Automatic Interferogram System

System architecture

Configuration and Operation Manual

Project Sponsored by: NOA – IAASARS

Project team

|  |  |  |
| --- | --- | --- |
| **Member** | **Task** | **Contact** |
| Dr Harris Kontoes – NOA Research Director | Sponsor, Management | [kontoes@space.noa.gr](mailto:kontoes@space.noa.gr) |
| Dr Giannis Papoutsis – NOA Reasearcher | Management, Design, Testing | [ipapoutsis@noa.gr](mailto:ipapoutsis@noa.gr) |
| Alex Apostolakis – Electrical and Computer Engineer, MSc SSTA | System Design, Development, Testing | [a.apostolakis@yahoo.gr](mailto:a.apostolakis@yahoo.gr)  [alex.apostolakis@technoesis.gr](mailto:alex.apostolakis@technoesis.gr)  tel: +306974045959 |
| Andreas Al Saer | Development, testing | [adreas.saer@gmail.com](mailto:adreas.saer@gmail.com) |

Document History

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Version | Author | Reviewer |
|  | 1.0 | Alex Apostolakis |  |
|  |  |  |  |

Contents

[1. Introduction 5](#_Toc517276680)

[2. System Architecture 6](#_Toc517276681)

[3. Installation 7](#_Toc517276682)

[3.1. Products collection and Task scheduling Service 7](#_Toc517276683)

[3.2. ENVI IDL 7](#_Toc517276684)

[3.3. PostgreSQL Service database 7](#_Toc517276685)

[4. Service basic flowchart 8](#_Toc517276686)

[5. Service Operation 9](#_Toc517276687)

[5.1. Start, Stop service 9](#_Toc517276688)

[5.2. The folder structure and file naming 9](#_Toc517276689)

[5.2.1. Configuration folder 9](#_Toc517276690)

[5.2.2. Orbits folder 9](#_Toc517276691)

[5.2.3. Logs folder 10](#_Toc517276692)

[5.2.4. Events folder 10](#_Toc517276693)

[5.2.5. Sentinel-1 folder 10](#_Toc517276694)

[5.3. New service request 11](#_Toc517276695)

[5.4. Monitor Processing 11](#_Toc517276696)

[5.4.1. View view\_service\_output 12](#_Toc517276697)

[5.4.2. View steps\_exec 13](#_Toc517276698)

[5.5. Control execution of Service requests 13](#_Toc517276699)

[5.5.1. Reset a service request 13](#_Toc517276700)

[5.5.2. Re-execute a step 14](#_Toc517276701)

[5.5.3. Stop execution and Cancel a step 14](#_Toc517276702)

[5.6. Logging 14](#_Toc517276703)

[5.6.1. Main application Log 14](#_Toc517276704)

[5.6.2. Monitor service status Log 14](#_Toc517276705)

[5.6.3. Service request Log 14](#_Toc517276706)

[5.6.4. The STEPS\_LOG table 14](#_Toc517276707)

[6. Service Configuration 16](#_Toc517276708)

[6.1. Main service configuration file 16](#_Toc517276709)

[6.2. The STEPS table 19](#_Toc517276710)

[6.3. The interferogram “ifg” service steps implementation 20](#_Toc517276711)

[6.4. The USERS table 21](#_Toc517276712)

[6.5. The SARscape, IDL scripts configuration files 22](#_Toc517276713)

[6.5.1. IDL scripts configuration file 22](#_Toc517276714)

[6.5.2. The xml ENVI - SARscript profiles 23](#_Toc517276715)

[A. Database Schema 24](#_Toc517276716)

[A.1. SATELLITE\_INPUT Table 24](#_Toc517276717)

[A.2. SERVICE\_OUTPUT Table 24](#_Toc517276718)

[A.3. SERVICE\_REQUESTS Table 24](#_Toc517276719)

[A.4. STEPS\_EXECUTION Table 24](#_Toc517276720)

[A.5. STEPS Table 25](#_Toc517276721)

[A.6. STEPS\_LOG Table 25](#_Toc517276722)

[A.7. USERS Table 25](#_Toc517276723)

[B. Python Class reference 26](#_Toc517276724)

[B.1. autoifgsrv.py 26](#_Toc517276725)

[B.2. serviceevent.py 28](#_Toc517276726)

[B.3. searchimages.py 31](#_Toc517276727)

[B.4. serviceprocess.py 34](#_Toc517276728)

[B.5. eventdetection.py 36](#_Toc517276729)

[B.6. geomtools.py 37](#_Toc517276730)

[B.7. step\_utils.py 37](#_Toc517276731)

[C. References 39](#_Toc517276732)

# Introduction

The Automatic Interferogram System has as purpose to automatically create all the SAR interferograms around the location of a geological hazard event like earthquake, volcano eruption etc. The interferograms production is triggered by the input of an area of interest or point and a timestamp. The timestamp represents the time of the event being studied and the area of interest a polygon of an area around the event or the point the point of the event.

After the system is triggered by an event input it automatically scans the Copernicus hubs to find the appropriate Sentinel-1 satellite data, downloads the data and executes the tasks needed to produce the interferograms. For the interferogram creation the system uses ENVI - SARscape commands.

The service users are notified about the progress and the status of processing steps automatically by email or consulting information tables of the service metadata database.

# System Architecture

The main components are:

* The “Products collection and Task scheduling service” written in python. This program initiates the request processing, finds and downloads the necessary satellite inputs and controls the output production tasks.
* The above system is assisted by a service metadata database (postgresql) that contains processing information, input and output metadata
* The ENVI SARscape module is the interferogram production engine. The “Products collection and Task scheduling service” execute commands from ENVI SARscape module using IDL scripts in order to create the interferograms
* The IDL scripts that execute the main interferogram tasks calling the SARscape commands.

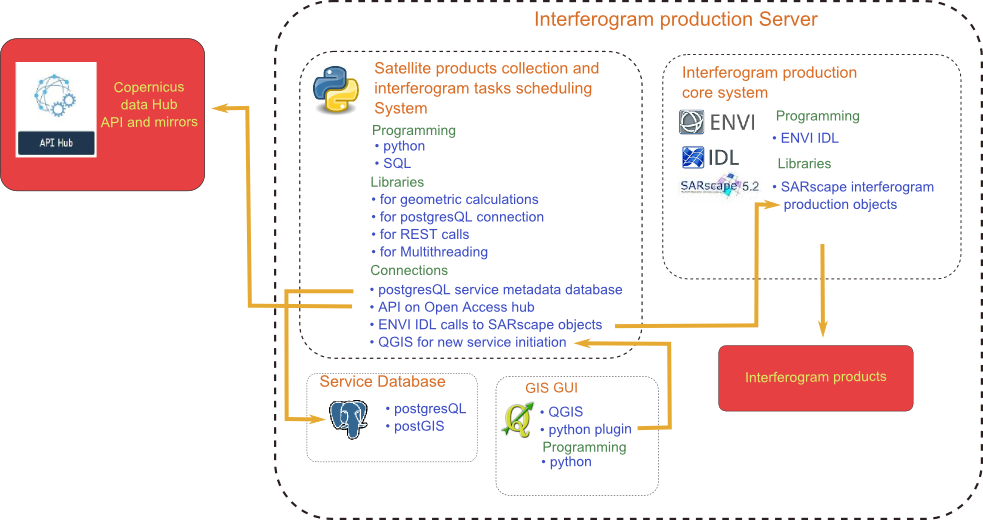


Figure : Technologies and Architecture

# Installation

## Products collection and Task scheduling Service

The “Products collection and Task scheduling Service” is a python program that consists of several python modules. The main .py module that starts the service is the autoifgsrv.py. The service root folder and database are defined by the GEOHUBROOT and GEOHUBDB environment variables that have to be set before running the autoifgsrv.py.

The modules are written for python 2.7.x, thus Python 2.7 has to be installed on the server and additional python 2.7 libraries for Gdal-osgeo, Postgresql, requests

If more than one python installations are present the environment variables for PYTHONHOME and PYTHONPATH have to be set for the right python executables and libraries.

Another point of attention is potential conflict between 32-bit and 64-bit installations of GDAL libraries that may exist simultaneously on the server. In such case the libraries of one installation might have to be completely uninstalled for the other to function properly.

## ENVI IDL

ENVI and SARscape have to be installed on the server of course with their usage license.

In order to run the IDL scripts and the SARscape module routines the environment variables IDL\_PATH, IDL\_DLM\_PATH have to be set like below.

IDL\_DLM\_PATH=<IDL\_DEFAULT>

IDL\_PATH=<IDL\_DEFAULT>;*service source folder*\idl\_code;

## PostgreSQL Service database

The Products collection and Task scheduling Service is assisted by a PostgreSQL database that is used as storage for the service requests data and metadata. Thus on the server a postgreSQl installation has to be present along with the PostGIS add-on.

A copy of the database in SQL format can be found in the source folder of the service program. In order to perform new installation of the service a new PostgreSQL database has to be created from that copy.

# Service basic flowchart

The service is constantly scanning for new trigger input to start processing a new request. If a new event request is detected the processing flow starts. A basic flowchart of the functionality is shown below.

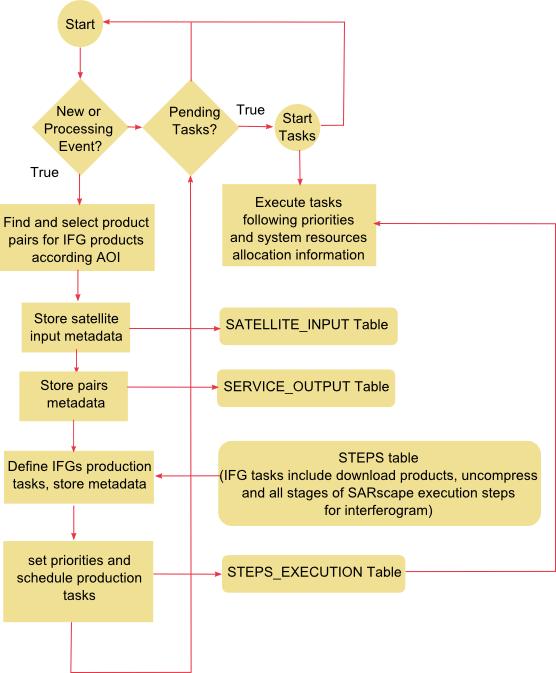


Figure : Basic flowchart

When an event trigger is detected the service starts the search for product pairs to produce the respective interferograms relevant to the event. The pairs searched are two kinds: the pre-seismic and co-seismic pairs.

Pre-seismic pairs consist of two images on the same orbit and direction. The “master image is the most close before the event and the “slave” is the next more recent before the event.

Co-seismic pairs consist again of two images on the same orbit and direction. The “master image is the most close before the event and the “slave” is the more timely close after the event.

First “master” products are searched. The service selects as “master” products all the timely closer to the event products in different orbits that their footprints intersect the AOI.

For each master product the service then run a search to find pairs for co-seismic and pre-seismic products. The condition to accept a pair is configurable and it is defined by a parameter in the main configuration file (see 6.1 paircondition: ).

Of course co-seismic product pairs may not be available the time of the request. The service constantly searches the hubs until those products become available.

Each “approved” product pair, from which an interferogram output will derive, is stored to the SERVICE\_OUTPUT table. In this table the “master” and “slave” pair is recorded in the “inputs” column.

After defining the interferogram outputs the service sets priorities according which co-seismic pair is the most likely to be produced first. Then it stores all the tasks (steps) for each output in the STEPS\_EXECUTION table and starts the execution according priorities, prerequisites and resources limitations. In order to form the records of the STEPS\_EXECUTION table the service looks at the STEPS table (see 6.2, 6.3) where the tasks their sequence, prerequisites and parameters of necessary steps for producing the interferogram (“ifg”) type output are stored.

# Service Operation

## Start, Stop service

The start stop operation is done by windows command (or batch) files.

To start the service run:

**start\_autoifg.bat**

The service starts and then checks the status constantly.

If the start command window is closed, to check the status of the service run:

**check\_autoifg.bat**

To stop the service run:

**stop\_autoifg.bat**

If in a command window (start or check) you get the message "Server has abnormally stopped or hung" check with Task manager for IDL processes or other high CPU consuming processes running (after the server is stopped). Kill those processes and then run:

**clean\_autoifg.bat**

before starting the server again.

## The folder structure and file naming

Under the root folder (see 3.1, 6.1 rootpath:) of the service there are the following folders:

### Configuration folder

The Configuration folder is named by configpath: parameter (see 6.1 configpath:)

Under the configuration folder we find:

* the processed folder (see 6.1 processed:) where the event trigger files are moved
* the “profiles” folder where the XML ENVI - SARscript profiles are located (see 6.5.2)
* the event trigger files are sought in this folder by the service (see 5.3)

### Orbits folder

The Orbits folder is named by orbits: parameter (see 6.1)

Under the orbits folder the orbits files are stored in the following way:

<Orbits\_folder>/<mission (S1A or S1B)>/<Year>/<Month>/<orbit file name>

### Logs folder

The Logs folder is named by logs: parameter (see 6.1)

Under the logs folder the system logs are located (see 5.6)

### Events folder

The events folder is named by datapath: parameter (see 6.1) and it is the main location where output files from processing a service request are created. Under the event folder the output is organized like below:

* <event folder> the event folder name consists of event name and event date time as they were given in the trigger file of the event.
  + A log file specific to the event is located in this folder (see 6.1 processlog:, 5.6).
  + <Output folders> are located under <event folder>. They contain the output of the interferogram processing for the output in the “ifg” folder under this folder. The output folders are named like below:

<master sensing time>\_<slave sensing time>\_<relative orbit>\_<direction>

* + - Interferogram (“ifg”) folder is located under each output folder and contains all the interferogram process output files
      * Under “ifg” except of the output files there are working directories for each process that contain logs of the ENVI – SARscape command execution. The working directory names are defined by the IDL scripts configuration file. (see 6.5.1)

### Sentinel-1 folder

The sentinel1 folder contains all the downloaded products and the output processing of individual products (not pairs) like for example the ENVI import (ingestion) or the DEM extraction. It is named by the sentinel1path: parameter (see 6.1)

Under the sentinel-1 folder the product folders are organized like below:

* <Year>/<Month>/<sensing time>\_<relative orbit>\_<direction>
  + Product file compressed
  + Product quick look image
  + <uncompressed> folder that contains the product data in uncompressed form
  + Ingestion folder contains the output processing data of ENVI SARscape import process (see 6.5.1 import\_dir:)
  + DEM folder contains the output processing data of ENVI SARscape import process (see 6.5.1 DEM\_dir:)
    - Under ingestion and DEM folders except of the output files there are working directories for each process that contain logs of the ENVI – SARscape command execution. The working directory names are defined by the IDL scripts configuration file. (see 6.5.1)

## New service request

### New service initiation based on trigger file from qgis python interface

To initiate and start processing a new service request the system has to detect an input trigger. This trigger is in fact a file that contains the necessary input to form a new service request.

This file can be created automatically by the QGIS add on “Hazard Pro” or it can be created manually by the user. The file name is defined by the parameter “eventfile:” in the main configuration file (see 6.1). If a file with the right specs is present in the configuration folder the service loads it and initiate a new request.

The file format is:

Event Name

YYYY-MM-DD HH:MM:SS

(long,lat)

(long,lat)

Profile XML file full path name

The first line is the event name, the second line is the date, the third and fourth lines are the diagonal coordinates of the vertices of a rectangle that its sides are parallel to meridians and parallels of latitude. The last line is the full path name of an XML file that contains the parameters needed as input to SARscape commands for creating interferogram (see 6.5.2).

### New service initiation based on file from automated event detection system

The service constantly monitors earthquakes web services like USGS and EMSC and collects data for events according the user preferences. The service each time it collects an event creates a file with name :

event\_<id>.json

The file format is:

{

"depth": 20.0,

"magnitude": 5.6,

"name": "NEAR N COAST OF PAPUA, INDONESIA",

"epicenter": "POINT (139.8 -1.84)",

"time": "2018-04-26 16:56:00"

}

All files of the above format are loaded into the service\_request table and have as status ‘ev\_detected’. The user must change the status from ‘ev\_detected’ to ‘ev\_trigger’ to start processing the event (see 6.1). That kind of file can also be created manually by the user in order to initiate a service request.

## Monitor Processing

The tool to monitor the process and steps execution is the pgadmin III tool of the postgreSQL database.

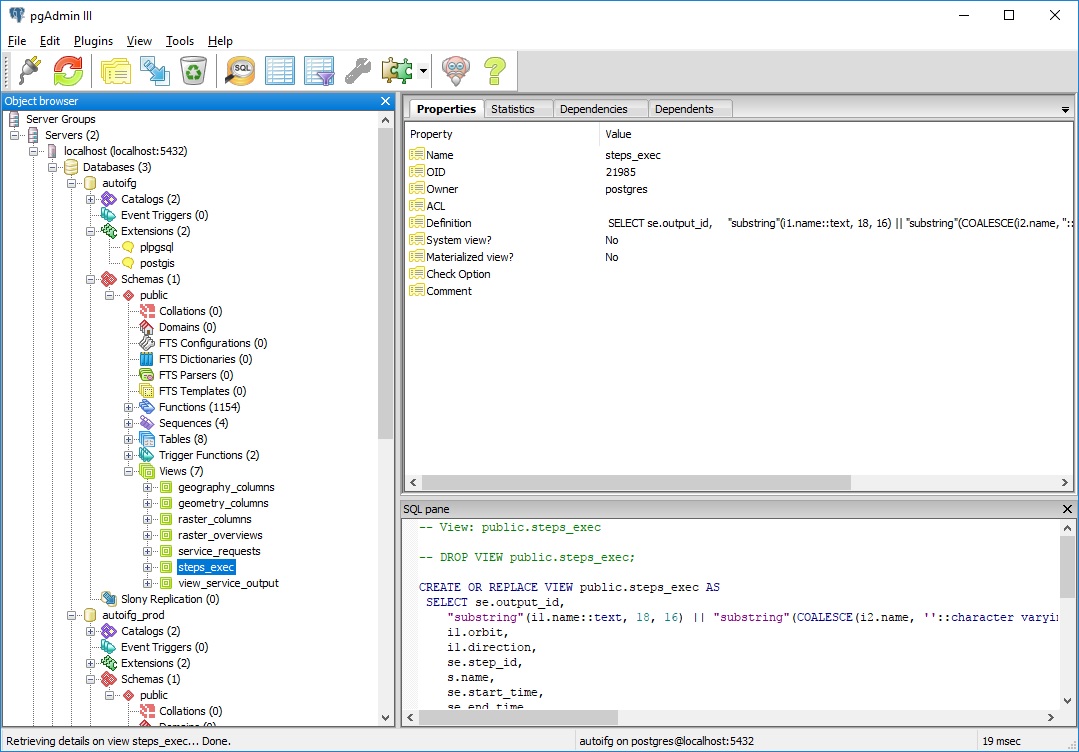


Figure : the PgAdmin III tool

In order to open a database view we start the pgadmin III tool, we select the localhost server and double click to connect. We select from the databases the one that has the name defined by the environment variable GEOHUBDB in the start\_autoifg.bat command window and we open the tree. From the schemas we select public and then the “views” tree (in case we want to open a table we select the “tables” tree). We select the desired view and then we click on the table icon ().

After opening a view we can apply a filter for any of the columns of the view or a combination of them. To apply a filter we click on the filter button (). The conditions syntax is like the “WHERE” part of an SQL sentence. For example if we want to see only the rows where the output\_id column has values over 309 we write: output\_id>309.

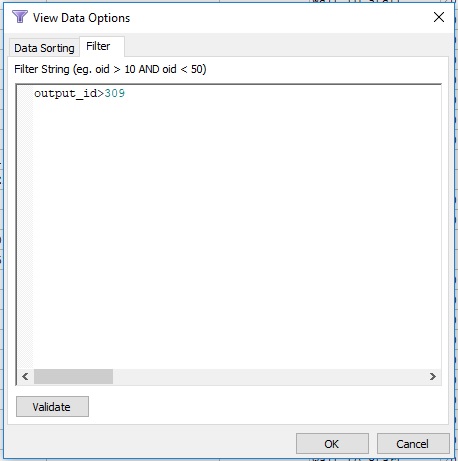


Figure : Filter the View

The database views “view\_service\_output” and “steps\_exec” contain all the necessary information about the processing of the outputs.

To refresh the contents of the view click on or press F5

### View view\_service\_output

The view\_service\_output view contains all the service outputs that are processed, processing or wait to be processed.

Table : The service output view

|  |  |
| --- | --- |
| **Column** | **Description** |
| output\_id | the output id |
| event\_location | The event name (location usually) |
| event\_date | the event date and time |
| Master | the master product name |
| Slave | the slave product name |
| Priority | The processing priorirty of the output |
| output\_status | Status can be requsted, processing or finished |
| first\_step | The first step-task id in the processing seuence |
| out\_start | Start date-time of processing |
| last\_step | The last step-task id in the processing seuence |
| out\_end | End date-time of processing |
| out\_est\_end | Estimated end time of processing |

### View steps\_exec

The steps\_exec view contains in its rows in detail all the service steps of all outputs that are processed, processing or wait to be processed.

Table : The STEPS\_EXEC View

|  |  |
| --- | --- |
| **Column** | **Description** |
| Outname | The output name consist of master product sensing time and slave product sensing time if slave is available |
| Orbit | the relative orbit number |
| Direction | the direction |
| step\_id | the step ID |
| Name | the step name |
| start\_time | The start time of the step |
| end\_time | The end time of the step |
| Duration | the duration |
| Status | The status can be waiting, processing, failed or cancelled. A step is cancelled after it fails to finish successfully a number of times. |
| estimate\_end | the estimation end time for the step |
| Progress | The progress of the step execution. This column has a value only when the step is in processing status. Each step gets input from the process running about its progress.  In the case of external OS processes called (like IDL scripts) the service uses an assigned log parser to parse the appropriate information log of the OS process. |

## Control execution of Service requests

### Reset a service request

You may want to recreate the outputs of a service request in case for example you change the parameters of the interferogram production. To force the recalculation of a service request open the database table “SERVICE\_REQUEST”. To open a database table, follow the procedure described in 5.4 but instead of opening the “views” tree open the “tables” tree.

Locate the row of the service request you want to recalculate, double click on the “status” column of that row and change the status to the value defined by “ev\_reset:” in the main configuration file (see 6.1). Save the table (File->Save or click the “save” icon). In a few seconds all the steps running will stop and the outputs creation steps of this service request will be erased. To start processing the service request again change the status from ‘detected’ to ‘requested’. If you want to erase files created from this service\_request you have to do it manually

### Re-execute a step

You may want to restart a step that was cancelled or because you changed some execution parameters. To restart a step open the database table “STEPS\_EXECUTION” (see 5.4)

Locate the row of the step using columns step\_id and output\_id. You can use the Filter options to make your search easier. Double click on the “status” column of the right step record row and change the status to the value defined by “step\_reset:” parameter in the main configuration file (see 6.1). Save the table (File->Save or click the “save” icon). The step will be first terminated (if running) and will be scheduled to run again from the service according priorities, prerequisites and resource allocation.

### Stop execution and Cancel a step

You may want to avoid the execution of a step and those that are depended on that step. To do that open the database table “STEPS\_EXECUTION” (see 5.4)

Locate the row of the step using columns step\_id and output\_id. You can use the Filter options to make your search easier. Double click on the “status” column of the right step record row and change the status to the value defined by “step\_kill:” parameter in the main configuration file (see 6.1). Save the table (File->Save or click the “save” icon). The step task will be terminated if running and it will turn to cancel status, thus it will be ignored by the step running process.

## Logging

### Main application Log

The main log is located under the “Logs” folder (see 6.1 logs: parameter) and is named system.log. It records the starting and stopping of the service and errors that may happen in the core functionality of the service.

Search for “error”, “warning” or “traceback” keywords in the log to locate problems in execution”.

### Monitor service status Log

This file, under the Logs folder (see 6.1 runfile: parameter) is constantly updated by the server with a timestamp in order to show activity. If that file exists and the service is stopped an abnormal termination has probably happened. To restart the service this file has to be deleted. Additionally the task manager has to be checked for OS processes running that may have been called by the service before abnormal termination.

### Service request Log

This log is located under each service request folder (see 5.2.4). It contains information about the seeking of products in the Copernicus hub for this service request and any warning or error may occur during the initiation and definition of the request processing tasks.

Search for “error”, “warning” or “traceback” keywords in the log to locate problems in execution”.

### The STEPS\_LOG table

For each step-task that is defined uniquely by the step ID and the output ID a history is kept. This table contains the task execution history.

Filter with values of parameters “step\_failed:”, “step\_processing:” or “step\_cancel:” (see 6.1) the status to locate problems in execution of steps.

Table : The STEPS\_LOG Table

|  |  |
| --- | --- |
| **Column** | **Description** |
| output\_id | Output ID |
| step\_id | Step ID |
| Logtime | Log Time |
| Message | Message |
| Status | Status of the step the time of the logging |

# Service Configuration

## Main service configuration file

This file is called ifgconfig.ini and it is the main configuration file that defines the service parameters. It is located in the python source folder. The parameters are described below:

Table : Main configuration table

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| **[Paths]** | **Service Folders** |
| rootpath**:** | The root path of the service. All folders in the configuration are relative to the root path. This parameter is overridden by the environment variable GEOHUBROOT |
| logs**:** | The logs path of the service |
| processlog**:** | The log filename for each event. It is stored under each event folder |
| configpath**:** | The configuration path. In that folder the trigger event file is stored. |
| datapath**:** | In that folder all the interferogram data for each event are stored |
| sentinel1path**:** | Downloaded Sentinel-1 data folder. Additionally ingestion and DEM processing files are stored in that folder, |
| sentinelunzip**:** | Folder to store the uncompressed data of the downloaded sentinel-1 product |
| processed**:** | Folder to store trigger files uncompressed |
| orbits**:** | Folder to store orbits files |
|  |  |
| **[Filenames]** | **Service File names** |
| eventfile**:** | The trigger file name for a new event. This file is moved to "processed folder" after it is processed |
| stopfile**:** | The information file that stops the service |
| runfile**:** | The information file used to check the service is running |
| idl\_pathconfig**:** | The file used to pass parameters to idl scripts (deprecated) |
| idl\_configini**:** | the initial IDL scripts configuration in IDL source code folder |
| idl\_python\_config**:** | The merged configuration file for IDL scripts in the configuration folder. This file is automatically created before running an IDL command (if not present) by merging the initial IDL configuration file with the ifgconfig.ini. In case of change of the initial IDL configuration file or the ifgconfig.ini this file has to be deleted in order to be recreated with the changes applied. |
|  |  |
| **[Copernicushubs]** | **Copernicus hubs** |
| hubs1..n**:** | Each entry (hubsN where N integer from 1 to n) represent a Copernicus hub where the service is searching for products |
| orbitshub**:** | The URL where the orbits data are stored |
|  |  |
| **[dbconnection]** | **Database connection** |
| pg\_dbname**:** autoifg | Database name. The database name is overridden by the environment variable GEOHUBDB |
| pg\_user**:** | Database user |
| pg\_password**:** | Database password |
| pg\_host**:** | Database host |
| pg\_port**:** | Database liscening port |
|  |  |
| **[Status]** | **Status definitions** |
| ev\_trigger**:** | New event, it will automatically start processing |
| ev\_process**:** | Processing event |
| ev\_ready**:** | Finished processing event |
| ev\_reset**:** | Request to reset event and re-process it |
| ev\_detected: | New event that will not automatically start processing. Processing will start after changing the status to ev\_trigger**:** |
| inp\_new: | New satellite input |
| inp\_searching**:** | Searching for Input |
| inp\_downloading**:** | Downloading input |
| inp\_available**:** | Input is localy available (dowloaded) |
| out\_new**:** | out\_new**:** requested |
| out\_searching**:** | searching output |
| out\_downloading**:** | downloading output |
| out\_processing**:** | processing output |
| out\_ready**:** | finished processing output |
| out\_archiving | The output files are currently copied to the archive storage location |
| out\_archived | The output files are moved to the archive storage location |
| step\_wait**:** | Step is waiting for its turn to be started by the service |
| step\_processing**:** | step is running |
| step\_completed**:** | step is completed successfully |
| step\_failed**:** | step has failed |
| step\_cancel**:** | step is cancelled probably after a number of failed process |
| step\_archiving | The output files are currently copied to the archive storage location |
| step\_archived | The output files are moved to the archive storage location |
|  |  |
| **[Types]** | **Output types** |
| out\_sentinel**:** | Sentinel download |
| out\_ifg**:** ifg | interferogram production |
|  |  |
| **[IFG service]** | **[IFG service]** |
| pastperiod**:** | Period to check for first product in the past of the event date. Example: If "repassing" period is 6 days and pastperiod is 1 the service will look first in the range[event date-6 days, event date] to find products. If pastperiod is 2 the service will look first in the range[event date-12 days, event date-6 days] to find products |
| repassing**:** | Repassing satellite period in days |
| searchperiods**:** | How many periods in the past (of the event) to look for products |
| tilesearchrange**:** | Range to search before and after a specific sensing time in minutes when the service is searching based on sensing time. This type of search is happening when the service is searching for the "slave" products. For example if tilesearchrange is 2 the searching range is [sensing time-2 minutes,sensing time+2 minutes] |
| filters**:** | Main filters to apply in search for interferogram product pairs according Full text search Copernicus hub specifications. Separate filters with “,”  Example value:  platformname:Sentinel-1, producttype: SLC |
| inpsplit**:** | Character used to separate output inputs in inputs field of service\_output table |
| paircondition**:** | Condition to check in order to decide if a "slave" product is eligible to form an interferogram pair. For example if paircondition is: pcarea>0.7 or (pcarea>0.3 and not f2roiintersect.IsEmpty()) this means that the service is accepting pairs where master and slave products either have an area overlap greater than 70% or the overlap is greater than 30% and the slave’s area intersects the given area of interest |
| checkslow**:** | Re-check time in minutes for new products when no new product is expected soon |
| checkfast**:** | Re-check time in minutes for new products when a new product is expected soon |
| fastsearchperiod**:** | Time range in minutes to consider a new product is expected soon after the sensing time has past |
| # IFG service parameters | Parameter aliases for calling IFG processing steps |
| pathmaster**:** | Alias for Path to master product |
| pathslave**:** | Alias for Path to slave product |
| pathorbitmaster**:** | Alias for Path to orbit file of master product |
| fileorbitmaster**:** | Alias for Orbit file name of master product |
| pathorbitslave**:** | Alias for Path to orbit file of master product |
| fileorbitslave**:** | Alias for Orbit file name of slave product |
| pathdem**:** | Alias for Path to DEM output |
| pathifg**:** | Alias for Path to IFG output |
| configxml**:** | Alias for XML configuration file and path |
| masterid**:** | Alias for Id of master product |
| slaveid**:** | Alias for Id of slave product |
| outputid**:** | Alias for Id of processing output |
|  |  |
| #request params | Parameters aliases for service request |
| xmlprofile**:** XML profile | Parameter alias for service request XML profile |
|  |  |
| **[Resources]** | **Resources definition** |
| procnum**:** | We assume that each execution step of the service acquire a virtual resource. Thus, limits are defined to the number of parallel steps (or processes-threads) that can use those resources according step priority. For example if procnum is {"server": {"1":1, "2":2}, "internet": {"1":1, "2":2} } that means that the service is using two types of resources ("internet", "server") and processes with priority "1" can run up to one process for both types while processes with lower priority ("2" and above) can run up to two processes for both types again. Of course lower priority steps can not acquire a resource if higher priority steps have consumed the total of available resources for their higher priority. |
| stepretries**:** | Number of retries after failed to characterize a step "cancelled" |
| bestdownloadbytes: | Number of bytes to download for hub speed test |
| slowdownloadbytes: | Number of bytes to download to determine if a download is very slow |
| slowdownloadtime: | Time limit (sec) to download ‘slowdownloadbytes’ |
| mediumdownloadbytes: | Number of bytes to download to determine if a download is slow and search for faster hub |
| Mediumdownloadtime: | Time limit (sec) to download ‘mediumdownloadbytes’ |
|  |  |
| **[Notifications]** | **Notifications mail server** |
| smtphost**:** | smtp host |
| smtpuser**:** | smtp user |
| smtppass**:** | smtp password |
|  |  |
| **[Event]** | **Automated event detection parameters** |
| Minmagnitude: | Minimum magnitude to collect events. |
| minmagnitudegreece: | Minimum magnitude to collect events from Greece area. |
| minmagnitudeworld: | Minimum magnitude to collect events from all over the world. |
| eventpastrange: | Past minutes from now to collect events |
| eventfileprefix: | Event file name prefix |
| rectcornerdist: | Rectangle corner distance in meters from event point to in order to form the search AOI |
| checkinterval: | Minutes interval for querying API for new event |
|  |  |
| **[Archive]** | **Automated archiving parameters** |
| freespacetrigger: | Start archiving when production storage free space is under this number of GBs |
| freespacelimit: | Do not archive if archive location free space is under this number of GBs |
| oldnesstrigger: | Archive outputs that have finished more than this number of days ago. |
| archiveroot: | Path to archive location |
| eventfileprefix: | Event file name prefix |
| rectcornerdist: | Rectangle corner distance in meters from event point to in order to form the search AOI |
| checkinterval: | Minutes interval for querying API for new event |

bestdownloadbytes: 100000

slowdownloadbytes: 1000000

slowdownloadtime: 50

mediumdownloadbytes: 200000000

mediumdownloadtime: 50

## The STEPS table

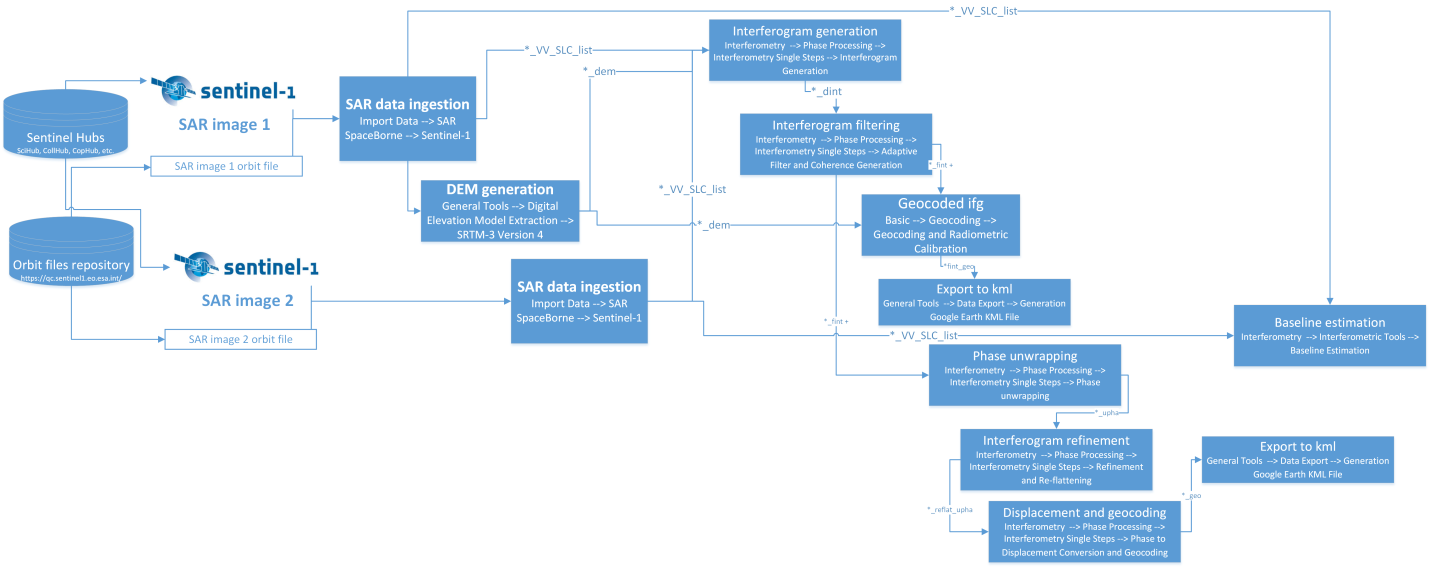
The steps table of the service database contains the necessary tasks information and the flow to execute in order to produce a specific type of a service output. The tasks schedule in the steps\_execution table is created based on the information in that configuration table. Except for the meantime which is automatically calculated from the past execution times all the other fields have to manually be filled by a system’s advanced user.

Table : The STEPS Table

|  |  |
| --- | --- |
| **Column** | **Description** |
| id | The step ID |
| name | The step name |
| command | The command that execute the step. It can be a python function or an OS command |
| params | The parameters and configuration of the command in JSON format. Example 1, step that calls a python function: {"system": "python", "dyn\_params": "#SlaveId#" } The command name is in this case a python function and the parameters to be passed in the actual command is the slave product ID  Example 2, step that calls an external OS function:  { "system": "os", "dyn\_params": "-e sarscape\_script\_EXPORTING\_TO\_KML -args #Path IFG#", "parser": "parseIDLscriptout"}  The command in this case is an external OS process (the IDL CLI) and additionally the parameters to be passed we add a “parser” name which is a python function that monitors the progress and successful or failed completion of the external OS process by parsing its logs. |
| type | The type of the service output ("ifg" for interferogram) |
| prereq\_steps | Prerequisite steps that have to be successfully completed before running this step |
| resource | Virtual resource acquired by this task |
| activated | If false the step is ignored when creating the tasks of a new output |
| meantime | Execution mean time of the step |
|  |  |

## The interferogram “ifg” service steps implementation

The steps for the Interferogram (“ifg”) output type are already filled based on the following flow:



In the steps table the following records are filled that execute the flow in the right order. The commands in the steps correspond to the python functions and IDL scripts that execute the respective tasks.

Table : STEPS table contents for Interferogram output type

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ID** | **Name** | **Command** | **params** | **type** | **Prereq steps** | **resource** |
| 10 | Download master | searchimages.ProductDownloader | { "system": "python", "dyn\_params": "#MasterId#"} | ifg |  | internet |
| 15 | Uncompress master | searchimages.ProductUnzipper | { "system": "python", "dyn\_params": "#MasterId#"} | ifg | 10 | server |
| 20 | Download slave | searchimages.ProductDownloader | { "system": "python", "dyn\_params": "#SlaveId#"} | ifg |  | internet |
| 25 | Uncompress slave | searchimages.ProductUnzipper | { "system": "python", "dyn\_params": "#SlaveId#"} | ifg | 20 | server |
| 30 | Ingestion master | C:\\Program Files\\Exelis\\IDL85\\bin\\bin.x86\_64\\idl.exe | { "system": "os", "dyn\_params": "-e SARscape\_script\_import\_Sentinel\_1 -args #Path Master# #Path Orbit Master# #File Orbit Master#", "parser": "parseIDLscriptout"} | ifg | 15 | server |
| 35 | DEM creation | C:\\Program Files\\Exelis\\IDL85\\bin\\bin.x86\_64\\idl.exe | { "system": "os", "dyn\_params": "-e SARscape\_script\_dem\_extraction -args #Path Master#", "parser": "parseIDLscriptout"} | ifg | 30 | server |
| 40 | Ingestion slave | C:\\Program Files\\Exelis\\IDL85\\bin\\bin.x86\_64\\idl.exe | { "system": "os", "dyn\_params": "-e SARscape\_script\_import\_Sentinel\_1 -args #Path Slave# #Path Orbit Slave# #File Orbit Slave#", "parser": "parseIDLscriptout"} | ifg | 25 | server |
| 50 | Interferogram creation | C:\\Program Files\\Exelis\\IDL85\\bin\\bin.x86\_64\\idl.exe | { "system": "os", "dyn\_params": "-e sarscape\_script\_interferogram -args #Path Master# #Path Slave# #Path IFG# #XML Configuration#", "parser": "parseIDLscriptout"} | ifg | 35,40 | server |
| 60 | Baseline estimation | C:\\Program Files\\Exelis\\IDL85\\bin\\bin.x86\_64\\idl.exe | { "system": "os", "dyn\_params": "-e sarscape\_script\_baseline\_estimation -args #Path Master# #Path Slave# #Path IFG# #XML Configuration#", "parser": "parseIDLscriptout"} | ifg | 30,40 | server |
| 70 | Interferogram filtering | C:\\Program Files\\Exelis\\IDL85\\bin\\bin.x86\_64\\idl.exe | { "system": "os", "dyn\_params": "-e SARscape\_script\_adapt\_filt\_coh\_gen -args #Path Master# #Path Slave# #Path IFG# #XML Configuration#", "parser": "parseIDLscriptout"} | ifg | 50 | server |
| 80 | Interferogram geocoding | C:\\Program Files\\Exelis\\IDL85\\bin\\bin.x86\_64\\idl.exe | { "system": "os", "dyn\_params": "-e sarscape\_script\_geocoding\_rad\_cal -args #Path Master# #Path Slave# #Path IFG# #XML Configuration#", "parser": "parseIDLscriptout"} | ifg | 70 | server |
| 90 | Interferogram export to kml | C:\\Program Files\\Exelis\\IDL85\\bin\\bin.x86\_64\\idl.exe | { "system": "os", "dyn\_params": "-e sarscape\_script\_EXPORTING\_TO\_KML -args #Path IFG#", "parser": "parseIDLscriptout"} | ifg | 80 | server |
| 100 | Phase unwrapping | C:\\Program Files\\Exelis\\IDL85\\bin\\bin.x86\_64\\idl.exe | {"system": "os", "dyn\_params": "-e sarscape\_script\_phase\_unwrapping -args #Path IFG#", "parser": "parseIDLscriptout"} | ifg | 70 | server |
| 105 | Automatic GCP computation | C:\\Program Files\\Exelis\\IDL85\\bin\\bin.x86\_64\\idl.exe | { "system": "os", "dyn\_params": "-e sarscape\_script\_automatic\_gcp\_computation -args #Path Master# #Path IFG# #XML Configuration#", "parser": "parseIDLscriptout"} | ifg | 100 | server |
| 110 | Interferogram refinement | C:\\Program Files\\Exelis\\IDL85\\bin\\bin.x86\_64\\idl.exe | { "system": "os", "dyn\_params": "-e sarscape\_script\_ifg\_refinement -args #Path Master# #Path Slave# #Path IFG# #XML Configuration#", "parser": "parseIDLscriptout"} | ifg | 105 | server |
| 120 | Displacement and geocoding | C:\\Program Files\\Exelis\\IDL85\\bin\\bin.x86\_64\\idl.exe | { "system": "os", "dyn\_params": "-e sarscape\_script\_displacement\_and\_geocoding -args #Path Master# #Path IFG#", "parser": "parseIDLscriptout"} | ifg | 110 | server |
| 130 | Phase unwrapping Export to kml | C:\\Program Files\\Exelis\\IDL85\\bin\\bin.x86\_64\\idl.exe | {"system": "os", "dyn\_params": "-e sarscape\_script\_phase\_export\_to\_kml -args #Path IFG#", "parser": "parseIDLscriptout"} | ifg | 120 | server |

## The USERS table

The users table is the table that contains the users who should be notified for the different phases of the operation of the system and the service requests processing. Each user can be notified for different types of event notifications

Table : the USERS table

|  |  |
| --- | --- |
| **Column** | **Description** |
| name | The user name |
| Email | The user email |
| Notifications | A json format string that contains the types of notifications the user should receive. Example: {"Step started": ["10", "20","40","30","50", "100"], "Step finished": ["10","20","40","30","80","120" ], "output": ["new"],"event":["new"],"error":["hub"]} This user should receive be notified for notifications of type "Step started" for steps with id 10, 20, 40,30,50, 100, for notifications of type "Step finished" for steps with id 10,20,40,30,80,120, for notifications of type "output" for "new" output, for notifications of type "event" in case of "new" event and finally for notifications of type "error" in case of "hub" error |
| Registered | The notification is sent to the user only in case this field is true |

## The SARscape, IDL scripts configuration files

The interferogram production is based on the ENVI – SARscape commands for that purpose. IDL scripts have been developed to execute the necessary tasks that are in their turn executed from the service environment.

### IDL scripts configuration file

Some configuration items like folder and file names have to be common to all IDL scripts and in the same time accessible to the main service program. For that purpose the system uses the “autoifg\_idl.ini” file in JSON format located in the IDL source folder (idl\_code/ folder under python source folder). This file is merged with some parameters of the main configuration file (see 6.1) and is copied to the configuration folder under the service root folder. The configuration parameters are described below:

Table : the IDL scripts configuration file

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| import\_dir: | The ingestion folder of ENVI SARScape import process. It is located under the product download folder |
| import\_work\_dir: | The work folder of the ingestion task. It is located under the import folder |
| dem\_dir: | The DEM folder |
| DEM\_file: | The DEM file |
| dem\_work\_dir: | The work folder of the DEM task |
| ifg\_out: | interferogram output files prefix |
| interferogram\_work\_dir: | The work folder of the interferogram task |
| interferogram\_xml\_params: | Parameter list to import from xml profile for interferogram task |
| baseline\_work\_dir: | The baseline task work folder |
| baseline\_out1: | Baseline task output file name |
| baseline\_out2: | Baseline task output file |
| adapt\_filt\_work\_dir: | Interferogram filtering work folder |
| filtering\_out: | Interferogram filtering output file |
| geocoding\_work\_dir: | Interferogram geocoding work folder |
| geocoding\_out: | Interferogram geocoding output file suffix |
| exporting\_work\_dir: | Interferogram exporting work folder |
| exportifg\_out: | Interferogram exporting output file |
| phase\_work\_dir: | Phase unwrapping work folder |
| phase\_out: | Phase unwrapping output file |
| gcp\_work\_dir: | Automatic GCP computation work folder |
| autogcp\_out: | Automatic GCP computation output file |
| ifg\_refinement\_work\_dir: | Interferogram refinement work folder |
| refinement\_out: | Interferogram refinement output file |
| displacement\_work\_dir: | Displacement and geocoding work folder |
| dispandgeo\_out: | Displacement and geocoding output suffix |
| phase\_export\_work\_dir: | Phase unwrapping export work folder |
| exportphase\_out: | Phase unwrapping export output file |

### The xml ENVI - SARscript profiles

SARScape commands can take as input a significant number of configuration parameters defining the processing methods, inputs and results. All these parameters are included in an xml file located under the “profiles” folder which is under the service’s configuration folder (see 5.1)

Each service request can be bound to a different xml file providing as input to the SARscape commands the calculation parameters of that specific xml.

# Database Schema

## SATELLITE\_INPUT Table

|  |  |
| --- | --- |
| **Column** | **Description** |
| id | The ID given by GEOHUb database |
| product\_id | The ID product ID from Copernicus hubs |
| sensing\_start | Sensing start time (UTC) |
| sensing\_stop | Sensing stop time (UTC) |
| direction | Orbit direction (Ascending- Descending) |
| orbit | The relative orbit number |
| footprint | The footprint polygon of the product in "Well Known Text" format |
| orbit\_file | The file name of that includes orbit information retrieved from: https://qc.sentinel1.eo.esa.int/aux\_resorb/ |
| status | the status of the product. It can be requested, available or downloading |
| name | The name of the product |
| params | The parameters of the product in JSON format retrieved from query to Copernicus hub |

## SERVICE\_OUTPUT Table

|  |  |
| --- | --- |
| **Column** | **Description** |
| service\_id | The ID of the service request that created the output |
| inputs | The product input ids (master, slave) of the output |
| id | The ID of the output |
| status | the status can be requested, processing or finished (see 6.1) |
| type | The output type. "ifg" for interferogram |
| priority | The output Priority. Outputs will be processed in the sequence defined by this number |

## SERVICE\_REQUESTS Table

|  |  |
| --- | --- |
| **Column** | **Description** |
| id | The ID of the service request |
| name | The name of the request, usually the event location |
| date | The event date |
| status | It can be requested, processing or ready (see 6.1) |
| request\_date | The of the request initiation |
| search\_poly | The polygon of the Area of interest in binary format |
| last\_check | The date time of the last search for products relevant to the request |
| request\_params | The parameters the specific to the service request in JSON format. (eg SARscape xml parameters file) |

## STEPS\_EXECUTION Table

|  |  |
| --- | --- |
| **Column** | **Description** |
| output\_id | The output ID for that step |
| step\_id | The step ID |
| start\_time | Date time the task started |
| end\_time | Date time the task ended |
| status | it can be waiting, processing, failed, completed or cancelled (see 6.1) |
| estimate\_end | An estimation of the date time the task processing will finish |
| dyn\_params | The specific parameters for that task that are passed to the task command |
| Progress | A message that is updated every few seconds when the task is in processing status and contains information about the progress. |
| enabled | If false the task can' t start |

## STEPS Table

See 6.2

## STEPS\_LOG Table

See 5.6.4

## USERS Table

See 6.4

# Python Class reference

## autoifgsrv.py

DESCRIPTION

This is the file that contains the class of the main service loop and starts the service and controls the running of the steps.

CLASSES

Looper

class Looper(\_\_builtin\_\_.object)

| Main Service Loop

|

| Methods defined here:

|

| \_\_init\_\_(self)

| Loads Environment

| archive(self)

| Moves outputs to an archive storage when there is limited space on production storage

| or there are "old" outputs present in production storage

|

| check\_already\_running(self)

| Check if service already running

|

| check\_archived\_status(self, archive\_not\_running)

| Checks if the archived outputs are really moved from production storage

|

| check\_archiving\_status(self, archivekeys)

| Checks if the archiving has finished for an output

|

| check\_steps\_status(self)

| Checks steps metadata for abnormal states, restart failed steps, handles step kill or reset commands

|

| check\_stop\_status(self)

| Check if stop command activated (if stop file trigger exists)

|

| cleanthreads(self)

| Cleans stopped threads and checks for reset or cancel commands for a step.

| Updates metadata database with steps logging and status information.

|

| done\_prereq(self, step)

| Check if prerequisite steps of a step are executed

|

| execute\_step(self, step)

| Execute a service step

|

| get\_freeSpace(self, drive)

| Return the free space of a drive (Windows command)

|

| get\_notif\_data(self, step)

| Get data to form Notification

|

| has\_processing\_conflict(self, step\_id, output\_id, steprec)

| Checks a step before starting it for processing conflicts with steps running.

| Avoids to run the same step twice and checks if a step accesses the same input files with running steps

|

| inputinuse(self, input\_id)

| Checks if a running step uses a specific input file

|

| is\_resource\_available(self, resource, step\_priority, terminate=True, allpriorities=True)

| Check if a 'resource' is available in order to run a step

| In case of priority 1 step it terminates all non priority 1 to free resources

|

| log\_except\_hook(self, \*exc\_info)

| Log Unhandled exceptions to system log

|

| loopforever(self)

| Run main Loop

|

| print\_instructions(self)

|

| process\_count(self, resource, priority=None)

| Count all threads. If 'priority' if specified it counts threads of the specific priority

|

| step\_runner(self)

| Collects the steps that should run in the right order and starts a step if conditions are met

|

| stop\_on\_escape(self)

|

| stop\_on\_stopstatus(self)

| Stop server if stop command activated

|

| store\_process\_time(self)

| Store average process time to metadata database

|

| terminate\_all(self, resource)

| Terminates all threads or a specific resource

|

| terminate\_all\_non\_p1(self, resource)

| Terminates all non priority 1 threads

|

| terminate\_service\_threads(self, sid)

| Terminates all threads or a specific service

|

| update\_steps(self)

| Updates metadata database with information collected from threads about running steps.

| Steps progress and logging information are updated

|

| validate\_history(self, step)

| Check if a step with same parameters with the one specified by 'step' is running or is completed

## serviceevent.py

DESCRIPTION

This file contain the class that initiates the service and searches for the Interferogram pairs

The service requests are started based on the event information files that are located in the confiuration folder.

There are of two possible formats: The automatied event detection system or the python-qgis interface.

The user in order to create a custom service request that does not come from either the detection system or the Qgis interface can manually create an event information file of either format.

CLASSES

servicerequest

class servicerequest

| It initiates the service, searches for the Interferogram pairs, creates service metadata

|

| Methods defined here:

|

| \_\_init\_\_(self, env)

| initiates environment and class scope variables

|

| check\_update\_output(self, output)

| Automatic update of output status

|

| check\_update\_service(self)

| Automatically updates service request status

|

| clean\_steps\_output(self, output)

| Clean steps in steps\_execution table that are inactive or deleted in steps table

|

| closelog(self)

| Closes the service request log

|

| create\_output\_notif(self, output, est\_start, estim\_start\_inter, prev\_est\_end, prev\_est\_end\_inter)

| Prepares notification information for new output

|

| estimate\_ingestion\_delay(self, master\_id, method=1)

| Estimates the delay of imagery availability on sentinel hubs after satellite passing using different methods

|

| estimate\_output\_end(self, output, enabled)

| estimates output end time

|

| estimate\_output\_start(self, output, prev\_out\_estim\_end, prev\_out\_estim\_end\_inter)

| estimates output start time

|

| estimate\_step\_end(self, est\_start, prevstep, freshstep, step)

| Estimates step's end time

|

| filter\_masters(self, entries)

| Filter master entries older than existing in same orbit and masters that have small intersection with roi

|

| filter\_slaves(self, entry, candidatepairs, preco)

| Filter slave entries according intersection with master and ROI

| and according sensing time if others exist on same orbit

|

| find\_masters(self, ptracer)

| Find 'master' imagery that intersects the AOI for a number of sentinel re-passing periods in the past

| Product Orbits found in more recent periods are filtered out

|

| find\_outputs(self)

| Searches, finds and stores outputs and steps for service in metadata

|

| find\_probable\_repassing(self, inp1)

| Estimates and returns most probable re-passing time of the Sentinel-1 for specific product

|

| find\_probable\_repassing\_event(self)

| Estimates and returns most probable re-passing time of the Sentinel-1 for all products for the specific service

|

| find\_slaves(self, masterentry, ptracer)

| Find 'slave' imagery for co-seismic and pre-seismic pairs.

| Searches in the estimated repassing time range for a number of sentinel re-passing periods after the master

|

| get\_detected\_AOI(self, wktpoint)

| Creates a rectangle around the event point

|

| get\_eventfolder(self, eventregion, eventdate)

| Returns service request processing folder for IFGs

|

| get\_eventname(self, eventregion, eventdate)

| Returns the service request safe name for folder.

|

| get\_eventparams(self)

| Get parameters from qgis interface file

|

| get\_orbits\_sensing(self, dtype='masters')

| Creates a list with newest or oldest sensing time per relative orbit

| in masters, pre-seismic slaves or co-seismic slaves for the service output

|

| get\_output(self, output\_id)

| Returns output properties of specific output and sets to the class the service properties of this output

|

| get\_output\_config(self, output\_id)

| Returns configuration properties of specific output in a dictionary

|

| get\_output\_inputs(self, ifg\_output)

| Returns input files of specific output

|

| get\_output\_name(self, output\_id)

| Forms and returns output name for notification

|

| get\_service\_ids(self, status)

| Returns a list with the service request ids of a specific status

|

| init\_detected\_event(self, eventfname, systemlog)

| Initiates a service in 'detected' status based on automated event detection files

|

| init\_service(self, systemlog)

| Initiates service requests in case event information files exist in configuration location

|

| init\_trigger(self, systemlog)

| Initiates a service in 'requested' status from trigger file from python-qgis interface

|

| insert\_event(self, eventregion, eventdatest, status, rect\_wkt2D, request\_params, systemlog, magnitude=None, epicenter=None, depth=None)

| inserts event in service\_request table

|

| masters\_seeker(self, ptracer, period, period\_from, period\_to, orbitfilter, roi)

| Seeks 'master' imagery in the AOI for a specific period in all specified sentinel hubs

|

| move\_trigger(self, eventpath, eventfile, eventregion, eventdate, systemlog)

| moves trigger file to processed location

|

| next\_product\_time(self, sid=None)

| Estimate and return availability time of next satellite imagery product of service (not available)

|

| openlog(self, eventfolder)

| Creates or initiates the service request log

|

| reset\_service(self, sid=None)

| Reset service by stopping all running service steps and erasing all outputs and steps metadata

| Files created on file system must be deleted manually

|

| sat\_input\_orbit\_file(self, entry)

| Store satellite imagery files information to satellite\_input table

|

| set\_priorities(self, sid=None)

| Sets the processing priority of the for the outputs of the service request

|

| set\_service(self, status, rs\_id=None, initlog=True)

| Sets the service attributes of the service event class reading them from the service\_request table

|

| slaves\_seeker(self, master\_entry, ptracer, orbitfilter, preco, searchfrom, searchto)

| Seeks 'slave' imagery for a specific time period in all specified sentinel hubs

|

| sql\_get\_output(self)

| Forms an SQL join to retrieve all properties needed for an output

|

| sql\_get\_repassing(self, inp1=None)

| Forms SQL for find most probable re-passing time of the Sentinel-1

|

| step\_params(self, output, step)

| Forms and returns step's dynamic parameters for steps\_execution table

|

| store\_idl\_config(self)

| Updates IDL configuration file

|

| store\_output(self, masterentry, pair=None)

| Store output to service\_output table

|

| store\_output\_config(self, output\_id)

| Store configuration file for IDL steps input (used for testing IDL code)

|

| store\_sat\_input(self, entry)

| Store satellite imagery files information to satellite\_input table

|

| store\_steps(self, sid=None)

| Stores steps in steps\_execution table for all outputs of the event

|

| store\_steps\_output(self, output, est\_start=datetime.datetime(2018, 6, 20, 13, 51, 23, 940000), est\_start\_inter=datetime.datetime(2018, 6, 20, 13, 51, 23, 940000))

| Store output steps in steps\_execution table

|

| update\_service(self, value, field='status')

| Updates service request fields

|

| utc\_to\_local(self, utc\_datetime)

| Converts UTC to local time

## searchimages.py

DESCRIPTION

Library for searching and download sentinel products with the use of metadata database

CLASSES

Downloader

OrbitFileDownloader

ProductDownloader

ProductTracer

ProductUnzipper

dbProduct

class Downloader(\_\_builtin\_\_.object)

| Methods for file download

|

| Methods defined here:

|

| \_\_init\_\_(self, procstatus=None, env=None, log=None)

|

| check\_faster\_hub(self, bytes\_read, product, url)

| Check for faster download or stop download if too slow

|

| download\_speed(self, speeds, response, \*args, \*\*kwargs)

| Measure download speed

|

| downloader(self, dest, product, response, \*args, \*\*kwargs)

| Downloads and stores a file in storage destination 'dest'

| In case of slow download searches for better download sources

| Information on download progress is send to parent processes

|

| find\_best\_host(self, product, exclude\_hosts\_url=[])

| Find best hub according download speed

|

class OrbitFileDownloader(\_\_builtin\_\_.object)

| class to download orbit file

|

| Methods defined here:

|

| \_\_init\_\_(self, orbitpath, orbitfile, procstatus=None, env=None, log=None)

| Initiates class and downloads orbit file

|

| orbit\_download(self)

| Downloads orbit file

|

| validate\_orbitfile(self)

| Validates orbit file

|

class ProductDownloader(\_\_builtin\_\_.object)

| Class contains methods for downloading a sentinel-1 product and update metadata information

|

| Methods defined here:

|

| \_\_init\_\_(self, value, procstatus=None, field='product\_id', env=None, log=None)

| Initiates class, retrieves product metadata information and downloads product

|

| decide\_download(self, uri, filename, status, knownsize=0, knownchecksum='', product=None)

| Decides to download a product from start, not download at all or continue download

| based on existing files and metadata information

|

| file\_as\_blockiter(self, afile, blocksize=65536)

| Iterates through a file with a certain 'blocksize'

|

| find\_host\_session(self)

| Finds a working hub that contains the product

|

| hash\_bytestr\_iter(self, bytesiter, hasher, ashexstr=False)

| calculates MD5 checksum

|

| product\_download(self)

| Downloads Quick look icon and full product

|

| set\_host\_session(self, hub, checksum, httpsize)

| Prepare the class properties to download from a hub host that contains the product

|

| set\_product\_dest(self, product)

| Sets the product destination

|

class ProductTracer(\_\_builtin\_\_.object)

| Methods to search Copernicus hub for sentinel products

|

| Methods defined here:

|

| \_\_init\_\_(self, env, log=None)

|

| checkhubs(self)

|

| format\_query(self, fromdate, todate, filters\_st, roi, datesearch)

| Form hub query

|

| get\_manifest(self, uri, name)

| Retrieves the product manifest from hub

|

| get\_property(self, uri, \_property, sess=None)

| Retrieves a product property from hub

|

| get\_size\_http(self, uri, sess=None)

| Retrieves product size from hub

|

| get\_uri(self, host, product\_id)

| Forms and Returns a sentinel product uri for download

|

| product\_seeker(self, fromdate, todate, filters\_st, roi='', datesearch='beginposition', start=0, rows=100, hubs=None)

| Seek products that meet hub query

|

class ProductUnzipper(\_\_builtin\_\_.object)

| class to unzip downloaded product

|

| Methods defined here:

|

| \_\_init\_\_(self, value, procstatus=None, field='product\_id', env=None, log=None)

| Inits and unzip downloaded product

|

class dbProduct(\_\_builtin\_\_.object)

| This class is used to retrieve and access the metadata of the sentinel product imagery

|

| Methods defined here:

|

| \_\_init\_\_(self, env)

|

| get\_orbit\_dest(self)

| Form and return the orbit file destination on storage

|

| get\_product(self, value, field='product\_id')

| Retrieve the product metadata

|

| get\_product\_dest(self)

| Form and return the product files destination on storage

|

| update\_product(self, value, field=None)

| Update product metadata in satellite\_input table

|

## serviceprocess.py

DESCRIPTION

Classes to handle multithreading

CLASSES

\_\_builtin\_\_.object

osprocess

step\_process

class osprocess(\_\_builtin\_\_.object)

| Handles OS processes run for steps

|

| Methods defined here:

|

| \_\_init\_\_(self)

|

| parseIDLscriptout(self, parsetype, out\_err=None, parsefile=None)

| Parse IDL scripts and ENVI SARscapeo utput files for update status of completion and progress

|

| parseIDLscriptout\_file(self, varargs)

| Find ENVI SARscape output file for parsing progress and define time to consider process frozen

|

| parsping(self, parsetype, out\_err)

|

| parsping\_file(self, varargs)

|

| run\_osprocess(self, procinfo, step, command, varargs=[], environ=None, waitsecs=10)

| Start and monitor OS process

|

| terminate\_osprocess(self, os\_process)

| Terminate OS process

|

class step\_process(\_\_builtin\_\_.object)

| This class handles the execution of a service step either it is a python or OS or IDL process

|

| Methods defined here:

|

| \_\_init\_\_(self, env, output, step)

|

| get\_message(self)

| Retrieve message from process

|

| get\_progress(self)

| Retrieve progress from process

|

| get\_status(self)

| Retrieve process status from process

|

| get\_step(self)

| Retrieve step properties from steps\_execution table

|

| init\_osprocess(self, command, varargs=[], environ=None)

| Initiate class in case of OS process

|

| init\_pyprocess(self, func, varargs)

| Initiate class in case of python process

|

| is\_alive(self)

| Check if process running

|

| terminate(self)

| Terminate process

|

## eventdetection.py

DESCRIPTION

Contains class with methods to retrieve quake events from usgs or emsc APIs

CLASSES

\_\_builtin\_\_.object

EventDetection

class EventDetection(\_\_builtin\_\_.object)

| Methods defined here:

|

| \_\_init\_\_(self, env, log)

| Initiates environment and log

|

| get\_emsc\_uri(self)

| query emsc event API

|

| uri example:

| http://www.seismicportal.eu/fdsnws/event/1/query?starttime=2018-03-18&format=json&minmagnitude=5.5

|

| get\_last\_quakes(self, uri)

| Queries uri and returns last quakes

|

| get\_last\_quakes\_emsc(self)

| Returns last quakes from emsc

|

| get\_last\_quakes\_usgs(self)

| Returns last quakes from usgs

|

| get\_minmag(self)

| Return minimum magnitude

|

| get\_starttime(self)

| Return start time for searching events

|

| get\_usgs\_uri(self)

| query usgs event API

|

| uri example:

| https://earthquake.usgs.gov/fdsnws/event/1/query?format=geojson&starttime=2018-02-21T12:19:58&minmagnitude=5

|

| store\_last\_quakes(self, quakes, source)

| Store found quakes from source according conditions

|

| store\_last\_quakes\_emsc(self, quakes)

| Store quakes from emsc

|

| store\_last\_quakes\_usgs(self, quakes)

| Store quakes from usgs

|

| store\_quake(self, detected\_quake, source)

| Store quake in json file

|

## geomtools.py

DESCRIPTION

Geometry tools based on osgeo ogr,osr libraries

CLASSES

geomtools

class geomtools(\_\_builtin\_\_.object)

| Geometry tools based on osgeo ogr,osr libraries

|

| Static methods defined here:

|

| convert\_geom(WKT, inputEPSG, outputEPSG)

| Convert coordinate system

|

| create\_polygon(coords)

| Create polygon from coordinates list

|

| getGreecePoly()

| Polygon containing Greece Cyprus and part of Asia Minor

|

| pointGeom(point)

| Return point geometry from point coordinates

|

| point\_in\_greecepoly(pt)

| Check if point is within Greece polygon

|

| point\_in\_polygon(pt, poly)

| Check if point is within a polygon

|

| rectangle\_coords(p1, p2)

| Create rectangle coordinates list from two diagonal corners points

|

## step\_utils.py

DESCRIPTION

Utilities to retrieve or update steps\_log, steps\_execution and service\_ouput tables information

CLASSES

steputils

class steputils

| Utilities to retrieve or update steps\_log, steps\_execution and service\_ouput tables information

|

| Methods defined here:

|

| \_\_init\_\_(self, env, systemlog=None)

|

| get\_output(self, output\_id=None, order=None, where\_cl=None)

| Retrieves information from an output based on query criteria

|

| get\_output\_sql(self, output\_id=None, order=None, where\_cl=None)

| Form SQL to retrieve wide range of output information combining steps

|

| get\_step(self, output\_id, step\_id)

| Retrieves a step information dictionary by query in steps, steps\_executionn and service\_output tables

|

| get\_steps(self, query)

| Retrieves a list of steps from steps\_execution table

|

| iter\_output(self, output\_id=None, order=None, where\_cl=None)

| Creates an output iterator based on query criteria

|

| update\_output(self, output, value, field='output\_status', extra='')

| Updates service\_output table

|

| update\_step(self, output, step, value, field='status', extra='')

| Updates step information in steps\_execution table

|

| update\_step\_log(self, output, step, status, message, connection=None)

| Inserts a new entry in steps\_log table

# References

1. ENVI SARscape. [Online] http://www.harrisgeospatial.com/SoftwareTechnology/ENVISARscape.aspx.

2. Python. [Online] https://www.python.org/.

3. Python 2.7 Installation. [Online] https://www.python.org/download/releases/2.7/.

4. GDAL - Geospatial Data Abstraction Library. [Online] http://www.gdal.org/.

5. Command line and environment. [Online] https://docs.python.org/2/using/cmdline.html.

6. Installing SARscape on Windows. [Online] http://www.harrisgeospatial.com/Support/SelfHelpTools/HelpArticles/HelpArticles-Detail/TabId/2718/ArtMID/10220/ArticleID/17296/Installing-SARscape-on-Windows.aspx.

7. ENVI 5.4 / IDL 8.6 Quick Start Install and Licensing Guide. [Online] http://www.harrisgeospatial.com/Support/SelfHelpTools/HelpArticles/HelpArticles-Detail/TabId/2718/ArtMID/10220/ArticleID/15146/14988.aspx.

8. Directory and Search Path Preferences. [Online] http://www.harrisgeospatial.com/docs/prefs\_directory.html.

9. postgresQL. [Online] https://www.postgresql.org/.

10. Postgres backup and restore. [Online] https://www.postgresql.org/docs/8.1/static/backup.html#BACKUP-DUMP-RESTORE.

11. Copernicus Open Access Hub. [Online] https://scihub.copernicus.eu/.

12. Copernicus Collaborative Node (HNSDMS). [Online] https://sentinels.space.noa.gr/.

13. Introducing JSON. [Online] http://www.json.org/.

14. PostGIS. [Online] http://postgis.net/.