can fail if the user placing the order is not actually registered in the PDF portal. In order to verify that a given user is present in VITO's LDAP database, one can use the *ldapsearch* command. For example, to see if the user *demouser* exists in VITO's LDAP server, issue the command:

```
ldapsearch -v -x -D <bind_dn> -w <password> -s sub -b <search_base> -P 3 -H <ldap_uri> uid=demouser
```

Note: The variables *bind_dn*, *password search_base* and *ldap_uri* are sensitive information. They're values are stored at IPMA's internal network. Look for them in the passwords section of the collaborative platform.

CHAPTER 7

giosystem source code repositories

CHAPTER 8

Debugging giosystemcore errors

In order to debug giosystemcore troubles it is usefull to use an interactive ipython shell.

```
cd <giosystem_code_location>
source venv/bin/activate
ipython
```

Whenever there is some error during a package execution, it is usually usefull to instantiate the package and check that both its properties and those of its inputs and outputs are set correctly.

```
import giosystemcore.settings
import giosystemcore.packages.packagefactory as pf
giosystemcore.settings.get_settings('<settings_url>')
p = pf.get_package('<troublesome_package_name>', '<troublesome_timeslot>')
# check that the package and its files have the expected timeslots
p.timeslot
[f.timeslot for f in p.inputs + p.outputs]
# check that the search paths and search patterns are correct
inp = p.inputs[0]
inp.search_path, inp.search_pattern
# check whether the io_buffer and archive are being used as expected
p.use_io_buffer_for_searching
p.use_archive_for_searching
# make sure not to copy any of these debug outputs to the io_buffer or the
# archive
p.copy_outputs_to_io_buffer = False
p.copy_outputs_to_archive = True
# try to find the inputs
found = p.find_inputs()
# check where is the working directory for the package
p.working_dir
# make sure not to remove the working directory
p.remove_working_directory = False
```

CHAPTER 9

giosystem workshop

This is a short workshop that covers the various aspects of the giosystem software package.

9.1 Session #0 - Tools of the trade

In this session we introduce some software tools used when developing giosystem components.

9.1.1 virtualenv

http://virtualenv.readthedocs.org/en/latest/

Virtualenv is a Python package that allows creating isolated Python environments in any directory. These environments can be controlled independently of the system wide Python installation and can be installed anywhere inside the user's home directory. This means that we can:

- install Python packages without requiring root access;
- install different versions of the same package without conflicts;
- install as many packages as a project needs without polluting the system wide Python installation with packages that are only used by a single project;
- create a virtualenv for each project, which keeps each projects libraries stable and isolated

In short, a virtualenv is a custom Python interpreter, located in a directory chosen by the user, where it is possible to install project-specific packages.

Installation

```
sudo apt-get install python-virtualenv
```

Operation

In order to use a virtualenv, we must:

- create it
- activate it

Lets say we are working on project *~/dev/myproject* and wish to use a virtualenv.

Virtual environments can be created wherever the user wishes, but usually we create them as a sub-directory under the project that will be using the virtualenv. They can have any name, but it is common practice to name them *venv*.

```
mkdir -p ~/dev/myproject && cd ~/dev/myproject
virtualenv venv
```

Now that the virtualenv has been created, we need to activate it, so that it can be used. There are two ways to activate and use a virtualenv:

1. When working in a terminal, we can source the *activate* script. The terminal's command prompt then changes, including the name of the virtualenv to provide a visual cue that the virtualenv is active.

```
geo2@geo2:~/dev/myproject$ source venv/bin/activate
(venv)geo2@geo2:~/dev/myproject$ # the virtualenv has been activated
```

   After activating a virtualenv, the *python* command will now refer to a new Python interpreter, located inside the virtualenv's directory. We can verify this by issuing the *which* command:

```
(venv)geo2@geo2:~/dev/myproject$ which python
~/dev/myproject/venv/bin/python
```

   When we no longer wish to use the virtualenv, we can issue the *deactivate* command and it will be deactivated. The terminal's prompt returns to its normal state to indicate that the virtualenv is no longer active.

```
(venv)geo2@geo2:~/dev/myproject$ deactivate
geo2@geo2:~/dev/myproject$ # the virtualenv is no longer active
```

2. When running scripts and applications, we can directly use the Python interpreter that was created by the virtualenv instead of using the default system Python interpreter. For example, if we had a *myscript.py* file to run, we'd execute it like this:

```
# instead of calling python myscript.py
venv/bin/python myscript.py
```

   In this case, the virtualenv is only used for executing the script, as soon as execution is terminated, so is the virtualenv.

Since a virtualenv is just a directory with custom (but isolated) Python interpreter and packages, uninstalling it is just a matter of deleting the virtualenv directory.

```
rm -rf ~/dev/myproject/venv
```

9.1.2 pip

http://www.pip-installer.org/en/latest/

Pip is an installer of Python packages. It provides some very convenient functionality:

- download and install packages from:
  - the Python Package Index (http://pypi.python.org)
  - source code repositories
- install **and** uninstall packages
- install packages from a list of requirements

Installation

Pip can be installed system wide, for using with the default Python interpreter. It can also be used inside a virtualenv. In this case, the virtualenv command takes care of installing pip for us when creating the virtualenv, so there is no need to do anything.

```
sudo apt-get install python-pip
```

Operation

Using pip to install packages is straightforward. Lets try installing the *requests* package, which is available on pypi:

```
# pip install <package_name or url>
(venv) pip install requests
```

Pip will download and install the packages. There are many other options available, such as specifying the version number for a package, separating the downloading and installing steps, etc.

Uninstalling a package is just as simple:

```
(venv) pip uninstall requests
```

For inspectings which packages are currently installed we can use the *freeze* command:

```
(venv) pip freeze
```

Pip has the ability to install packages from a requirements file. A requirements file is just a normal text file where each package is defined in a new line. This means we can store the output of *pip freeze* into a file and then use this file in another location to install the same packages.

```
(venv) pip install requests
(venv) pip freeze > requirements.txt
(venv) pip uninstall requests
(venv) pip install -r requirements.txt
```

This makes requirements files a nice thing to have in a source code repository, as they can be used to install dependencies.

9.1.3 django

https://www.djangoproject.com/

Django is a web framework. It is used to create applications that operate over the Internet, such as websites and the like.

Django documentation is really good, so the best way to learn it is going through the tutorial at:

https://docs.djangoproject.com/en/1.6/intro/tutorial01/

Generally speaking, django is composed of the following components:

- models - Store data and application logic. Models are saved in a database and are manipulated using an object-relational mapper. This means that Django takes care of writing the actual SQL code to fetch data from the database while we can operate on them as regular objects and methods;
- views - Retrieve data from models and process it to produce the requested information. Views are normal Python functions that usually fetch model objects, do something with them and then return a result as a form of a rendered template.

- urlconfs - Do the matching between URLs and the view functions. Urlconfs take care of matching specific URLs with the respective view functions that fetch the actual data from the models.
- templates - Present the resulting information to the user. Templates use the returned data from views and present it to the user, with nicely formatted fonts, colors, etc.

Installation

Operation

9.1.4 Other libraries

- requests
- pyxb

#Session #1 - giosystem_settings_django application #:: # #*to be written* # #Session #2 - giosystemcore library #:::::::::::::::::::::::::::::::::::: # #*to be written* # #Session #3 - giosystem-processing components #:: # #*to be written* # #Session #4 - giosystem-ecflow integration #:: # #*to be written* # #Session #5 - giosystem ordering service #::::::::::::::::::::::::::::::::::::::: # #*to be written*

CHAPTER 10

giosystemcore

10.1 giosystemcore package

10.1.1 Subpackages

giosystemcore.catalogue package

Submodules

giosystemcore.catalogue.cswinterface module

giosystemcore.catalogue.metadata module

Module contents

giosystemcore.hosts package

Subpackages

giosystemcore.hosts.proxies package

Submodules

giosystemcore.hosts.proxies.ftpproxy module

class giosystemcore.hosts.proxies.ftpproxy.**FTPProxy**(*local*, *remote*)

Bases: object

Connect to another server through FTP and perform various actions.

Note: This proxy will try to connect with the remote host using two methods. One uses the host IP address and the other uses the hostname. The second one takes place if the given IP address is '0.0.0.0' or in case of the first method, by IP, fails. The hostname is build by joining the coresetting Host attributes 'Name' and 'Domain' with a dot.

Example: Name: xpto Domain: meteo.pt

hostname: xpto.meteo.pt

create_connection()

delete_file(*path*)
Delete the remote path.

Parameters **path** (*str*) – The remote path to delete

fetch(*path*, *destination*)
Fetch the input path from remoteHost.

The path are copied to localHost's destination.

Inputs:

path - A string with the full path of the file to get. This string is assumed to contain a real path and not a regular expression.

destination - The directory on this instance's localHost attribute where the file is to be copied to.

Returns:

The full path to the newly fetched file.

find(*path*)
Return a list of paths that match the input path's pattern.

Inputs:

path - A string specifying a regular expression to be interpreted as the full directory plus a pattern for the file's name

send(*path*, *destination*, *temporary_extension='.tmp'*)
Put the local path to the remote server.

Inputs:

path - A string with the path in the local file system. It is assumed to contain a full path not search pattern.

destination - The directory on the remote server where the path will be put. It will be created in case it doesn't exist.

temporary_extension - The extension to use while copying the file. After copying the file is renamed back to its original name.

Returns:

A boolean indicating if the transfer was successful.

giosystemcore.hosts.proxies.sftpproxy module A module providing a class for executing file transfer operations through SFTP.

class giosystemcore.hosts.proxies.sftpproxy.**SFTPClient_RETRY**(*host*, *uname*, *pw*, *max_trys=10*, *sleep=1*)
Bases: object

get_connection()

class giosystemcore.hosts.proxies.sftpproxy.**SFTPProxy**(*local*, *remote*)
Bases: object

Connect to another server through SFTP and perform various actions.

This class stores the SFTP connection in a local cache.

close_connection()

delete_file(*path*)
Delete the remote path.

Parameters **path** (*str*) – The remote path to delete

fetch(*path*, *destination*)
Fetch the input paths from remoteHost.

The paths are copied to localHost's destination.

Inputs:

paths - A string with the full path of the file to get.

destination - The directory on this instance's localHost attribute where the file is to be copied to.

Returns:

The full path to the newly fetched file.

find(*path*)
Return a list of paths that match the 'path' argument.

Inputs:

path - A string specifying a regular expression to be interpreted as the full directory plus a pattern for the file's name

run_command(*command*, *local_bin*, *working_dir=None*)

send(*path*, *destination*)
Put the local path to the remote server.

Inputs:

path - A string with the path in the local file system. It is assumed to contain the full path to the file and not a search pattern.

destination - The directory on the remote server where the path will be put. It will be created in case it doesn't exist.

Returns:

A boolean specifying the overall result of the operation.

class giosystemcore.hosts.proxies.sftpproxy.**SftpProxyStateless**(*local*, *remote*)
Bases: object

Use SFTP to find and fetch files on a remote location

This class does not cache the SFTP connection.

close_connection()
This method is not used by this class. It is kept only for maintaining API compatibility.

delete_file(*path*)
Delete the remote path.

Parameters **path** (*str*) – The remote path to delete

fetch(*path*, *destination*)

find(*path*)

run_command(*command*, *local_bin*, *working_dir=None*)

send(*path*, *destination*)

giosystemcore.hosts.proxies.sshproxy module Script's doctring goes here.

class giosystemcore.hosts.proxies.sshproxy.**SSHProxy**(*user=None*, *ip=None*, *other_options=None*)

Bases: object

Connect to another server through SSH and perform various actions.

CSI_Pattern = '\\x1b\\[\\d+.?\\d*\\w?'

SSH_PROMPT = '.+@.+:.+\\$'

list(*re_pattern=''*)

Perform an 'ls' command using the re_pattern.

this method needs some more work in order to be consistent.

run_command(*command*)

Run a command and parse the result into a clean list.

Inputs:

command - A string with the full command to run.

Returns:

A list of strings with the output of the command.

setup(*user*, *host*, *other_options=None*)

Inputs:

user - A string with the name of the user.

host - A string with the ip or the name of the host.

other_options - A string with extra commands for the ssh login.

Module contents

Submodules

giosystemcore.hosts.hostfactory module

giosystemcore.hosts.hostroles module

giosystemcore.hosts.hosts module

Module contents

giosystemcore.orders package

Submodules

giosystemcore.orders.authentication module

giosystemcore.orders.orderpreparator module

giosystemcore.orders.singledownloads module

Module contents

giosystemcore.packages package

Subpackages

giosystemcore.packages.externalcode package

Submodules

giosystemcore.packages.externalcode.cma module

giosystemcore.packages.externalcode.lst module

giosystemcore.packages.externalcode.lst10 module

giosystemcore.packages.externalcode.ngp2grid module

giosystemcore.packages.externalcode.preprocessdata module

giosystemcore.packages.externalcode.preprocessgribdata module

giosystemcore.packages.externalcode.preprocessingalgorithms module

giosystemcore.packages.externalcode.processingalgorithms module

giosystemcore.packages.externalcode.ref module

giosystemcore.packages.externalcode.sa module

giosystemcore.packages.externalcode.satdatastat module

Module contents

Submodules

giosystemcore.packages.basepackage module

giosystemcore.packages.fetchdata module

giosystemcore.packages.metadata module

giosystemcore.packages.packagefactory module

giosystemcore.packages.quicklooks module

giosystemcore.packages.selectionrules module This module has special classes that take care of altering a GIOFile's timeslot and search strings as well as some functions for creating multiple rules.

class giosystemcore.packages.selectionrules.**AgeRule**(*base_timeslot*, *offset_years=0*, *offset_months=0*, *offset_days=0*, *offset_hours=0*, *offset_minutes=0*, *offset_decades=0*, *age='youngest'*, *relative_to_reference='before'*, *fix_year=False*, *fix_month=False*, *fix_day=False*, *fix_hour=False*)

Bases: *giosystemcore.packages.selectionrules.SelectionRule*

AFTER = 'after'

BEFORE = 'before'

IGNORE = 'ignore'

OLDEST = 'oldest'

YOUNGEST = 'youngest'

day

filter_paths(*paths*)

hour

minute

month

timeslot

year

class giosystemcore.packages.selectionrules.**OffsetRule**(*base_timeslot*, *offset_years=0*, *offset_months=0*, *offset_days=0*, *offset_hours=0*, *offset_minutes=0*, *offset_decades=0*, *specific_offset_years=0*, *specific_offset_months=0*, *specific_offset_days=0*, *specific_offset_hours=0*, *specific_offset_minutes=0*, *specific_offset_decades=0*)

Bases: *giosystemcore.packages.selectionrules.SelectionRule*

day

filter_paths(*paths*)
Return the first path on the list of input paths, after sorting.

The list of input paths is first sorted alphabetically and in increasing order and then the first path on the list is chosen.

hour

julian_day

minute

month

timeslot

year

class giosystemcore.packages.selectionrules.**SelectionRule**(*base_timeslot=None*, *offset_years=0*, *offset_months=0*, *offset_days=0*, *offset_hours=0*, *offset_minutes=0*, *offset_decades=0*)

Bases: object

Base class for all selection rules. It is not to be instantiated directly.

base_timeslot

day

hour

minute

month

reference_timeslot

timeslot

year

giosystemcore.packages.selectionrules.**create_arbitrary_interval_input_rules**(*base_timeslot*, *package_offsets*, *frequency_offsets*, *start_offsets*, *end_offsets*)

Create the SelectionRule objects necessary for inputs of type 'multiple'.

This function only creates objects of type OffsetRule.

Inputs:

base_timeslot - the timeslot to be used as a base for the OffsetRule objects that will be created

specifications - a dictionary with the interval specifications. It must have the following key/value pairs:

- **frequency_offsets: a dictionary with the offsets to apply** when iterating through each step of the interval

- **start_offsets: a dictionary with the offsets to apply** to the base_timeslot when determining the start of the interval
- **end_offsets: a dictionary with the offsets to apply** to the base_timeslot when determining the end of the interval

Each of the previous dictionaries must have the following key/value pairs:

- years
- months
- days
- hours
- minutes

giosystemcore.packages.selectionrules.**create_decadal_interval_input_rules**(*base_timeslot*, *package_offsets*, *fix_hour=False*)

Module contents

giosystemcore.scripts package

Submodules

giosystemcore.scripts.installalgorithms module

giosystemcore.scripts.installhdf5 module

Module contents

giosystemcore.tools package

Submodules

giosystemcore.tools.customization module

giosystemcore.tools.georeferencer module

giosystemcore.tools.graphs module

giosystemcore.tools.metadataanalyzer module

giosystemcore.tools.ows module

Module contents

10.1.2 Submodules

giosystemcore.errors module

This module holds all the custom exceptions for the giosystem package.

exception giosystemcore.errors.**CswLoginFailedError**
Bases: *giosystemcore.errors.GiosystemCoreError*

exception giosystemcore.errors.**ExternalProgramError**
Bases: *giosystemcore.errors.GiosystemCoreError*

exception giosystemcore.errors.**FileNotFoundError**
Bases: *giosystemcore.errors.GiosystemCoreError*

exception giosystemcore.errors.**FileNotSentError**
Bases: *giosystemcore.errors.GiosystemCoreError*

exception giosystemcore.errors.**GiosystemCoreError**
Bases: exceptions.Exception

exception giosystemcore.errors.**InvalidClassPathError**
Bases: *giosystemcore.errors.GiosystemCoreError*

exception giosystemcore.errors.**InvalidExecutionTimeslot**
Bases: *giosystemcore.errors.GiosystemCoreError*

exception giosystemcore.errors.**InvalidFileType**
Bases: *giosystemcore.errors.GiosystemCoreError*

exception giosystemcore.errors.**InvalidSettingsError**
Bases: *giosystemcore.errors.GiosystemCoreError*

exception giosystemcore.errors.**OutputNotFoundError**
Bases: *giosystemcore.errors.GiosystemCoreError*

exception giosystemcore.errors.**ProcessingError**
Bases: *giosystemcore.errors.GiosystemCoreError*

exception giosystemcore.errors.**ProductNotDefinedError**
Bases: *giosystemcore.errors.GiosystemCoreError*

exception giosystemcore.errors.**RemoteHostNotFoundError**
Bases: *giosystemcore.errors.GiosystemCoreError*

exception giosystemcore.errors.**SettingsNotFoundError**
Bases: *giosystemcore.errors.GiosystemCoreError*

exception giosystemcore.errors.**UndefinedExternalAlgorithmError**
Bases: *giosystemcore.errors.GiosystemCoreError*

giosystemcore.files module

giosystemcore.products module

This module holds the GIOProduct base class and subclasses for the giosystem package.

class giosystemcore.products.**GioProduct**(*name*)
Bases: object

classmethod from_settings(*short_name*)
Create a product instance.

Inputs:

short_name - A string with the short name of the product

giosystemcore.settings module

A module for reading settings stored in an online REST API.

This module provides a global interface for accessing giosystem settings stored in the giosystem-settings-django app, which is another component of giosystem.

Usually all that is necessary is to instantiate a *SettingsManager* object, providing it with the correct URL for the giosystem-settings-django app. This is achieved by calling the module's *get_settings()* function.

```
import giosystemcore.settings

settings_url = 'http://geo2.meteo.pt/giosystem/settings/api/v1/'
giosystemcore.settings.get_settings(settings_url)
```

class giosystemcore.settings.**SettingsManager**(*url*, *initialize_logging=True*)
Bases: object

Read the settings stored online in the django app.

Settings are fetched when needed instead of all in one go.

get_all_areas_settings()
Return the settings for all of the defined areas

This method will only query the online REST API if its cache is empty.

get_all_ecflow_servers_settings()
Return the settings for all of the defined ecflow servers

This method will only query the online REST API if its cache is empty.

get_all_files_settings()
Return the settings for all of the defined GioFiles

This method will only query the online REST API if its cache is empty.

get_all_hosts_settings()
Return the settings for all of the defined hosts

This method will only query the online REST API if its cache is empty.

get_all_organizations_settings()
Return the settings for all of the defined organizations

This method will only query the online REST API if its cache is empty.

get_all_packages_settings()
Return the settings for all of the defined GioPackages

This method will only query the online REST API if its cache is empty.

get_all_palettes_settings()
Return the settings for all of the defined palettes

This method will only query the online REST API if its cache is empty.

get_all_products_settings()
Return the settings for all of the defined products

This method will only query the online REST API if its cache is empty.

get_all_sources_settings()
Return the settings for all of the defined sources

This method will only query the online REST API if its cache is empty.

get_archive_settings()
Get the specific settings for the archive host.

get_data_receiver_settings()
Get the specific settings for the data receiver host.

get_file_settings(*name*)
Return the settings for the file.

get_ftp_server_settings()
Get the specific settings for the ftp server host.

get_host_settings(*name*)
Return the settings for the host.

get_io_buffer_settings()
Get the specific settings for the io buffer host.

get_multiple_host_settings(*host_uris*)

get_multiple_package_settings(*package_uris*)

get_order_server_settings()
Get the specific settings for the ordering server host.

get_organization_settings(*uri*)

get_package_settings(*name*)
Return the settings for the package.

get_palette_settings(*uri*)

get_product_files_settings(*product_short_name*, *file_type*)

get_product_settings(*uri*)

get_product_settings_by_name(*short_name*)
Get product settings given its short name.

get_source_settings(*uri*)

get_source_settings_by_name(*name*)
Get source settings given its name.

get_suite_settings(*uri*)

get_suite_settings_by_name(*name*)
Get a suite's settings given its name.

get_web_server_settings()
Get the specific settings for the web server host.

giosystemcore.settings.**get_settings**(*base_url=None*, *initialize_logging=True*)
Return the settings manager.

If the settings manager has already been created by a previous call, it will be returned. If not, the base_url argument must be provided so that a new settings manager can be created from the settings present at the URL. This ensures that multiple calls to this function will not trigger unneeded network traffic.

Parameters **base_url** (*str*) – URL to query the REST API and get the defined settings. A value of None will assume that the previously cached settings should be returned instead.

giosystemcore.sources module

This module holds the GioSource class, which is used to represent the remote sources of information for the giosystem.

class giosystemcore.sources.**GioSource**(*name*)
Bases: object

classmethod **from_settings**(*name*)
Create a new instance from the input name.

Inputs:

name - The name of the source to create

giosystemcore.utilities module

This module holds some utility functions for the giosystem package.

In order to use logging inside any function defined here, remember to call the _get_logger function() first.

giosystemcore.utilities.**apply_timeslot_offset**(*original*, *operation*, *unit*, *value*)
Offset a timeslot according to input parameters

Inputs:

original - a datetime.datetime object to offset

operation - a string with the arithmetic operation to perform. Can be either 'ADD' or 'SUBTRACT'

unit - a string with the temporal unit to use when offsetting the timeslot. Can be either MINUTE, HOUR or DAY

value - an int with the value to use for offsetting.

giosystemcore.utilities.**create_directory_tree**(*target_directory*)
Create a directory tree

This function has been moved out of the hosts module in order to start a needed refactoring of the hosts.

giosystemcore.utilities.**displace_timeslot**(*timeslot*, *unit*, *operation*, *value*)
Inputs:

timeslot - A datetime.datetime object with the timeslot to displace

unit - A string with the unit of time to use. Accepted values are:

- minute
- hour
- day

operation - A string with the time displacement operation that is to be used. Accepted values are: - add - subtract

value - An integer specifying how many units of time is the timeslot to be displaced

giosystemcore.utilities.**extract_product_name_from_file_name**(*file_name*)

giosystemcore.utilities.**extract_timeslot_from_directory_structure**(*directory*)

giosystemcore.utilities.**extract_timeslot_from_file_name**(*file_name*)

giosystemcore.utilities.**extract_timeslot_from_path**(*path*)
Extract a datetime object from a path

The timeslot is first searched in the basename of the path and if nothing is found it searches in the dirname.

giosystemcore.utilities.**find_area**(*search_string*, *settings*)
Find the area by inspecting the input search string.

Available area names are retrieved from the settings

Inputs:

search_string - the string to process

settings - a giosystemcore.settings.SequentialSettings object

Returns:

A string with the name of the area or None.

giosystemcore.utilities.**get_directory_contents**(*directory*)
Yield the files that are contained in a directory.

This is a generator function. It yields results one by one. It was designed to cope with directories that store a large number of files. It does not keep the list of files that are inside the input directory in memory.

directory: str The path to the directory to scan

str The full path to each of the directory's contents.

giosystemcore.utilities.**get_file_originator_package**(*file_name*, *settings*)

giosystemcore.utilities.**get_last_modified_time**(*file_path*)
Get the datetime of the last modification of the file.

giosystemcore.utilities.**get_mimetype**(*path*)

giosystemcore.utilities.**get_packages_that_use_file**(*file_name*, *settings*)

giosystemcore.utilities.**get_parent_output_dirs**(*file_name*, *settings*)
Get the output dirs for a GIOFile based on its parent package.

Inputs:

file_name - A string with the name of the GIOFile

settings - A dictionary holding the global settings

timeslot - A datetime.datetime object with the timeslot of the GIOFile.

giosystemcore.utilities.**get_unofficial_creation_date**(*record_identifier*, *catalogue_hacks_api_url*)
Retrieve a record's creation date.

This function uses an unofficial API to reach the catalogue's database because the CSW protocol defines a record's creation date as only of type Date, while we need it to be a DateTime.

Returns

giosystemcore.utilities.**offset_timeslot**(*timeslot*, *offset_years=0*, *offset_months=0*, *offset_days=0*, *offset_hours=0*, *offset_minutes=0*, *offset_decades=0*)

Offset a timeslot.

The 'offset_decades' argument is applied first, before the other offsets.

Inputs:

timeslot - the timeslot to offset

offset_years - an integer specifying how many years to offset the timeslot with

offset_months - an integer specifying how many months to offset the timeslot with

offset_days - an integer specifying how many days to offset the timeslot with

offset_hours - an integer specifying how many hours to offset the timeslot with

offset_minutes - an integer specifying how many minutes to offset the timeslot with

offset_decades - an integer specifying how many decades to offset the timeslot with.

There are three decades in each month, starting at the 1st, 11th and 21st days. The third decade's length depends on the number of days in the month.

The default value is None. If it is set 0, The timeslot is reduced to the 1st day of the current decade.

Returns:

A new timeslot.

giosystemcore.utilities.**parse_marks**(*the_string*, *base_objects=[]*, *clean_special_characters=True*)

Will convert the marks on input string to meaningful values.

Marks are inserted in the input string by wrapping expressions in {}.

Inputs:

the_string - the string that will be converted. This string will

typically be:

1 - the output directory of the package that originated this file;

2 - the search_path of this file, in case it is an input to the system;

3 - the search pattern of this file;

base_objects - a list of objects or strings that specify which objects will be used for attribute susbtitution. Objects will be tried in sequence and the first to match is kept.

Example strings:

1 - OUTPUTS/PROCESSING/{external_code-name}/V{external_code-version}/{year}/{month}/{day} 1 - OUTPUTS/PROCESSING/{name}/V{external_code-version}/{year}/{month}/{day}

In this example provides, {name} refers to the name of the originating package, not to the name of the file. The same is true for the {year}, {month} and {day} marks. When the string to parse refers to a directory, and the GIOFile is an output from some package, the attributes will be refering to the package and not the GIOFile.

2 - raw_inputs/GOESE 3 - HDF5_GEOLAND2_{source-name}_CMa_{source-disk_name}_{timeslot}.h5

giosystemcore.utilities.**process_token**(*token*, *base_objects*)

giosystemcore.utilities.**replace_age_path**(*the_string*, *timeslot*)

giosystemcore.utilities.**revert_timeslot**(*timeslot*, *revert_rules*)

giosystemcore.utilities.**revert_timeslot_temporal_rule**()

giosystemcore.utilities.**unzip_dictionary**(*dictionary*)

giosystemcore.utilities.**write_namelists**(*namelists*, *parameters*)

Generate a Fortran namelist string with the inputs.

namelists: list An iterable with dicts holding the namelists' structure. Each dict should have the following keys: 'name', 'default_values'. The contents of the 'default_values' key must be a dict with parameter names as keys and respective values as values

parameters: dict A mapping with the values to use for the various namelist parameters. The keys that are missing from this dictionary will be filled with the values coming from the 'default_values' section of the input *namelists*

str A string with the namelist contents

giosystemcore.utilities.**write_single_namelist**(*namelist_spec*, *parameters*)

giosystemcore.version module

10.1.3 Module contents

CHAPTER 11

Indices and tables

- genindex
- modindex
- search

g

A

B

C

D

E

F

G

H

I

J

L

M

O

P

R

S

T

U

W

Y