ESA Contract No. 4000108867/13/I-BG

FEASIBILITY STUDY FOR GEO-LOCATION ASSESSMENT OF OPTICAL SENSORS GEOACCA

Software User Manual

Deliverable D10

Prepared by

Petra MALCHER, Ursula BLUMTHALER and Thomas NAGLER

ENVEO IT, Innsbruck, AUSTRIA

Issue / Revision: 1 / 1

Date: 21 Dec 2015

Document controlled by: P. Malcher

ESA STUDY CONTRACT REPORT

ESA CONTRACT NO:	SUBJECT:		CONTRACTOR:
4000108867/13/I-BG	Software User Manual		ENVEO
ESA CR ( )NO:	STAR CODE:	NO OF VOLUMES: 1 THIS IS VOLUME NO: 1	CONTRACTOR'S REF: Deliverable D10

ABSTRACT:

This report describes the usage of the GeoAcca software package.

After configuring GeoAcca with the configuration files included in the GeoAcca software package, a command-line tool is used to set up the reference data. This step includes the setup of the database tables, the extraction of image subsets covering the Ground Control Points (GCP), the extraction of Water Body Data shapefiles for each GCP and the storage of the related information to the database. Once the reference dataset has been created, a second command-line tool is used for the calculation of the geolocation errors. This tool includes orthorectification, resampling and extraction of an image subset. After successfully passing multiple quality checks, each subset is matched with the respective reference image subset. The resulting geolocation errors are written to the database together with the image metadata and the quality information.

The full documentation in html format including the software installation and usage is part of the GeoAcca software package.

The work described in this report was done under ESA Contract. Responsibility for the contents resides in the authors or organisation that prepared it.

AUTHORS:
P. MALCHER, U. BLUMTHALER, T. NAGLER

ESA STUDY MANAGER:	ESA BUDGET HEADING
Bojan Bojkov	

This page is intentionally left blank.

DOCUMENT CHANGE LOG

Issue/ Revision	**Date**	**Modification**	**Modified pages**	**Observations**
1.0	27/10/2015	All	All	-
1.1	21/12/2015	All	All	Update to Software version 1.1.0

This page is intentionally left blank.

GeoAcca Documentation

Release 1.1.0

ENVEO IT GmbH

December 21, 2015

CHAPTER

ONE

GETTING STARTED WITH GEOACCA

1.1 Configuration files

For configuration GeoAcca uses the Python ConfigParser module, handling configuration files. The GeoAcca configuration files `*.cfg` can be found in your installation Egg under `config`. The configuration files are organized into sections, where each section contains key-value pairs for your configuration.

1.1.1 GeoAcca

Adapt the file defined by `CONFIG_PKG_FILE` with your GeoAcca settings.

Section **database**:

- `dbuser` your database user (`geoacca`)
- `dbname` your database name (`geoacca`)
- `dbhost` your database host
- `dbport` your database port
- `schema` your database schema name (`myschema`)

Section **beam**:

- `gpt_path` path to the Beam GPT script

Section **thresholds**:

- `sel_threshold` solar elevation on acquisition, processing is stopped for values less than threshold
- `scc_threshold` percentage of snow/cloud coverage of image, processing is stopped for values greater than threshold
- `scc_threshold_feat` percentage of snow/cloud coverage of control object, processing is stopped for values greater than threshold

Section **resampling**:

- `resampling` resampling method to use, available methods are: `Nearest`, `Bilinear` and `Bicubic`

Sections for input sensors **meris**, **aatsr**, **atsr2**, **probav**:

- `sensor_abbr` sensor abbreviation
- `product_id` product identification name
- `in_data_dir_YYYY` product directory of the year `YYYY`; the products are assumed to be organized in yearly periods on disk; for additional years add the respective name-value pairs

- `lines` number of product header lines containing metadata information
- `band_nir` NIR band number
- `out_data_dir` directory where the output should go to
- `out_data_format` format of the output products

Note: All input sensor sections are organized in the same manner. The following keys `sensor_abbr`, `product_id`, `out_data_format`, `lines`, `band_nir` should already be properly assigned.

Sections for reference sensors **landsat**:

- `in_data_dir` product directory

Note: See *sections for input sensors* for residual key explanations.

Sections **[sensor]_gen_aux**, **[sensor]_gcp_aux**:

Note: The sections `[sensor]_gen_aux`, `[sensor]_gcp_aux` hold the sensor specific metadata to be loaded. These settings shoud not be changed. If neccessary the sections can be extended for other metadata info.

1.1.2 Logging

Adapt the file defined by `CONFIG_LOG_FILE` with your settings.

Section **logger_root**:

- `handlers` you may want to change the logger output settings; by default the logger writes both, to file and to console

1.2 EO Data Pool

Ensure the reference scenes listed in `src/auxiliary/landsat.csv` are available in the directory stated as `in_data_dir` in the section `[landsat]` of the GeoAcca configuration file `CONFIG_PKG_FILE`. For each scene the NIR band GeoTiff and the corresponding MTL file is needed.

The EO scenes to be analysed for geolocation errors have to be stored in the folder stated as `in_data_dir_YYYY` (where `YYYY` is the acquisition year) in the respective sensor section of the GeoAcca configuration file `CONFIG_PKG_FILE`.

GeoAcca requires the N1 data format for AATSR and MERIS, E2 for ATSR-2 and hdf5 for PROBA-V. For AATSR and ATSR-2, the Level 1B Top of Atmosphere product (ATS_TOA_1P, AT2_TOA_1P) is required. For MERIS, the Level 1B Full Resolution Full Swath product (MER_FRS_1P) is required. For PROBA-V, the Level 3 Top Of Atmosphere 1-day synthesis product (PROBAV_S1_TOA) is required.

Note: Arrange the EO scenes in monthly and daily sub-directories: `in_data_dir_YYYY/<MONTH>/<DAY>/`

1.3 Command-Line Tools

GeoAcca comes along with some command-line tools which allow the common tasks to be performed without having to open a Python interpreter.

1.3.1 pregeoacca

`pregeoacca` is the GeoAcca command line script for setting up the package basis.

Note: `pregeoacca` has to be run at least once before running `getgeoacca`. Rerun `pregeoacca` if the package basis changes, e.g. you add a new GCP.

The GeoAcca package comes along with multiple CSV files holding information on GCPs, sensors, sensor tracks and matching methods. Basically, `pregeoacca` extracts the relevant information for each GCP listed and processes an appropriate orthorectified reference image subset. The collected information is stored in the database.

Note: At present only LANDSAT is available as reference.

Example uses of pregeoacca:

1. Process the reference subsets for all GCPs listed and add gained information to the database. Neither database update nor reprocessing is forced. The logger output is set to DEBUG level:

```
$ pregeoacca --log=DEBUG
```

2. Process the reference subsets for all GCPs listed and add gained information to the database. Force an update of the reference raster database table:

```
$ pregeoacca --update
```

Attention: The `--update` will delete all or if GCPs are given the respective entries in the database shift table.

3. Process the reference subsets for all GCPs listed and add gained information to the database. Force an update from scratch:

```
$ pregeoacca --update-all
```

Attention: The `--update-all` will delete all entries in the database.

4. Process the reference subsets for the GCPs given in the command line and add gained information to the database. Neither database update nor reprocessing is forced:

```
$ pregeoacca -g N59734_W118694 -g S01862_E137894 -g N35899_E014451
```

5. Give a list of valid GCP IDs:

```
$ pregeoacca --list
```

With GeoAcca installed, please run `pregeoacca --help` to see the full usage documentation.

1.3.2 getgeoacca

`getgeoacca` is the GeoAcca command line script for analyzing the GEOlocation ACCurAcy of level 1B satellite data of the sensors MERIS, AATSR and ATSR2 and level 3 synthesis products of the sensor PROBA-V. The script operates on a given set of orthorectified reference images covering Ground Control Points (GCPs). For each level

1B file given, `getgeoacca` searches matching GCPs, of which orthorectified image subsets are created. After successfully passing multiple quality tests, the image subset is approved to be matched with the reference. The resulting matching parameters are then stored in the database.

Example uses of getgeoacca:

1. Process the GEOlocation ACCurAcy of all MERIS level 1B input files of the year 2004 stored under `in_data_dir_2004` (see *Configuration files*). The processing involves all GCPs stored in DB. The command line parameters `viewmode`, `matchmethod`, and `log` are initialized with its default settings. The output is stored on disk, no update is forced in database:

```
$ getgeoacca -i MERIS -Y 2004
```

2. Process the GEOlocation AccurAcy of all AATSR (default) level 1B input files with acquisition date 28-01-2004 stored under `in_data_dir_2004` (see *Configuration files*) matching the GCP with ID N06244_W005109. The command line parameters `viewmode`, `matchmethod`, and `log` are initialized with its default settings. The output is stored on disk, no update is forced in database:

```
$ getgeoacca -g N06244_W005109 -Y 2004 -M 1 -D 28
```

3. Process the GEOlocation AccurAcy of all MERIS level 1B input files with acquisition date 02-03-2003 stored under `in_data_dir_2003` (see *Configuration files*) matching the GCP with ID S38572_W068754. For matching the TM_CCORR_NORMED method is used. The command line parameters `viewmode` and `log` are initialized with its default settings. The output is stored on disk, no update is forced in database:

```
$ getgeoacca -g S38572_W068754 -Y 2003 -M 3 -D 2 -i MERIS -m TM_CCORR_NORMED
```

4. Process the GEOlocation AccurAcy of all AATSR level 1B input files with acquisition date 22-06-2004 stored under `in_data_dir_2004` (see *Configuration files*) matching the GCP with ID N37614_E024328. Sensor view mode is set to fward. The command line parameters `matchmethod` and `log` are initialized with its default settings. The output is stored on disk, an update is forced in database:

```
$ getgeoacca -g N37614_E024328 -Y 2004 -M 6 -D 22 -v fward --update
```

5. Process the GEOlocation AccurAcy of the AATSR level 1B input files given in the command line. The processing involves all GCPs stored in DB. The command line parameters `viewmode` and `matchmethod` are initialized with its default settings. Logging is set on DEBUG. The output is stored on disk, an update is forced in database:

```
$ getgeoacca -f /home/xyz/2004/06/22/ATS_TOA_1PUUPA20040622_080048_000065272028_00006_12084_5531.N1
```

With GeoAcca installed, please run `getgeoacca --help` to see the full usage documentation.

Note: `getgeoacca` is capable of controlled termination. Therefore create a file named `STOP` in your `out_data_dir` (see *Configuration files*) directory by running the following command:

```
$ touch STOP
```

Do not forget to delete the file before program restart.

1.4 Importing GeoAcca

To import the whole GeoAcca package in python , type:

```
>>> import geoacca
```

CHAPTER

TWO

GEOACCA PACKAGE

2.1 geoacca Package

The geoacca package contains modules and command line scripts for analyzing the GEOlocation ACCurAcy of georeferenced images of optical sensors by comparing EO input data with high-resolution reference data. The comparison is performed at locations of pre-defined Ground Control Points (GCPs) by template matching. The calculated shifts are stored in a database together with metadata information on the input and reference images and geographic information on the GCPs.

2.2 aatsr Module

This module contains a class representing AATSR raster files.

2.2.1 Classes and Inheritance Structure

2.2.2 Module API

class aatsr.**AATSR_RASTERFILE**(*in_data_dir*, *input_filename*, *out_data_dir*, *controlpoint_id*, *PGDB*, *track_id*, *psz*, *band_nir*, *view_mode*, *software_ver*)

Bases: geoacca.raster.RASTERFILE

AATSR raster file class.

It is initiated for the given GCP subset and subclasses the RASTERFILE class, extending it for a method to detect the percentage of snow/cloud coverage.

Parameters

- **in_data_dir** (*string*) – path to the input file directory
- **input_filename** (*string*) – name of the level 1B AATSR input file

- **out_data_dir** (*string*) – path to the output directory
- **controlpoint_id** (*string*) – GCP ID as given in the DB. Processing of the input file will be referred to the given GCP.
- **PGDB** – PostGres DB class
- **track_id** (*integer*) – track ID of the input file as given in the DB
- **psz** (*float*) – pixel size for the output files [m]. It should be the same resolution as the reference files.
- **band_nir** (*integer*) – near infrared band number of the input file
- **view_mode** (*string*) – view mode of the input file acquisition
- **software_ver** (*string*) – GeoAcca software version

Raises

`ValueError`: if the given viewmode is not valid for AATSR

VALID_VIEW_MODES = ('fward', 'nadir')

chkRasterSCC(*scc_threshold*, *scc_threshold_feat*, *mask*, *fmt*)

Check the AATSR raster snow/cloud coverage.

The method applies on the orthorectified and resampled subset, covering the GCP the AATSR raster class is constructed for. If the snow/cloud coverage exceeds either the `scc_threshold` or the `scc_threshold_feat`, which refers to the GCP itself, the `to_be_proc` flag is set to False. The snow/cloud mask is written to a raster file.

Parameters

- **scc_threshold** (*float*) – snow/cloud cover threshold [%]
- **scc_threshold_feat** (*float*) – snow/cloud cover threshold of the GCP feature [%]
- **mask** (*string*) – path to the extended GCP feature mask covering the input control window
- **fmt** (*string*) – format of the snow/cloud mask raster file

Raises

`IOError`: if the AATSR raster file is missing

`ValueError`: if the extended GCP feature mask does not match the snow/cloud mask window size

2.3 atsr2 Module

This module contains a class representing ATSR-2 raster files.

2.3.1 Classes and Inheritance Structure

2.3.2 Module API

class atsr2.**ATSR2_RASTERFILE**(*in_data_dir*, *input_filename*, *out_data_dir*, *controlpoint_id*, *PGDB*, *track_id*, *psz*, *band_nir*, *view_mode*, *software_ver*)

Bases: geoacca.raster.RASTERFILE

ATSR-2 raster file class.

It is initiated for the given GCP subset and subclasses the RASTERFILE class, extending it for a method to detect the percentage of snow/cloud coverage.

Parameters

- **in_data_dir** (*string*) – path to the input file directory
- **input_filename** (*string*) – name of the level 1B ATSR2 input file
- **out_data_dir** (*string*) – path to the output directory
- **controlpoint_id** (*string*) – GCP ID as given in the DB. Processing of the input file will be referred to the given GCP.
- **PGDB** – PostGres DB class
- **track_id** (*integer*) – track ID of the input file as given in the DB
- **psz** (*float*) – pixel size for the output files [m]. It should be the same resolution as the reference files.
- **band_nir** (*integer*) – near infrared band number of the input file
- **view_mode** (*string*) – view mode of the input file acquisition
- **software_ver** (*string*) – GeoAcca software version

Raises

ValueError: if the given viewmode is not valid for ATSR-2

VALID_VIEW_MODES = ('fward', 'nadir')

chkRasterSCC(*scc_threshold*, *scc_threshold_feat*, *mask*, *fmt*)

Check the ATSR2 raster snow/cloud coverage.

The method applies on the orthorectified and resampled subset, covering the GCP the ATSR2 raster class is constructed for. If the snow/cloud coverage exceeds either the scc_threshold or the scc_threshold_feat, which refers to the GCP itself, the to_be_proc flag is set to False. The snow/cloud mask is written to a raster file.

Parameters

- **scc_threshold** (*float*) – snow/cloud cover threshold [%]
- **scc_threshold_feat** (*float*) – snow/cloud cover threshold of the GCP feature [%]
- **mask** (*string*) – path to the extended GCP feature mask covering the input control window
- **fmt** (*string*) – format of the snow/cloud mask raster file

Raises

`IOError`: if the ATSR2 raster file is missing
`ValueError`: if the extended GCP feature mask does not match the snow/cloud mask window size

2.4 config_log Module

This module contains a function managing the project logger.

2.4.1 Module API

config_log.**log_cfg_load**()
Load the GeoAcca logging configuration from the configparser-format file `CONFIG_LOG_FILE`.

By default the logs are written to `GEOACCA.LOG` in your `HOME` directory. To redirect the logs set the environment variable `GEOACCA_LOGFILE`.

Note: Adapt `CONFIG_LOG_FILE` to customize your logging configuration.

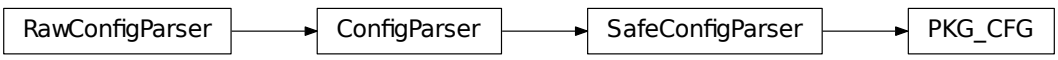
Raises

`IOError`: if configuration `CONFIG_LOG_FILE` not found

2.5 config_pkg Module

This module contains a class managing the GeoAcca package configuration.

2.5.1 Classes and Inheritance Structure



2.5.2 Module API

class config_pkg.**PKG_CFG**
Bases: `ConfigParser.SafeConfigParser`

GeoAcca package configuration class.

It subclasses the SafeConfigParser class, extending it for a method to load the GeoAcca configuration settings.

load(*view_mode='nadir'*, *cfg_section_names=None*, *aux_section_names=None*)
Set up the GeoAcca package configuration.

Note: Adapt `CONFIG_PKG_FILE` to customize your package configuration.

Parameters

- **cfg_section_names** (*list*) – section(s) to be loaded from the configparser-format file `CONFIG_PKG_FILE` (optional, default: None loads all)
- **aux_section_names** (*list*) – auxiliary data section(s) to be added to the package configuration (optional, default: None loads all)

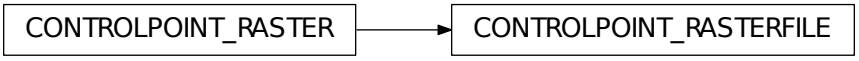
Returns package configuration

Return type dictionary

2.6 controlpoint_raster Module

This module contains a class for the initialisation of a reference raster file including raster file extraction and resampling, extraction of the corresponding water body data and plotting of quicklooks.

2.6.1 Classes and Inheritance Structure



2.6.2 Module API

class `controlpoint_raster.`**CONTROLPOINT_RASTERFILE**(**args*, ***kwargs*)
Bases: `geoacca.db.CONTROLPOINT_RASTER`

A class to work with reference raster files.

Parameters

- ***args** – variable length argument list.
- ****kwargs** – arbitrary keyword arguments see below

Keyword Arguments

- *ref_in_data_dir* (`string`) – path to the reference file directory
- *ref_filename* (`string`) – name of the reference file
- *ref_out_data_dir* (`string`) – path to the output directory

- *metadata_file* (`string`) – metadata file
- *PGDB* (`class`) – PostGres DB object
- *sensor* (`string`) – name of the reference EO sensor
- *band_nir* (`integer`) – near infrared band number of the reference file
- more valid ***kwargs*

Raises

`IOError`: if file could not be found

`extractWBDShape` (*shapefile*, *fmt*)

Create a shape, a mask and an extended mask file of the given GCP outline in UTM projection. The fitting outline is selected from a GCP outline collection shape. The masks have the dimensions of the raster file to be compared with. The water mask of the extended mask is streched by two kilometers.

Parameters

- **`shapefile`** (*string*) – GeoAcca GCP outline collection
- **`fmt`** (*string*) – mask output format

Raises

`CalledProcessError`: if an external function call failed

`loadRasterVar` (*lines*, *auxitems*)

Load class variables from file.

Parameters

- **`lines`** (*int*) – number of lines in the product header containing metadata
- **`auxitems`** (*dict*) – metadata to be loaded

`plotOrthoRasterfileQKL` ()

Plot a quicklook of the orthorectified and resampled subset, covering the GCP the reference raster file class is constructed for, overlaid with the GCP outline.

Raises

`IOError`: if the GCP shapefile could not be found

`resampUTMRaster` (*resampling*)

Load the reference raster file. Extract a spatial subset of the raster, covering the given GCP. The reference raster subset is resampled and reprojected to UTM projection.

Parameters **`resampling`** (*string*) – resampling method (options: Nearest, Bilinear or Bicubic)

Raises

`CalledProcessError`: if an external function call failed

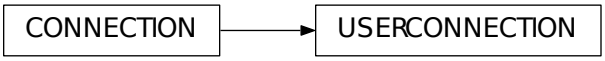
`Error`: if an unexpected error appeared

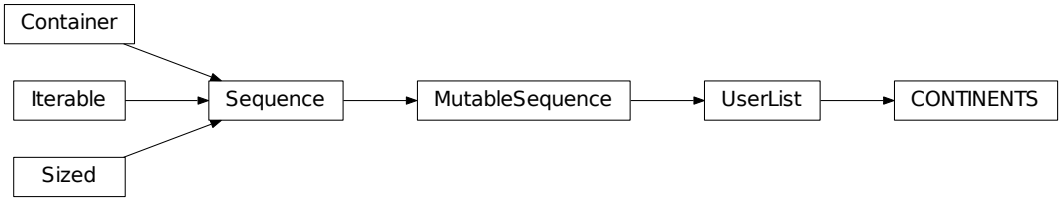
`IOError`: if input or output file could not be opened

2.7 `db` Module

This module contains classes handling the database access.

2.7.1 Classes and Inheritance Structure





Note: Inheritance structure of CONTINENTS also applies for COUNTRIES, CONTROLPOINTS, SENSORS, CONTROLPOINT_RASTERS, TRACKS, CPSMTRS, RASTERS, COMPOSITE_RASTERS, METHODS, SHIFTS

2.7.2 Module API

class db.**COMPOSITE_RASTER**(***kwargs*)

Bases: object

A class handling a DB COMPOSITE RASTER object.

Parameters ****kwargs** – arbitrary keyword arguments see below

Keyword Arguments

- *id* (integer) – composite raster ID as given in the DB
- *controlpoint_id* (string) – GCP ID
- *sensor_id* (integer) – sensor ID
- *acquisition_time* (string) – acquisition time at the GCP center point
- *proc_time* (string) – product generation time
- *proc_center* (string) – processing center
- *software_ver* (string) – software version used by the processing center
- *view_mode* (string) – direction of sensor view (options: nadir or forward)
- *sun_elev* (float) – sun elevation angle at the GCP center point [deg]
- *sun_azimuth* (float) – sun azimuth angle at the GCP center point [deg]
- *view_elev* (float) – sensor view elevation angle at the GCP center point [deg]

- *view_azimuth* (`float`) – sensor view azimuth angle at the GCP center point [deg]
- *absdir* (`string`) – path to the composite raster file directory
- *filenm* (`string`) – name of the composite raster file
- *psz* (`integer`) – pixel size [m]
- *pix_num_x* (`integer`) – number of pixels in x-direction
- *pix_num_y* (`integer`) – number of pixels in y-direction
- *pix_num_flagged* (`integer`) – number of pixels flagged due to snow or cloud
- *pix_num_none* (`integer`) – number of NaN pixels in the NIR band
- *xmin_geo* (`float`) – longitude of the GCP lower left corner point [deg]
- *ymin_geo* (`float`) – latitude of the GCP lower left corner point [deg]
- *xmax_geo* (`float`) – longitude of the GCP upper right corner point [deg]
- *ymax_geo* (`float`) – latitude of the GCP upper right corner point [deg]
- *srid_geo* (`integer`) – PostGIS Spatial Reference System Identifier of geographic coordinates
- *xmin_utm* (`float`) – UTM x coordinate of the GCP lower left corner point [m]
- *ymin_utm* (`float`) – UTM y coordinate of the GCP lower left corner point [m]
- *xmax_utm* (`float`) – UTM x coordinate of the GCP upper right corner point [m]
- *ymax_utm* (`float`) – UTM y coordinate of the GCP upper right corner point [m]
- *srid_utm* (`integer`) – PostGIS Spatial Reference System Identifier of UTM coordinates
- *zone_utm* (`integer`) – UTM zone
- *hemisphere* (`string`) – either 'N'orth or 'S'outh
- *to_be_proc* (`bool`) – flag to enable or disable matching
- *sensing_start* (`string`) – sensing start
- *sensing_stop* (`string`) – sensing stop
- *last_insert* (`string`) – timestamp of last insert
- *geoacca_ver* (`string`) – GeoAcca version

Raises

`IOError`: if the controlpoint_raster file path is invalid

delete(*CURSOR*, *mode*)

Delete in DB.

Parameters

- **CURSOR** (*class*) – DB CURSOR object
- **mode** (*string*) – DB request mode (options: ID, IDX, IDX1 or IDXI)

Returns number of rows deleted

Return type int

Raises

`ValueError`: if the mode option is invalid

Choose the `mode` option depending on the attributes known. See the detailed option list below.

mode	requires
ID	composite raster id
IDX	controlpoint id, sensor id, acquisition time, processing time, processing center, software version and view mode
IDX1	absdir and filenm
IDXI	controlpoint id, sensor id, processing time, processing center, software version, view mode, sensing start and sensing stop

insert (*CURSOR*)
Insert into DB.

Parameters **CURSOR** (*class*) – DB cursor object

Raises

`ValueError`: if SRID is not 4326

iprint (*stdout=True*)
Print.

Parameters **stdout** (*bool*) – True, if the output should be piped to stdout, otherwise False

Returns string to print

Return type string

select (*CURSOR*, *mode*)
Select from DB.

Parameters

- **CURSOR** (*class*) – DB cursor object
- **mode** (*string*) – DB request mode (options: ID, IDX, IDX1 or IDXI)

Raises

`IOError`: if the composite raster file path is invalid
`ValueError`: if the mode option is invalid or attributes are missing

For details on mode options *see*.

update (*CURSOR*, ***kwargs*)
Update in DB.

Parameters

- **CURSOR** (*class*) – DB cursor object
- ****kwargs** – arbitrary keyword arguments. Valid arguments are the same as for the object constructor.

Raises

`ValueError`: if ID is invalid or missing
`KeyError`: if an invalid object attribute is given

class db.**COMPOSITE_RASTERS** (***kwargs*)
Bases: `UserList.UserList`

A class handling a list of DB COMPOSITE RASTER objects.

Parameters ****kwargs** – arbitrary keyword arguments see below

Keyword Arguments

- *controlpoint_id* (`string`) – GCP ID
- *sensor_id* (`integer`) – sensor ID

iprint (*stdout=True*)
Print object list.

Parameters **stdout** (*bool*) – True, if the output should be piped to stdout, otherwise False

Returns string to print

Return type string

select (*CURSOR*, *mode*)
Select object list from DB.

Parameters

- **CURSOR** (*class*) – DB cursor object
- **mode** (*string*) – DB request mode (options: MD1, MD2 or ALL)

Returns True, if the data are returned, otherwise False

Return type bool

Choose the `mode` option depending on the attributes known. See the detailed option list below. Order direction is ascending.

mode	requires	order
MD1	controlpoint id and sensor_id	data is ordered by acquisition time, processing time, processing center, software version and view mode
MD2	controlpoint id	data is ordered by sensor_id, acquisition time, processing time, processing center, software version and view mode
ALL	nothing	data is ordered by controlpoint id, sensor_id, acquisition time, processing time, processing center, software version and view mode

class db.**CONNECTION** (*api*, *debug*, ***kwargs*)
A class handling a DB CONNECTION object.

Construct a CONNECTION object and connect to the DB addressed using the api stated.

Parameters

- **api** (*string*) – application interface name
- **debug (optional)** (*bool*) – True, to turn DB interface output on, otherwise False
- ****kwargs** – arbitrary keyword arguments see below

Keyword Arguments

- *user* (`string`) – database user
- *password* (`string`) – database password
- *host* (`string`) – database host
- *port* (`string`) – database port
- *database* (`string`) – database name

cursor ()
Request a DB CURSOR object.

Returns DB cursor

Return type CURSOR object

debug = False

reconnect = True

class db.**CONTINENT**(***kwargs*)
Bases: object

A class handling a DB CONTINENT object.

Parameters ****kwargs** – arbitrary keyword arguments see below

Keyword Arguments

- *id* (integer) – continent ID as given in the DB
- *name* (string) – continent name
- *last_insert* (string) – timestamp of last insert

delete(*CURSOR*, *mode*)
Delete in DB.

Parameters

- **CURSOR** (*class*) – DB CURSOR object
- **mode** (*string*) – DB request mode (options: ID or IDX)

Returns number of rows deleted

Return type integer

Raises

ValueError: if the mode option is invalid

Choose the mode option depending on the attributes known. See the detailed option list below.

mode	requires
ID	continent id
IDX	continent name

insert(*CURSOR*)
Insert into DB.

Parameters **CURSOR** (*class*) – DB cursor object

iprint(*stdout=True*)
Print.

Parameters **stdout** (*bool*) – True, if the output should be piped to stdout, otherwise False

Returns string to print

Return type string

select(*CURSOR*, *mode*)
Select from DB.

Parameters

- **CURSOR** (*class*) – DB cursor object
- **mode** (*string*) – DB request mode (options: ID or IDX)

Raises

`ValueError`: if the mode option is invalid

For details on mode options *see*.

update (*CURSOR*, ***kwargs*)
Update in DB.

Parameters

- **CURSOR** (*class*) – DB cursor object
- ****kwargs** – arbitrary keyword arguments. Valid arguments are the same as for the object constructor.

Raises

`ValueError`: if ID is invalid or missing
`KeyError`: if an invalid object attribute is given

class db.**CONTINENTS** (*initlist=None*)
Bases: `UserList.UserList`

A class handling a list of DB CONTINENT objects.

delete (*CURSOR*, *mode*)
Delete all objects in DB.

Parameters

- **CURSOR** (*class*) – DB CURSOR object
- **mode** (*string*) – mode (options: ALL)

Returns number of rows deleted

Return type integer

Raises

`ValueError`: if the mode option is invalid

iprint (*stdout=True*)
Print object list.

Parameters **stdout** (*bool*) – True, if the output should be piped to stdout, otherwise False

Returns string to print

Return type string

select (*CURSOR*)
Select object list from DB.

Parameters **CURSOR** (*class*) – DB cursor object

Returns True, if the data are returned, otherwise False

Return type bool

class db.**CONTROLPOINT** (***kwargs*)
Bases: `object`

A class handling a DB CONTROLPOINT object.

Parameters ****kwargs** – arbitrary keyword arguments see below

Keyword Arguments

- *id* (`string`) – GCP ID as given in the DB
- *country_id* (`integer`) – country ID
- *name* (`string`) – GCP name
- *srid* (`integer`) – PostGIS Spatial Reference System Identifier
- *x* (`float`) – longitude of the GCP [deg]
- *y* (`float`) – latitude of the GCP [deg]
- *last_insert* (`string`) – timestamp of last insert

delete(*CURSOR*, *mode*)
Delete in DB.

Parameters

- **CURSOR** (*class*) – DB CURSOR object
- **mode** (*string*) – DB request mode (options: ID or IDX)

Returns number of rows deleted

Return type int

Raises

`ValueError`: if the mode option is invalid or SRID is not 4326

Choose the `mode` option depending on the attributes known. See the detailed option list below.

mode	requires
ID	controlpoint id
IDX	latitude, longitude [deg] of controlpoint

insert(*CURSOR*)
Insert into DB.

Parameters **CURSOR** (*class*) – DB cursor object

Raises

`ValueError`: if SRID is not 4326

iprint(*stdout=True*)
Print.

Parameters **stdout** (*bool*) – True, if the output should be piped to stdout, otherwise False

Returns string to print

Return type string

select(*CURSOR*, *mode*)
Select from DB with its geographical coordinates.

Parameters

- **CURSOR** (*class*) – DB cursor object
- **mode** (*string*) – DB request mode (options: ID or IDX)

Raises

`ValueError`: if the mode option is invalid

For details on mode options *see*.

update(*CURSOR*, ***kwargs*)
Update in DB.

Parameters

- **CURSOR** (*class*) – DB cursor object
- ****kwargs** – arbitrary keyword arguments. Valid arguments are the same as for the object constructor.

Raises

`ValueError`: if ID is invalid or missing
`KeyError`: if an invalid object attribute is given

class db.**CONTROLPOINTS**(*initlist=None*)
Bases: `UserList.UserList`

A class handling a list of DB CONTROLPOINT objects.

delete(*CURSOR*, *mode*)
Delete all objects in DB.

Parameters

- **CURSOR** (*class*) – DB CURSOR object
- **mode** (*string*) – mode (options: ALL)

Returns number of rows deleted

Return type integer

Raises

`ValueError`: if the mode option is invalid

iprint(*stdout=True*)
Print object list.

Parameters **stdout** (*bool*) – True, if the output should be piped to stdout, otherwise False

Returns string to print

Return type string

select(*CURSOR*)
Select object list from DB with its geographical coordinates.

Parameters **CURSOR** (*class*) – DB cursor object

Returns True, if the data are returned, otherwise False

Return type bool

class db.**CONTROLPOINT_RASTER**(***kwargs*)
Bases: `object`

A class handling a DB CONTROLPOINT_RASTER object.

Parameters ****kwargs** – arbitrary keyword arguments see below

Keyword Arguments

- *id* (`integer`) – controlpoint_raster ID as given in the DB
- *controlpoint_id* (`string`) – GCP ID

- *sensor_id* (`integer`) – sensor ID
- *acquisition_time* (`string`) – acquisition time at the GCP center point
- *proc_time* (`string`) – product generation time
- *proc_center* (`string`) – processing center
- *software_ver* (`string`) – software version
- *sun_elev* (`float`) – sun elevation angle at the GCP center point [deg]
- *psz* (`integer`) – pixel size [m]
- *absdir* (`string`) – path to the controlpoint_raster file directory
- *filenm* (`string`) – name of the controlpoint_raster file
- *xmean_geo* (`float`) – longitude of the GCP center point
- *ymean_geo* (`float`) – latitude of the GCP center point
- *xmin_geo* (`float`) – longitude of the GCP lower left corner point [deg]
- *ymin_geo* (`float`) – latitude of the GCP lower left corner point [deg]
- *xmax_geo* (`float`) – longitude of the GCP upper right corner point [deg]
- *ymax_geo* (`float`) – latitude of the GCP upper right corner point [deg]
- *srid_geo* (`integer`) – PostGIS Spatial Reference System Identifier of geographic coordinates
- *xmean_utm* (`float`) – UTM x coordinate of the GCP center point [m]
- *ymean_utm* (`float`) – UTM y coordinate of the GCP center point [m]
- *xmin_utm* (`float`) – UTM x coordinate of the GCP lower left corner point [m]
- *ymin_utm* (`float`) – UTM y coordinate of the GCP lower left corner point [m]
- *xmax_utm* (`float`) – UTM x coordinate of the GCP upper right corner point [m]
- *ymax_utm* (`float`) – UTM y coordinate of the GCP upper right corner point [m]
- *srid_utm* (`integer`) – PostGIS Spatial Reference System Identifier of UTM coordinates
- *zone_utm* (`integer`) – UTM zone
- *hemisphere* (`string`) – either 'N'orth or 'S'outh
- *last_insert* (`string`) – timestamp of last insert

Raises

`IOError`: if the controlpoint_raster file path is invalid

delete(*CURSOR*, *mode*)

Delete in DB.

Parameters

- **CURSOR** (*class*) – DB CURSOR object
- **mode** (*string*) – DB request mode (options: ID, IDX or IDX1)

Returns number of rows deleted

Return type int

Raises

`ValueError`: if the mode option is invalid

Choose the `mode` option depending on the attributes known. See the detailed option list below.

mode	requires
ID	controlpoint_raster id
IDX	controlpoint id, sensor id and acquisition time
IDX1	absdir and filenm

insert (*CURSOR*)
Insert into DB.

Parameters **CURSOR** (*class*) – DB cursor object

Raises

`ValueError`: if SRID is not 4326

iprint (*stdout=True*)
Print.

Parameters **stdout** (*bool*) – True, if the output should be piped to stdout, otherwise False

Returns string to print

Return type string

select (*CURSOR*, *mode*, *tmstp=None*, *sun_elev=None*)
Select from DB.

Parameters

- **CURSOR** (*class*) – DB cursor object
- **mode** (*string*) – DB request mode (options: ID, IDX or IDX1)
- **tmstp** (*string*) – timestamp to look for
- **sun_elev** (*float*) – sun elevation to look for [deg]

Raises

`IOError`: if the controlpoint_raster file path is invalid

`ValueError`: if the mode option is invalid, attributes are missing or timestamp format is not readable

For details on mode options *see*.

update (*CURSOR*, ***kwargs*)
Update in DB.

Parameters

- **CURSOR** (*class*) – DB cursor object
- ****kwargs** – arbitrary keyword arguments. Valid arguments are the same as for the object constructor.

Raises

`ValueError`: if ID is invalid or missing

`KeyError`: if an invalid object attribute is given

class db.**CONTROLPOINT_RASTERS**(***kwargs*)
Bases: UserList.UserList

A class handling a list of DB CONTROLPOINT_RASTER objects.

Parameters ****kwargs** – arbitrary keyword arguments see below

Keyword Arguments

- *controlpoint_id* (string) – GCP ID
- *sensor_id* (integer) – sensor ID

delete(*CURSOR*, *mode*)
Delete all objects in DB.

Parameters

- **CURSOR** (*class*) – DB CURSOR object
- **mode** (*string*) – mode (options: ALL)

Returns number of rows deleted

Return type int

Raises

ValueError: if the mode option is invalid

iprint(*stdout=True*)
Print object list.

Parameters **stdout** (*bool*) – True, if the output should be piped to stdout, otherwise False

Returns string to print

Return type string

select(*CURSOR*, *mode*)
Select object list from DB.

Parameters

- **CURSOR** (*class*) – DB cursor object
- **mode** (*string*) – DB request mode (options: MD1, MD2 or ALL)

Returns True, if the data are returned, otherwise False

Return type bool

Choose the mode option depending on the attributes known. See the detailed option list below. Order direction is ascending.

mode	requires	order
MD1	controlpoint id	data is ordered by sensor id and acquisition time
MD2	controlpoint id, sensor id	data is ordered by acquisition time
ALL	nothing	data is ordered by controlpoint id, sensor id and acquisition time

class db.**COUNTRIES**(*initlist=None*)
Bases: UserList.UserList

A class handling a list of DB COUNTRY objects.

delete(*CURSOR*, *mode*)
Delete all object in DB.

Parameters

- **CURSOR** (*class*) – DB CURSOR object
- **mode** (*string*) – mode (options: ALL)

Returns number of rows deleted

Return type integer

Raises

ValueError: if the mode option is invalid

iprint (*stdout=True*)
Print object list.

Parameters **stdout** (*bool*) – True, if the output should be piped to stdout, otherwise False

Returns string to print

Return type string

select (*CURSOR*)
Select object list from DB.

Parameters **CURSOR** (*class*) – DB cursor object

Returns True, if the data are returned, otherwise False

Return type bool

class db.**COUNTRY** (***kwargs*)
Bases: object

A class handling a DB COUNTRY object.

Parameters ****kwargs** – arbitrary keyword arguments see below

Keyword Arguments

- *id* (integer) – country ID as given in the DB
- *continent_id* (integer) – continent ID
- *name* (string) – country name
- *last_insert* (string) – timestamp of last insert

delete (*CURSOR*, *mode*)
Delete in DB.

Parameters

- **CURSOR** (*class*) – DB CURSOR object
- **mode** (*string*) – DB request mode (options: ID or IDX)

Returns number of rows deleted

Return type integer

Raises

ValueError: if the mode option is invalid

Choose the `mode` option depending on the attributes known. See the detailed option list below.

mode	requires
ID	country id
IDX	country name, continent id

insert (*CURSOR*)
Insert into DB.

Parameters **CURSOR** (*class*) – DB cursor object

iprint (*stdout=True*)
Print.

Parameters **stdout** (*bool*) – True, if the output should be piped to stdout, otherwise False

Returns string to print

Return type string

select (*CURSOR*, *mode*)
Select from DB.

Parameters

- **CURSOR** (*class*) – DB cursor object
- **mode** (*string*) – DB request mode (options: ID or IDX)

Raises

`ValueError`: if the mode option is invalid

For details on mode options *see*.

update (*CURSOR*, ***kwargs*)
Update in DB.

Parameters

- **CURSOR** (*class*) – DB cursor object
- ****kwargs** – arbitrary keyword arguments. Valid arguments are the same as for the object constructor.

Raises

`ValueError`: if ID is invalid or missing
`KeyError`: if an invalid object attribute is given

class db.**CPMTR** (***kwargs*)
Bases: `object`

A class handling a DB CPMTR object.

Parameters ****kwargs** – arbitrary keyword arguments see below

Keyword Arguments

- *id* (`integer`) – cpmtr ID as given in the DB
- *controlpoint_id* (`string`) – GCP ID
- *track_id* (`integer`) – track ID
- *last_insert* (`string`) – timestamp of last insert

delete(*CURSOR*, *mode*)
Delete in DB.

Parameters

- **CURSOR** (*class*) – DB CURSOR object
- **mode** (*string*) – DB request mode (options: ID or IDX)

Returns number of rows deleted

Return type int

Raises

`ValueError`: if the mode option is invalid

Choose the `mode` option depending on the attributes known. See the detailed option list below.

mode	requires
ID	cpmtr id
IDX	controlpoint id and track id

insert(*CURSOR*)
Insert into DB.

Parameters **CURSOR** (*class*) – DB cursor object

iprint(*stdout=True*)
Print.

Parameters **stdout** (*bool*) – True, if the output should be piped to stdout, otherwise False

Returns string to print

Return type string

select(*CURSOR*, *mode*)
Select from DB.

Parameters

- **CURSOR** (*class*) – DB cursor object
- **mode** (*string*) – DB request mode (options: ID or IDX)

Raises

`ValueError`: if the mode option is invalid

For details on mode options *see*.

update(*CURSOR*, ***kwargs*)
Update in DB.

Parameters

- **CURSOR** (*class*) – DB cursor object
- ****kwargs** – arbitrary keyword arguments. Valid arguments are the same as for the object constructor.

Raises

`ValueError`: if ID is invalid or missing

`KeyError`: if an invalid object attribute is given

class db.**CPSMTRS**(***kwargs*)
Bases: UserList.UserList

A class handling a list of DB CPSMTRS objects.

Parameters ****kwargs** – arbitrary keyword arguments see below

Keyword Arguments

- *controlpoint_id* (string) – GCP ID
- *track_id* (integer) – track ID

delete(*CURSOR*, *mode*)
Delete all objects in DB.

Parameters

- **CURSOR** (*class*) – DB CURSOR object
- **mode** (*string*) – mode (options: ALL)

Returns number of rows deleted

Return type int

Raises

ValueError: if the mode option is invalid

iprint(*stdout=True*)
Print object list.

Parameters **stdout** (*bool*) – True, if the output should be piped to stdout, otherwise False

Returns string to print

Return type string

select(*CURSOR*, *mode*)
Select object list from DB.

Parameters **CURSOR** (*class*) – DB cursor object

Mode string mode DB request mode (options: MD1, MD2 or ALL)

Returns True, if the data are returned, otherwise False

Return type bool

Choose the mode option depending on the attributes known. See the detailed option list below. Order direction is ascending.

mode	requires	order
MD1	controlpoint id	data is ordered by track id
MD2	track id	data is ordered by controlpoint id
ALL	nothing	data is ordered by controlpoint id and track id

class db.**CURSOR**(*connection*, *adapter*, *debug*)
Bases: object

A class handling a DB CURSOR object.

Parameters

- **connection** (*class*) – DB connection object
- **adapter** (*bool*) – True, if adapter has been set, otherwise False

- **debug (optional)** (*bool*) – True, to turn DB interface output on, otherwise False

debug = False

debug_query(*query*, *args*)
Log the DB query.

Parameters

- **query** (*string*) – DB query
- **args** (*string*) – variable length argument list

delete(*query*, **args*)
Delete in DB.

Parameters

- **query** (*string*) – DB delete query
- ***args** (*string*) – variable length argument list for DB query

fetchall(*query*, **args*)
Fetch all rows of a DB select query result.

Parameters

- **query** (*string*) – DB select query
- ***args** (*string*) – variable length argument list for DB query for DB query

Yields

dictionary rows of parameter, value pairs loaded

fetchmany(*query*, **args*)
Fetch the next set of rows of a DB select query result.

The number of rows fetch per call is specified by the cursor's arraysize attribute.

Parameters

- **query** (*string*) – DB select query
- ***args** (*string*) – variable length argument list for DB query for DB query

Yields

dictionary rows of parameter, value pairs loaded

fetchone(*query*, **args*)
Fetch the next row of a DB select query result.

Parameters

- **query** (*string*) – DB select query
- ***args** (*string*) – variable length argument list for DB query

Returns row of parameter, value pairs loaded, or None

Return type dictionary

insert(*query*, **args*)
Insert in DB.

Parameters

- **query** (*string*) – DB insert query

- ***args** (*string*) – variable length argument list for DB query

next_refresh = 0

refresh()
Refresh the DB connection.

refresh_time = 60

update(*query*, **args*)
Update in DB.

Parameters

- **query** (*string*) – DB update query
- ***args** (*string*) – variable length argument list for DB query

class db.**METHOD**(***kwargs*)
Bases: object

A class handling a DB METHOD object.

Parameters ****kwargs** – arbitrary keyword arguments see below

Keyword Arguments

- *id* (integer) – method ID as given in the DB
- *name* (string) – method name
- *last_insert* (string) – timestamp of last insert

VALID_METHOD_TYPES = ('TM_CCORR_NORMED', 'TM_CCOEFF_NORMED')

delete(*CURSOR*, *mode*)
Delete in DB.

Parameters

- **CURSOR** (*class*) – DB CURSOR object
- **mode** (*string*) – DB request mode (options: ID or IDX)

Returns number of rows deleted

Return type int

Raises

ValueError: if the mode option is invalid

Choose the mode option depending on the attributes known. See the detailed option list below.

mode	requires
ID	method id
IDX	method name

insert(*CURSOR*)
Insert into DB.

Parameters **CURSOR** (*class*) – DB cursor object

iprint(*stdout=True*)
Print.

Parameters **stdout** (*bool*) – True, if the output should be piped to stdout, otherwise False

Returns string to print

Return type string

select (*CURSOR*, *mode*)
Select from DB.

Parameters

- **CURSOR** (*class*) – DB cursor object
- **mode** (*string*) – DB request mode (options: ID or IDX)

Raises

ValueError: if the mode option is invalid

For details on mode options *see*.

update (*CURSOR*, ***kwargs*)
Update in DB.

Parameters

- **CURSOR** (*class*) – DB cursor object
- ****kwargs** – arbitrary keyword arguments. Valid arguments are the same as for the object constructor.

Raises

ValueError: if ID is invalid or missing
KeyError: if an invalid object attribute is given

class db.**METHODS** (*initlist=None*)
Bases: UserList.UserList

A class handling a list of DB METHOD objects.

delete (*CURSOR*, *mode*)
Delete all objects in DB.

Parameters

- **CURSOR** (*class*) – DB CURSOR object
- **mode** (*string*) – mode (options: ALL)

Returns number of rows deleted

Return type int

Raises

ValueError: if the mode option is invalid

iprint (*stdout=True*)
Print object list.

Parameters **stdout** (*bool*) – True, if the output should be piped to stdout, otherwise False

Returns string to print

Return type string

select (*CURSOR*)
Select object list from DB.

Parameters **CURSOR** (*class*) – DB cursor object

Returns True, if the data are returned, otherwise False

Return type bool

class db.**RASTER**(***kwargs*)

Bases: object

A class handling a DB RASTER object.

Parameters ****kwargs** – arbitrary keyword arguments see below

Keyword Arguments

- *id* (integer) – raster ID as given in the DB
- *controlpoint_id* (string) – GCP ID
- *track_id* (integer) – track ID
- *acquisition_time* (string) – acquisition time at the GCP center point
- *acquisition_station* (string) – acquisition station
- *proc_time* (string) – product generation time
- *proc_center* (string) – processing center
- *software_ver* (string) – software version used by the processing center
- *view_mode* (string) – direction of sensor view (options: nadir or forward)
- *phase* (integer) – phase
- *cycle* (integer) – cycle number
- *rel_orbit* (integer) – relative orbit number
- *abs_orbit* (integer) – absolute orbit number
- *sun_elev* (float) – sun elevation angle at the GCP center point [deg]
- *sun_azimuth* (float) – sun azimuth angle at the GCP center point [deg]
- *view_elev* (float) – sensor view elevation angle at the GCP center point [deg]
- *view_azimuth* (float) – sensor view azimuth angle at the GCP center point [deg]
- *orbit_state_vector_file* (string) – auxiliary information on orbit state vector
- *digital_elevation_model_file* (string) – auxiliary information on DEM
- *across_track_pixel_pos* (integer) – across track pixel position at the GCP center point
- *orbit_ic_angle* (float) – orbit inclination angle at the GCP center point [deg]
- *absdir* (string) – path to the raster file directory
- *filenm* (string) – name of the raster file
- *psz* (integer) – pixel size [m]
- *pix_num_x* (integer) – number of pixels in x-direction
- *pix_num_y* (integer) – number of pixels in y-direction
- *pix_num_flagged* (integer) – number of pixels flagged due to snow or cloud
- *pix_num_none* (integer) – number of NaN pixels in the NIR band
- *xmin_geo* (float) – longitude of the GCP lower left corner point [deg]
- *ymin_geo* (float) – latitude of the GCP lower left corner point [deg]

- *xmax_geo* (`float`) – longitude of the GCP upper right corner point [deg]
- *ymax_geo* (`float`) – latitude of the GCP upper right corner point [deg]
- *srid_geo* (`integer`) – PostGIS Spatial Reference System Identifier of geographic coordinates
- *xmin_utm* (`float`) – UTM x coordinate of the GCP lower left corner point [m]
- *ymin_utm* (`float`) – UTM y coordinate of the GCP lower left corner point [m]
- *xmax_utm* (`float`) – UTM x coordinate of the GCP upper right corner point [m]
- *ymax_utm* (`float`) – UTM y coordinate of the GCP upper right corner point [m]
- *srid_utm* (`integer`) – PostGIS Spatial Reference System Identifier of UTM coordinates
- *zone_utm* (`integer`) – UTM zone
- *hemisphere* (`string`) – either 'N'orth or 'S'outh
- *to_be_proc* (`bool`) – flag to enable or disable matching
- *sensing_start* (`string`) – sensing start
- *sensing_stop* (`string`) – sensing stop
- *last_insert* (`string`) – timestamp of last insert
- *geoacca_ver* (`string`) – GeoAcca version

Raises

`IOError`: if the controlpoint_raster file path is invalid

delete(*CURSOR*, *mode*)

Delete in DB.

Parameters

- **CURSOR** (*class*) – DB CURSOR object
- **mode** (*string*) – DB request mode (options: ID, IDX, IDX1 or IDXI)

Returns number of rows deleted

Return type int

Raises

`ValueError`: if the mode option is invalid

Choose the `mode` option depending on the attributes known. See the detailed option list below.

mode	requires
ID	raster id
IDX	controlpoint id, track id, acquisition time, processing time, processing center, software version and view mode
IDX1	absdir and filenm
IDXI	controlpoint id, track id, processing time, processing center, software version, view mode, sensing start and sensing stop

insert(*CURSOR*)

Insert into DB.

Parameters **CURSOR** (*class*) – DB cursor object

Raises

ValueError: if SRID is not 4326

iprint (*stdout=True*)
Print.

Parameters **stdout** (*bool*) – True, if the output should be piped to stdout, otherwise False

Returns string to print

Return type string

select (*CURSOR*, *mode*)
Select from DB.

Parameters

- **CURSOR** (*class*) – DB cursor object
- **mode** (*string*) – DB request mode (options: ID, IDX, IDX1 or IDXI)

Raises

IOError: if the raster file path is invalid
ValueError: if the mode option is invalid or attributes are missing

For details on mode options *see*.

update (*CURSOR*, ***kwargs*)
Update in DB.

Parameters

- **CURSOR** (*class*) – DB cursor object
- ****kwargs** – arbitrary keyword arguments. Valid arguments are the same as for the object constructor.

Raises

ValueError: if ID is invalid or missing
KeyError: if an invalid object attribute is given

class db.**RASTERS** (***kwargs*)
Bases: UserList.UserList

A class handling a list of DB RASTER objects.

Parameters ****kwargs** – arbitrary keyword arguments see below

Keyword Arguments

- *controlpoint_id* (string) – GCP ID
- *track_id* (integer) – track ID

iprint (*stdout=True*)
Print object list.

Parameters **stdout** (*bool*) – True, if the output should be piped to stdout, otherwise False

Returns string to print

Return type string

select (*CURSOR*, *mode*, *sensor_id=None*)
Select object list from DB.

Parameters

- **CURSOR** (*class*) – DB cursor object
- **mode** (*string*) – DB request mode (options: MD1, MD2, MD3 or ALL)
- **sensor_id** (*integer*) – sensor id

Returns True, if the data are returned, otherwise False

Return type bool

Choose the `mode` option depending on the attributes known. See the detailed option list below. Order direction is ascending.

mode	requires	order
MD1	controlpoint id and track_id	data is ordered by acquisition time, processing time, processing center, software version and view mode
MD2	controlpoint id	data is ordered by track id, acquisition time, processing time, processing center, software version and view mode
MD3	controlpoint id and sensor id	data is ordered by track_id, acquisition time, processing time, processing center, software version and view mode
ALL	nothing	data is ordered by controlpoint id, track_id, acquisition time, processing time, processing center, software version and view mode

class db.**SENSOR**(***kwargs*)

Bases: `object`

A class handling a DB SENSOR object.

Parameters ****kwargs** – arbitrary keyword arguments see below

Keyword Arguments

- *id* (`integer`) – sensor ID as given in the DB
- *name* (`string`) – sensor name
- *isreference* (`bool`) – True, if sensor is for reference, otherwise False
- *last_insert* (`string`) – timestamp of last insert

delete(*CURSOR*, *mode*)

Delete in DB.

Parameters

- **CURSOR** (*class*) – DB CURSOR object
- **mode** (*string*) – DB request mode (options: ID or IDX)

Returns number of rows deleted

Return type int

Raises

`ValueError`: if the mode option is invalid

Choose the `mode` option depending on the attributes known. See the detailed option list below.

mode	requires
ID	sensor id
IDX	sensor name

insert(*CURSOR*)

Insert into DB.

Parameters **CURSOR** (*class*) – DB cursor object

iprint (*stdout=True*)
Print.

Parameters **stdout** (*bool*) – True, if the output should be piped to stdout, otherwise False

Returns string to print

Return type string

select (*CURSOR*, *mode*)
Select from DB.

Parameters

- **CURSOR** (*class*) – DB cursor object
- **mode** (*string*) – DB request mode (options: ID or IDX)

Raises

`ValueError`: if the mode option is invalid

For details on mode options *see*.

update (*CURSOR*, ***kwargs*)
Update in DB.

Parameters

- **CURSOR** (*class*) – DB cursor object
- ****kwargs** – arbitrary keyword arguments. Valid arguments are the same as for the object constructor.

Raises

`ValueError`: if ID is invalid or missing
`KeyError`: if an invalid object attribute is given

class db.**SENSORS** (*initlist=None*)
Bases: `UserList.UserList`

A class handling a list of DB SENSOR objects.

delete (*CURSOR*, *mode*)
Delete all objects in DB.

Parameters

- **CURSOR** (*class*) – DB CURSOR object
- **mode** (*string*) – mode (options: ALL)

Returns number of rows deleted

Return type int

Raises

`ValueError`: if the mode option is invalid

iprint (*stdout=True*)
Print object list.

Parameters **stdout** (*bool*) – True, if the output should be piped to stdout, otherwise False

Returns string to print

Return type string

select(*CURSOR*)
Select object list from DB.

Parameters **CURSOR** (*class*) – DB cursor object

Returns True, if the data are returned, otherwise False

Return type bool

class db.**SHIFT**(***kwargs*)
Bases: object

A class handling a DB SHIFT object.

Parameters ****kwargs** – arbitrary keyword arguments see below

Keyword Arguments

- *id* (integer) – shift ID as given in the DB
- *raster_id* (integer) – raster ID
- *composite_raster_id* (integer) – composite raster ID
- *controlpoint_raster_id* (integer) – controlpoint raster ID
- *x_shift* (float) – shift in x direction [m]
- *y_shift* (float) – shift in y direction [m]
- *ac_track_shift* (float) – across track shift [m]
- *al_track_shift* (float) – along track shift [m]
- *c_coeff* (float) – cross correlation coefficient
- *method_id* (integer) – method ID
- *last_insert* (string) – timestamp of last insert

delete(*CURSOR*, *mode*)
Delete in DB.

Parameters

- **CURSOR** (*class*) – DB CURSOR object
- **mode** (*string*) – DB request mode (options: ID or IDX)

Returns number of rows deleted

Return type int

Raises

ValueError: if the mode option is invalid

Choose the mode option depending on the attributes known. See the detailed option list below.

mode	requires
ID	shift id
R_IDX	raster id, controlpoint raster id and method id
C_IDX	composite raster id, controlpoint raster id and method id

insert(*CURSOR*)
Insert into DB.

Parameters **CURSOR** (*class*) – DB cursor object

iprint (*stdout=True*)
Print.

Parameters **stdout** (*bool*) – True, if the output should be piped to stdout, otherwise False

Returns string to print

Return type string

select (*CURSOR*, *mode*)
Select from DB.

Parameters

- **CURSOR** (*class*) – DB cursor object
- **mode** (*string*) – DB request mode (options: ID or IDX)

Raises

`ValueError`: if the mode option is invalid

For details on mode options *see*.

update (*CURSOR*, ***kwargs*)
Update in DB.

Parameters

- **CURSOR** (*class*) – DB cursor object
- ****kwargs** – Arbitrary keyword arguments. Valid arguments are the same as for the object constructor.

Raises

`ValueError`: if ID is invalid or missing
`KeyError`: if an invalid object attribute is given

class db.**SHIFTS** (***kwargs*)
Bases: `UserList.UserList`

A class handling a list of DB SHIFT objects.

Parameters ****kwargs** – arbitrary keyword arguments see below

Keyword Arguments

- *raster_id* (`integer`) – raster ID
- *composite_raster_id* (`integer`) – composite raster ID
- *controlpoint_raster_id* (`integer`) – controlpoint raster ID
- *sensor_id* (`integer`) – sensor ID

iprint (*stdout=True*)
Print object list.

Parameters **stdout** (*bool*) – True, if the output should be piped to stdout, otherwise False

Returns string to print

Return type string

select (*CURSOR*, *mode*)
Select object list from DB.

Parameters **CURSOR** (*class*) – DB cursor object

Mode string mode DB request mode (options: MD1, MD2 or ALL)

Returns True, if the data are returned, otherwise False

Return type bool

Choose the `mode` option depending on the attributes known. See the detailed option list below. Order direction is ascending.

mode	requires	order
MD1	controlpoint raster id and method id	data is ordered by input raster id
MD2	raster id and method id	data is ordered by controlpoint raster id
MD3	composite raster id and method id	data is ordered by controlpoint raster id
ALL	nothing	data is ordered by input raster id, controlpoint raster id and method id

class db.**SPATIALREFSYS**(*CURSOR*, *srid*)

A class handling the DB SPATIALREFSYS objects.

class db.**TRACK**(***kwargs*)

Bases: `object`

A class handling a DB TRACK object.

Parameters ****kwargs** – arbitrary keyword arguments see below

Keyword Arguments

- *id* (`integer`) – track ID as given in the DB
- *sensor_id* (`integer`) – sensor ID
- *number* (`integer`) – track number
- *last_insert* (`string`) – timestamp of last insert

delete(*CURSOR*, *mode*)

Delete in DB.

Parameters

- **CURSOR** (*class*) – DB CURSOR object
- **mode** (*string*) – DB request mode (options: ID or IDX)

Returns number of rows deleted

Return type int

Raises

`ValueError`: if the mode option is invalid

Choose the `mode` option depending on the attributes known. See the detailed option list below.

mode	requires
ID	track id
IDX	sensor id and track number

insert(*CURSOR*)

Insert into DB.

Parameters **CURSOR** (*class*) – DB cursor object

iprint (*stdout=True*)
Print.

Parameters **stdout** (*bool*) – True, if the output should be piped to stdout, otherwise False

Returns string to print

Return type string

select (*CURSOR*, *mode*)
Select from DB.

Parameters

- **CURSOR** (*class*) – DB cursor object
- **mode** (*string*) – DB request mode (options: ID or IDX)

Raises

`ValueError`: if the mode option is invalid

For details on mode options *see*.

update (*CURSOR*, ***kwargs*)
Update in DB.

Parameters

- **CURSOR** (*class*) – DB cursor object
- ****kwargs** – arbitrary keyword arguments. Valid arguments are the same as for the object constructor.

Raises

`ValueError`: if ID is invalid or missing
`KeyError`: if an invalid object attribute is given

class db.**TRACKS** (***kwargs*)
Bases: `UserList.UserList`

A class handling a list of DB TRACK objects.

Parameters ****kwargs** – arbitrary keyword arguments see below

Keyword Arguments

- *id* (`integer`) – track ID as given in the DB
- *number* (`integer`) – track number
- *sensor_id* (`integer`) – sensor ID

delete (*CURSOR*, *mode*)
Delete all objects in DB.

Parameters

- **CURSOR** (*class*) – DB CURSOR object
- **mode** (*string*) – mode (options: ALL)

Returns number of rows deleted

Return type int

Raises

ValueError: if the mode option is invalid

iprint (*stdout=True*)
Print object list.

Parameters **stdout** (*bool*) – True, if the output should be piped to stdout, otherwise False

Returns string to print

Return type string

select (*CURSOR*, *mode*)
Select object list from DB.

Parameters **CURSOR** (*class*) – DB cursor object

Mode string mode DB request mode (options: MD1 or ALL)

Returns True, if the data are returned, otherwise False

Return type bool

Choose the mode option depending on the attributes known. See the detailed option list below. Order direction is ascending.

mode	requires	order
MD1	sensor_id	data is ordered by track id and number
ALL	nothing	data is ordered by track id, sensor id and number

class db.**USERCONNECTION** (*debug*, ***kwargs*)
Bases: db.CONNECTION

A class handling a DB USERCONNECTION object.

Construct a USERCONNECTION object and connect to a PostgreSQL DB.

Parameters

- **debug (optional)** (*bool*) – True, to turn DB interface output on, otherwise False
- ****kwargs** – arbitrary keyword arguments, see valid arguments *here*

debug = False

db.**dict_factory** (*cursor*, *row*)

2.8 georas Module

This module contains a collection of geo raster tools.

2.8.1 Module API

georas.**chkRastermPoly** (*fn*, *wktPOLYGON*)
Check if a polygon is completely covered by a raster file.

Parameters

- **fn** (*string*) – raster file
- **wktPOLYGON** (*string*) – WKT polygon in geographic coordinates

Returns True if the polygon is completely covered by the raster file, False otherwise

Raises

`IOError`: if the raster file could not be opened

`Error`: if an unexpected error appeared

`georas.`**`getRasterCornerCoord`**(*hDataset*, *hTransform*, *x*, *y*)

Get the upper left corner coordinates of a defined pixel of a raster file. Transform the coordinates to another projection if indicated by the parameter hTransform.

Parameters

- **`hDataset`** – gdal raster dataset
- **`hTransform`** – gdal transformation object for coordinate transformation
- **`x`** (*integer*) – index of raster pixel in x-direction
- **`y`** (*integer*) – index of raster pixel in y-direction

Returns longitude, latitude

Return type tuple

`georas.`**`loadRaster`**(*ifn*, *bands*)

Load a raster file to a data array.

Parameters

- **`ifn`** (*string*) – raster file
- **`bands`** (*list*) – bands to load

Returns data array, pixel number in x-direction, pixel number in y-direction

Return type tuple

Raises

`IOError`: if the raster file could not be opened

`georas.`**`reprojectUTMtoGeo`**(*ifn*, *srid*, *ofn*)

Reproject a raster file from UTM coordinates to geographic coordinates.

Parameters

- **`ifn`** (*string*) – input file
- **`srid`** (*integer*) – EPSG code of input file projection
- **`ofn`** (*string*) – output file

Raises

`RuntimeError`: if the original raster file could not be found

`CalledProcessError`: if the external function call failed

`georas.`**`writeRaster`**(*data*, *bands*, *srid*, *psz*, *xmin*, *ymax*, *ofn*, *fmt*)

Write a raster file from a data array.

Parameters

- **`data`** (*array*) – data array
- **`bands`** (*list*) – layers to store
- **`srid`** (*integer*) – EPSG code of output file projection
- **`psz`** (*float*) – pixel size of the output file

- **xmin** (*float*) – x-coordinate of the lower left corner of the output file
- **ymax** (*float*) – y-coordinate of the upper right corner of the output file
- **ofn** (*string*) – output file
- **fmt** – output format

Raises

`IOError`: if the output raster file could not be created
`ValueError`: if the data array has incorrect dimensions

2.9 `getgeoacca` Module

`getgeoacca` is the GeoAcca command line script for analyzing the GEOlocation ACCurAcy of level 1B satellite data of the sensors MERIS, AATSR and ATSR2 and level 3 synthesis products of the sensor PROBA-V. The script operates on a given set of orthorectified reference images covering Ground Control Points (GCPs). For each level 1B file given, `getgeoacca` searches matching GCPs, of which orthorectified image subsets are created. After successfully passing multiple quality tests, the image subset is approved to be matched with the reference. The resulting matching parameters are then stored in the database.

Example uses of getgeoacca:

1. Process the GEOlocation ACCurAcy of all MERIS level 1B input files of the year 2004 stored under `in_data_dir_2004` (see *Configuration files*). The processing involves all GCPs stored in DB. The command line parameters `viewmode`, `matchmethod`, and `log` are initialized with its default settings. The output is stored on disk, no update is forced in database:

```
$ getgeoacca -i MERIS -Y 2004
```

2. Process the GEOlocation AccurAcy of all AATSR (default) level 1B input files with acquisition date 28-01-2004 stored under `in_data_dir_2004` (see *Configuration files*) matching the GCP with ID N06244_W005109. The command line parameters `viewmode`, `matchmethod`, and `log` are initialized with its default settings. The output is stored on disk, no update is forced in database:

```
$ getgeoacca -g N06244_W005109 -Y 2004 -M 1 -D 28
```

3. Process the GEOlocation AccurAcy of all MERIS level 1B input files with acquisition date 02-03-2003 stored under `in_data_dir_2003` (see *Configuration files*) matching the GCP with ID S38572_W068754. For matching the TM_CCORR_NORMED method is used. The command line parameters `viewmode` and `log` are initialized with its default settings. The output is stored on disk, no update is forced in database:

```
$ getgeoacca -g S38572_W068754 -Y 2003 -M 3 -D 2 -i MERIS -m TM_CCORR_NORMED
```

4. Process the GEOlocation AccurAcy of all AATSR level 1B input files with acquisition date 22-06-2004 stored under `in_data_dir_2004` (see *Configuration files*) matching the GCP with ID N37614_E024328. Sensor view mode is set to fward. The command line parameters `matchmethod` and `log` are initialized with its default settings. The output is stored on disk, an update is forced in database:

```
$ getgeoacca -g N37614_E024328 -Y 2004 -M 6 -D 22 -v fward --update
```

5. Process the GEOlocation AccurAcy of the AATSR level 1B input files given in the command line. The processing involves all GCPs stored in DB. The command line parameters `viewmode` and `matchmethod` are initialized with its default settings. Logging is set on DEBUG. The output is stored on disk, an update is forced in database:

```
$ getgeoacca -f /home/xyz/2004/06/22/ATS_TOA_1PUUPA20040622_080048_000065272028_00006_12084_5531.N1
```

With GeoAcca installed, please run `getgeoacca --help` to see the full usage documentation.

Note: `getgeoacca` is capable of controlled termination. Therefore create a file named `STOP` in your `out_data_dir` (see *Configuration files*) directory by running the following command:

```
$ touch STOP
```

Do not forget to delete the file before program restart.

`getgeoacca.`**`check_if_composite`**(*sensor_name*)

Print logging information on elapsed time for single processing steps (if debug mode is active).

Parameters

- **`start_time_gcp`** (*float*) – starting time of GCP processing
- **`setMetaData_time`** (*float*) – elapsed time for metadata extraction
- **`chkSunElev_time`** (*float*) – elapsed time for sun elevation check
- **`ortho_time`** (*float*) – elapsed time for orthorectification
- **`chkNaNVal_time`** (*float*) – elapsed time for NaN value check
- **`chkCldVal_time`** (*float*) – elapsed time for snow/cloud cover check
- **`match_time`** (*float*) – elapsed time for image matching
- **`ql_time`** (*float*) – elapsed time for creation of quicklooks

`getgeoacca.`**`log_process_loop`**(*directory*, *idx_cp*, *len_cp_ids*, *idx_input_file*, *len_input_files*)

Print logging information for termination of GCP processing.

Parameters

- **`directory`** (*string*) – name of directory that has been cleaned up after finishing GCP processing
- **`idx_cp`** (*integer*) – index of GCP for given input file
- **`len_cp_ids`** (*integer*) – number of GCPs covered by given input file
- **`idx_input_file`** (*integer*) – index of input file
- **`len_input_files`** (*integer*) – number of input files

`getgeoacca.`**`log_process_time`**(*start_time_gcp*, *setMetaData_time*, *chkSunElev_time*, *ortho_time*, *chkNaNVal_time*, *chkCldVal_time*, *match_time*, *ql_time*)

Print logging information on elapsed time for single processing steps (if debug mode is active).

Parameters

- **`start_time_gcp`** (*float*) – starting time of GCP processing
- **`setMetaData_time`** (*float*) – elapsed time for metadata extraction
- **`chkSunElev_time`** (*float*) – elapsed time for sun elevation check
- **`ortho_time`** (*float*) – elapsed time for orthorectification
- **`chkNaNVal_time`** (*float*) – elapsed time for NaN value check
- **`chkCldVal_time`** (*float*) – elapsed time for snow/cloud cover check

- **match_time** (*float*) – elapsed time for image matching
- **ql_time** (*float*) – elapsed time for creation of quicklooks

getgeoacca.**main**()

2.10 landsat Module

This module contains a class for the initialisation of a LANDSAT reference raster file.

2.10.1 Classes and Inheritance Structure

2.10.2 Module API

class landsat.**LANDSAT_RASTERFILE**(**args*, ***kwargs*)

Bases: geoacca.controlpoint_raster.CONTROLPOINT_RASTERFILE

A class to work with LANDSAT reference raster files.

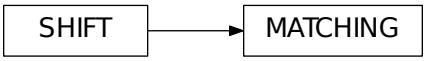
Parameters

- ***args** – variable length argument list.
- ****kwargs** – arbitrary keyword arguments *see*

2.11 matching Module

This module contains a class managing raster matching.

2.11.1 Classes and Inheritance Structure



2.11.2 Module API

class matching.**MATCHING**(*RASTERFILE*, *CONTROLPOINT_RASTERFILE*, *METHOD*, *out_data_dir*, *ref_sensor_name*, *software_ver*, *isComposite*)

Bases: geoacca.db.SHIFT

Raster file matching class.

It is initiated with a RASTERFILE and a CONTROLPOINT_RASTERFILE class and subclasses the SHIFT class, extending it for a matching tool, as well as some plotting tools.

Parameters

- **RASTERFILE** – input raster file object
- **CONTROLPOINT_RASTERFILE** – reference raster file object
- **METHOD** – method object
- **out_data_dir** (*string*) – path to the output directory
- **ref_sensor_name** (*string*) – name of the reference EO sensor
- **software_ver** (*string*) – GeoAcca software version

matching(*fmt*)

Search and find the location of the input raster within the larger reference raster using the cross correlation method. Calculate the shift between the two images and create a correlation coefficients raster file.

Parameters **fmt** (*string*) – output format

Raises

RuntimeError: if neither the python cv2 module nor the python cv module could be loaded

plotCCMatrixSurface()

Create a surface plot of the correlation coefficients.

plotOrthoRasterfileSFT(*cpshp*)

Plot a quicklook of the orthorectified and resampled subset, covering the GCP the raster file class is constructed for, overlaid with the GCP outline and the GCP outline displaced by the calculated shifts.

Parameters **cpshp** (*string*) – path to the GCP shapefile

Raises

IOError: if the GCP shapefile could not be found

2.12 meris Module

This module contains a class representing MERIS raster files.

2.12.1 Classes and Inheritance Structure

2.12.2 Module API

class meris.**MERIS_RASTERFILE**(*in_data_dir*, *input_filename*, *out_data_dir*, *controlpoint_id*, *PGDB*, *track_id*, *psz*, *band_nir*, *view_mode*, *software_ver*, *d_csv*, *irr_band1*)

Bases: geoacca.raster.RASTERFILE

MERIS raster file class.

It is initiated for the given GCP subset and subclasses the RASTERFILE class, extending it for a method to detect the percentage of snow/cloud coverage.

Parameters

- **in_data_dir** (*string*) – path to the input file directory
- **input_filename** (*string*) – name of the level 1B MERIS input file
- **out_data_dir** (*string*) – path to the output directory
- **controlpoint_id** (*string*) – GCP ID as given in the DB. Processing of the input file will be referred to the given GCP.
- **PGDB** – PostGres DB class
- **track_id** (*integer*) – track ID of the input file as given in the DB
- **psz** (*float*) – pixel size for the output files [m]. It should be the same resolution as the reference files.
- **band_nir** (*integer*) – near infrared band number of the input file
- **view_mode** (*string*) – view mode of the input file acquisition
- **software_ver** (*string*) – GeoAcca software version
- **d_csv** (*string*) – name of the file containing the daily sun earth distances
- **irr_band1** (*float*) – spectral solar irradiance at MERIS band 1 [W/m^2]

Raises

ValueError: if the given viewmode is not valid for MERIS

VALID_VIEW_MODES = 'nadir'

chkRasterSCC(*scc_threshold*, *scc_threshold_feat*, *mask*, *fmt*)

Check the MERIS raster snow/cloud coverage.

The method applies on the orthorectified and resampled subset, covering the GCP the MERIS raster class is constructed for. If the snow/cloud coverage exceeds either the scc_threshold or the scc_threshold_feat, which refers to the GCP itself, the to_be_proc flag is set to False. The snow/cloud mask is written to a raster file.

Parameters

- **scc_threshold** (*float*) – snow/cloud cover threshold [%]
- **scc_threshold_feat** (*float*) – snow/cloud cover threshold of the GCP feature [%]
- **mask** (*string*) – path to the extended GCP feature mask covering the input control window
- **fmt** (*string*) – format of the snow/cloud mask raster file

Raises

`IOError`: if the MERIS raster file is missing

`ValueError`: if the extended GCP feature mask does not match the snow/cloud mask window size

chkRastermCP (*lines*, *genauxitems*)

Check if a given GCP is covered by the original MERIS raster file.

Parameters

- **lines** (*integer*) – number of product header lines containing metadata information
- **genauxitems** (*dict*) – general metadata to be loaded

Returns 'True' if the GCP is completely covered by the original MERIS raster file, 'False' otherwise

rad2ref (*rad*, *sun_zenith*)

Convert MERIS band 1 radiance to reflectance values.

Parameters

- **rad** (*array*) – array of MERIS band 1 radiance values
- **sun_zenith** (*array*) – array of MERIS sun zenith angles

Returns MERIS band 1 reflectance values

Return type array

2.13 pregeoacca Module

`pregeoacca` is the GeoAcca command line script for setting up the package basis.

Note: `pregeoacca` has to be run at least once before running `getgeoacca`. Rerun `pregeoacca` if the package basis changes, e.g. you add a new GCP.

The GeoAcca package comes along with multiple CSV files holding information on GCPs, sensors, sensor tracks and matching methods. Basically, `pregeoacca` extracts the relevant information for each GCP listed and processes an appropriate orthorectified reference image subset. The collected information is stored in the database.

Note: At present only LANDSAT is available as reference.

Example uses of pregeoacca:

1. Process the reference subsets for all GCPs listed and add gained information to the database. Neither database update nor reprocessing is forced. The logger output is set to DEBUG level:

```
$ pregeoacca --log=DEBUG
```

2. Process the reference subsets for all GCPs listed and add gained information to the database. Force an update of the reference raster database table:

```
$ pregeoacca --update
```

Attention: The `--update` will delete all or if GCPs are given the respective entries in the database shift table.

3. Process the reference subsets for all GCPs listed and add gained information to the database. Force an update from scratch:

```
$ pregeoacca --update-all
```

Attention: The `--update-all` will delete all entries in the database.

4. Process the reference subsets for the GCPs given in the command line and add gained information to the database. Neither database update nor reprocessing is forced:

```
$ pregeoacca -g N59734_W118694 -g S01862_E137894 -g N35899_E014451
```

5. Give a list of valid GCP IDs:

```
$ pregeoacca --list
```

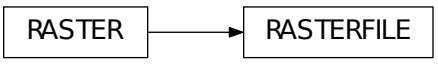
With GeoAcca installed, please run `pregeoacca --help` to see the full usage documentation.

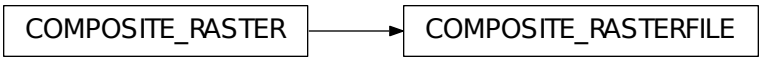
`pregeoacca.main()`

2.14 `raster` Module

This module contains a class representing EO raster files and a class representing composite raster files.

2.14.1 Classes and Inheritance Structure





2.14.2 Module API

class raster.**COMPOSITE_RASTERFILE**(*in_data_dir*, *input_filename*, *out_data_dir*, *controlpoint_id*, *PGDB*, *sensor_id*, *psz*, *sensor*, *band_nir*, *view_mode*, *software_ver*)

Bases: geoacca.db.COMPOSITE_RASTER

EO composite raster file class.

It is initiated for the given GCP subset and subclasses the COMPOSITE_RASTER class, extending it for methods to check the raster NANs and the sun elevation, to load the metadata and import the raster, as well as some plotting tools.

Parameters

- **in_data_dir** (*string*) – path to the input file directory
- **input_filename** (*string*) – name of the input file
- **out_data_dir** (*string*) – path to the output directory
- **controlpoint_id** (*string*) – GCP ID as given in the DB. Processing of input file will be referred to the given GCP
- **PGDB** (*class*) – PostGres DB object
- **psz** (*float*) – pixel size for the output files [m]. It should be the same resolution the reference files are.
- **sensor** (*string*) – name of the EO sensor
- **band_nir** (*integer*) – near infrared band number of the input file
- **view_mode** (*string*) – view mode of the input file acquisition
- **software_ver** (*string*) – GeoAcca software version

Raises

IOError: if the input file or the output directory could not be found

ValueError: if the sensor name is missing

chkRasterNaN()

Check the NaN values of the orthorectified and resampled subset, covering the GCP the composite raster file class is constructed for. The method applies on the NIR band. If any NaN values are found the to_be_proc flag is set to False.

Raises

IOError: if the orthorectified and resampled GCP subset file could not be found

chkSunElev(*sun_elev_threshold*)
Check the composite raster sun elevation at the GCP center point. If it is below the `sun_elev_threshold` the `to_be_proc` flag is set to False.

Parameters **sun_elev_threshold** (*float*) – threshold value for the sun elevation [deg]

importUTMRaster(*resampling*, *nir_data*, *blue_data*)
Import the composite raster input file. The method extracts the NIR and BLUE datasets from the input hdf5 file and writes them to GeoTiff raster files. A spatial subset, covering the GCP, is extracted from the GeoTiff raster files. The raster subsets are orthorectified to UTM projection and resampled.

Parameters

- **resampling** (*string*) – resampling method (options: Nearest, Bilinear or Bicubic)
- **nir_data** (*string*) – hdf5 composite raster dataset with data from NEAR INFRARED channel
- **blue_data** (*string*) – hdf5 composite raster dataset with data from BLUE channel

Raises

`Error`: if an unexpected error appeared
`IOError`: if one of the created raster files could not be opened
`CalledProcessError`: if an external function call failed

isRasterValidType()
Check the acquisition sensor and processing level of the composite raster.

Raises

`ValueError`: if the composite raster is not of the valid acquisition sensor and processing level

loadRasterVar(*genauxitems*, *gcpauxitems*)
Load and set the composite_raster class variables from the metadata.

Parameters

- **genauxitems** (*dict*) – general metadata to be loaded
- **gcpauxitems** (*dict*) – GCP specific metadata to be loaded

plotOrthoRasterfileQKL(*cpshp*)
Plot a quicklook of the orthorectified and resampled subset, covering the GCP the composite raster file class is constructed for, overlaid with the GCP outline.

Parameters **cpshp** (*string*) – path to the GCP shapefile

Raises

`IOError`: if the GCP shapefile could not be found

plotOrthoRasterfileSCM(*cpshp*)
Plot the snow/cloud mask, covering the GCP the composite raster file class is constructed for, overlaid with the GCP outline.

Parameters **cpshp** (*string*) – path to the GCP shapefile

Raises

`IOError`: if the GCP shapefile could not be found

class raster.**RASTERFILE**(*in_data_dir*, *input_filename*, *out_data_dir*, *controlpoint_id*, *PGDB*, *track_id*, *psz*, *sensor*, *band_nir*, *view_mode*, *software_ver*)
Bases: `geoacca.db.RASTER`

EO raster file class.

It is initiated for the given GCP subset and subclasses the RASTER class, extending it for methods to check the raster NANs and the sun elevation, to load the metadata and import the raster, as well as some plotting tools.

Parameters

- **in_data_dir** (*string*) – path to the input file directory
- **input_filename** (*string*) – name of the input file
- **out_data_dir** (*string*) – path to the output directory
- **controlpoint_id** (*string*) – GCP ID as given in the DB. Processing of input file will be referred to the given GCP
- **PGDB** (*class*) – PostGres DB object
- **track_id** (*integer*) – track ID of the input file as given in the DB
- **psz** (*float*) – pixel size for the output files [m]. It should be the same resolution the reference files are.
- **sensor** (*string*) – name of the EO sensor
- **band_nir** (*integer*) – near infrared band number of the input file
- **view_mode** (*string*) – view mode of the input file acquisition
- **software_ver** (*string*) – GeoAcca software version

Raises

`IOError`: if the input file or the output directory could not be found
`ValueError`: if the sensor name is missing

chkRasterNaN()
Check the NaN values of the orthorectified and resampled subset, covering the GCP the raster file class is constructed for. The method applies on the NIR band. If any NaN values are found the `to_be_proc` flag is set to False.

Raises

`IOError`: if the orthorectified and resampled GCP subset file could not be found

chkSunElev(*sun_elev_threshold*)
Check the raster sun elevation at the GCP center point. If it is below the `sun_elev_threshold` the `to_be_proc` flag is set to False.

Parameters **sun_elev_threshold** (*float*) – threshold value for the sun elevation [deg]

importUTMRaster(*resampling*, *orthoXML*, *subsetXML*, *beam_gpt*)
Import the raster input file. The method applies the BEAM subset operator to extract a spatial subset of the raster, covering the given GCP. The raster subset is orthorectified to UTM projection. For resampling the BEAM reproject operator is used.

Parameters

- **resampling** (*string*) – resampling method (options: Nearest, Bilinear or Bicubic)
- **orthoXML** (*string*) – path to the BEAM GPT pixel extraction operator file
- **subsetXML** (*string*) – path to the BEAM GPT subset operator file
- **beam_gpt** (*string*) – path to the BEAM GPT file

Raises

`CalledProcessError`: if an external function call failed

`Error`: if an unexpected error appeared

`IOError`: if the BEAM GPT subset or reproject extraction operator file, or its template files could not be opened

`ValueError`: if the file could not be found

isRasterValidType()
Check the acquisition sensor and processing level of the raster.

Raises

`ValueError`: if the raster is not of the valid acquisition sensor and processing level

loadRasterVar(*lines*, *genauxitems*, *beam_gpt*, *pixexXML*, *gcpauxitems*)
Load and set the raster class variables from the metadata.

Parameters

- **lines** (*integer*) – number of lines in the product header containing metadata
- **genauxitems** (*dict*) – general metadata to be loaded
- **beam_gpt** (*string*) – path to the BEAM GPT file
- **pixexXML** (*string*) – path to the BEAM GPT pixel extraction operator file
- **gcpauxitems** (*dict*) – GCP specific metadata to be loaded

plotOrthoRasterfileQKL(*cpshp*)
Plot a quicklook of the orthorectified and resampled subset, covering the GCP the raster file class is constructed for, overlaid with the GCP outline.

Parameters **cpshp** (*string*) – path to the GCP shapefile

Raises

`IOError`: if the GCP shapefile could not be found

plotOrthoRasterfileSCM(*cpshp*)
Plot the snow/cloud mask, covering the GCP the raster file class is constructed for, overlaid with the GCP outline.

Parameters **cpshp** (*string*) – path to the GCP shapefile

Raises

`IOError`: if the GCP shapefile could not be found

2.15 `utils` Module

This module contains a collection of utility functions.

2.15.1 Module API

`utils.`**clean**(*path*, *negate=None*)
Clean path recursively except of files matching `negate` patterns.

Parameters

- **path** (*string*) – directory to clean

- **negate** (*string*) – pattern to skip

utils.**get_csv_block**(*offset*, *fn*)
Read block from csv file `fn` starting from offset to EOF.

Parameters

- **offset** (*integer*) – position within the file where to start reading
- **fn** (*string*) – csv file

Returns list with text lines

Return type list

Raises

`Error`: if the csv file could not be opened or read

utils.**is_bool**(*value*)
Check if a string value is boolean.

Parameters **value** (*string*) – value to check

Returns True if the value is one of 'true', 'false', 'yes', 'no', 'on' or 'off' (case insensitive), False otherwise

Return type bool

utils.**is_float**(*value*)
Check if a string value can be converted to a float.

Parameters **value** (*string*) – value to check

Returns True if the value can be converted to float, False otherwise

Return type bool

utils.**is_int**(*value*)
Check if a string value can be converted to an integer.

Parameters **value** (*string*) – value to check

Returns True if the value can be converted to integer, False otherwise

Return type bool

utils.**is_long**(*value*)
Check if a string value can be converted to a long integer.

Parameters **value** (*string*) – value to check

Returns True if the value can be converted to long integer, False otherwise

Return type bool

utils.**is_none**(*value*)
Check if a string value is 'None'.

Parameters **value** (*string*) – value to check

Returns True if the value is 'None' (case insensitive), False otherwise

Return type bool

utils.**to_bool**(*value*)
Convert a string value to a boolean value.

Parameters **value** (*string*) – value to convert

Returns True if the value is one of 'true', 'yes' or 'on' (case insensitive), False otherwise

Return type bool

2.16 `utm` Module

This module contains a class managing the coordinate transformation between geographic and UTM coordinates.

2.16.1 Module API

class `utm.`**`TRANSFORMATION`**

Coordinate transformation class for geographic and UTM coordinates.

`get_utm_zone`(*lon*)

Calculate the UTM zone from a given longitude.

Parameters **`lon`** (*float*) – longitude

Returns zone

Return type int

`is_northern`(*lat*)

Determine if a given latitude is Northern (1) or Southern (0).

Parameters **`lat`** (*float*) – latitude

Returns hemisphere flag (1 for the Northern hemisphere, 0 for the Southern hemisphere)

Return type bool

`transform_latlon_to_utm`(*lon*, *lat*, *zone=None*, *zero_one=None*)

Transform geographic coordinates to UTM coordinates.

Parameters

- **`lon`** (*float*) – longitude
- **`lat`** (*float*) – latitude
- **`zone`** (*integer*) – UTM zone
- **`zero_one`** (*integer*) – hemisphere flag (1 for the Northern hemisphere, 0 for the Southern hemisphere)

Returns UTM x-coordinate, UTM y-coordinate, UTM zone

Return type tuple

`transform_utm_to_latlon`(*easting*, *northing*, *zone*, *zero_one*)

Transform UTM coordinates to geographic coordinates.

Parameters

- **`easting`** (*float*) – UTM x-coordinate
- **`northing`** (*float*) – UTM y-coordinate
- **`zone`** (*integer*) – UTM zone
- **`zero_one`** (*integer*) – hemisphere flag (1 for the Northern hemisphere, 0 for the Southern hemisphere)

Returns longitude, latitude, altitude

Return type tuple