CHAPTER

ONE

INTRODUCTION TO SIGLIB

Sphinx.
↳ makefile.

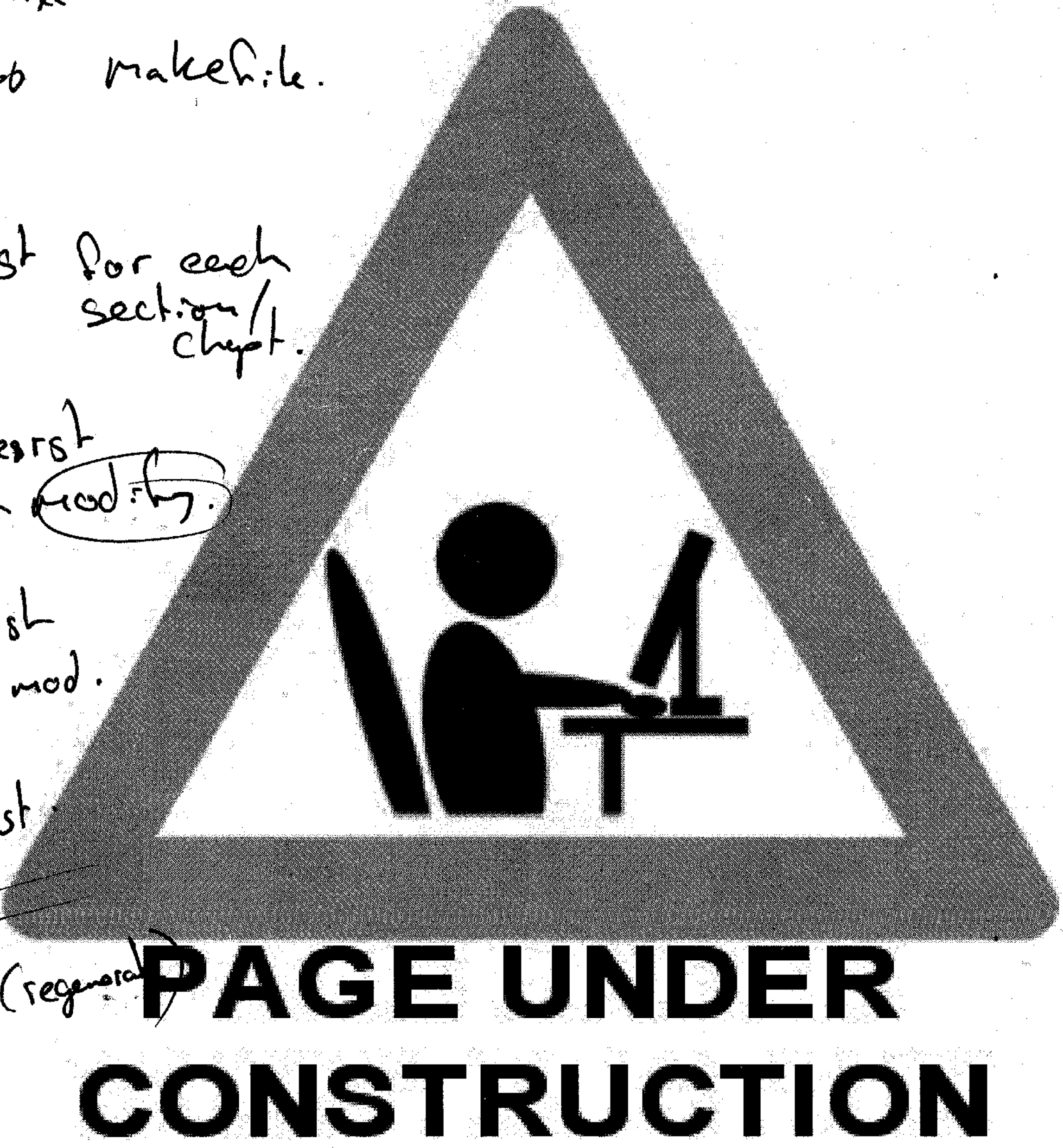
/Source
rst for each
section/
chapt.

Codesrst
↳ modify.

index.rst
can mod.

edit
intro.rst

compile. (regenerate)



↳ make latexpdf

look under build.

CHAPTER

TWO

PROJECT SUMMARY

CHAPTER

THREE

USING SIGLIB

PAGE UNDER CONSTRUCTION

CHAPTER

FOUR

SIGLIB API

4.1 Create Image

createImg.py

Created on Mon Oct 7 20:27:19 2013 **@author:** Sougal Bouh Ali

`createImgV2.`**`createHeader`**`()`

Creates the log file header

`createImgV2.`**`createImg`**`(`*zipfile*`)`

Creates an image file (.tif) based on the image type (*amp, sigma, noise, theta*) requested, and subsequently does image manipulation (*project, crop, mask, stretch*).

Parameters *zipfile* : raw data to be processed into image

`createImgV2.`**`discoverImgs`**`(`*path, pattern*`)`

Locates satelite image raw data files (zipfiles) using a *pattern* in *path* search method, and then calls,createImg() to process the data into image.

Parameters *path* : directory tree to scan

pattern : file pattern to discover

`createImgV2.`**`discoverImgs_query`**`(`*query, dbName*`)`

Locates satelite image raw data files (zipfiles) using a *query* in *db* search method, and then calls createImg() to process the data into image.

Parameters *db* : database

query : information retrieval query

4.2 Database

database.py

Created on Tue Feb 12 23:12:13 2013 **@author:** Cindy Lopes

This module creates an instance of class Database and connects to a database to create, update and query tables.

class `database.`**`Database`**`(`*dbname, user, port, host*`)`

This is the Database class for each database connection.

`alterTimestamp``(`*shpTable*`)`

Takes a shape file and converts the gps_time from character type to timestamp time.

•

Parameters *shpTable* :

`annaQuery` (*attributeList, shpTable*)

Parameters *attributeList* :

shpTable :

`beaconShapefilesToTables` (*dirName*)
Takes a directory containing beacon shape files and converts them to tables and inserts them into the database appending *beacon_* before the name

Parameters *dirName* :

`bothArchiveandMetadata` ()
Finds all the results in both the archive and tblmetadata.

`checkTblArchiveOverLapsTblMetadata` (*filename*)
Check if a file name from tblArchive is in the overlap table.

Parameters *filename* :

Returns *dictionary* :

`copyfiles` (*copylist, wrkdir*)
Copies files from cisarchive. If file could not be found, check that the drive mapping is correct (above).

Parameters *copylist* : a list of images + inst

wrkdir : working directory

`copylistExport` (*copylist, fname*)
Saves the copylist as a text file named fname.txt in the current dir.

Parameters *copylist* : a list of images + inst

fname : filename to write copylist to

`copylistImport` (*fname*)
Reads the copylist text file named fname.txt in the current dir.

Parameters *fname* : filename to read copylist from

Returns *new_copylist* : a list of images + inst

`createFromArchive` (*archdir*)
Goes to CIS Archive and create the tblArchive in postGIS database Assumes dbase postgis exists and that tblArchive does as well - overwrites!

Can be extended to import from other archives (Derek's ASF archive)

Parameters *archdir* : archive directory

`createTblMetadata` ()
Creates a metadata table, namely *tblmetadata*. It overwrites if *tblmetadata* already exist.

`customizedQuery` (*attributeList, roi, spatialrel, proj*)
Customizable query that takes an list of attributes to search for, a roi, a spatialrel, and a proj and returns a dictionary with all the requested attributes for the results that matched the query

Parameters *attributeList* :

roi :

spatialrel :

proj :

Returns *copylist* :

instimg :

dbProj (*proj*)
Relates *proj* the name (ie. proj.wkt) to *proj* the number (i.e. srid #).

Parameters *proj* : projection name

Returns *srid* : spatial reference id number of that projection

exportToCSV (*dictionary_list, outputName, orderedKeys=None*)
Given a dictionary of results from the database and a filename puts all the results into a csv with the filename outputName

Parameters *dictionary_list* :

outputName :

orderedKeys :

imgData2db (*imgData, xSpacing, ySpacing, bandName, inst, dimgname, granule*)
Here are the data in an array... upload to database need the imgData, the imgType, the bandName, the inst, dimgname and granule

will compute the count, mean, std, min, max for non-zero elements and send them to db as well

Parameters *imgData* :

xSpacing :

ySpacing :

bandName :

inst : instance id (i.e. a 5-digit string)

dimgname : Derek's image name

granule : granule name

instimg2db (*roi, spatialrel, instimg, copylist, mode='refresh'*)
There can be several relational tables that contain the name of an image and the feature that it relates to: For example: a table that shows what images intersect with general areas or a table that lists images that contain ROI polygons...

This function runs in create mode or refresh mode Create - Drops and re-creates the table

Refresh - Adds new data (leaves the old stuff intact)

Parameters *roi* : region of interest file

spatialrel : spatial relationship (i.e. ST_Contains or ST_Intersect)

instimg : a list of only images of that instance id

copylist : a list of images + inst

mode : create or refresh mode

Returns *copylist* : a list of images + inst

meta2db (*metaDict*)
Uploads image metadata to the database as discovered by the meta module. *meta* is a dictionary - no need to upload all the fields (some are not included in the table structure)

Note that granule and dimgname are unique - as a precaution - a first query deletes records that would otherwise be duplicated This assumes that they should be overwritten!

Parameters *metaDict* : dictionnary containing the metadata

nameTable (*roi*, *spatialrel*)
Automatically gives the table a name.

Parameters *roi* : region of interest

spatialrel : spatial relationship (i.e. ST_Contains or ST_Intersect)

Returns *name* : name of the table

numpy2sql (*numpyArray*, *dims*)
Converts a 1- or 2-D numpy array to an sql friendly array Do not use with a string array!

Parameters *numpyArray* : numpy array to convert

dims : dimension (1 or 2)

Returns *array_sql* : an sql friendly array

qryCropZone (*granule*, *roi*, *spatialrel*, *proj*, *inst*)
Writes a query to fetch the bounding box of the area that the inst polygon and image in question intersect. returns a crop ullr tupple pair in the projection given

Parameters *granule* : granule name

roi : region of interest file

spatialrel : spatial relationship (i.e. ST_Contains or ST_Intersect)

proj : projection name

inst : instance id (i.e. a 5-digit string)

Returns *ullr* : upper left, lower right tupple pair in the projection given

qryFromFile (*fname*, *path*)
Runs a query in the current databse by opening a file - adds the path and .sql extension - reading contents to a string and running the query

Note: do not use % in the query b/c it interfers with the pyformat protocol used by psycopg2

Parameters *fname* : file name

path : full path to fname

qryGetInstances (*granule*, *roi*, *spatialrel*, *proj*)
Writes a query to fetch the instance names that are associated spatially in the relational table.

Parameters *granule* : granule name

roi : region of interest file

spatialrel : spatial relationhip (i.e. ST_Contains or ST_Intersect)

proj : projection name

Returns *instances* : instances id (unique for entire project, i.e. 5-digit string)

qryMaskZone (*granule*, *roi*, *spatialrel*, *proj*, *inst*)
Writes a query to fetch the gml polygon of the area that the inst polygon and image in question intersect. returns gml text but also saves a file... mask.gml in the current dir

Parameters *granule* : granule name

roi : region of interest file

spatialrel : spatial relationship (i.e. ST_Contains or ST_Intersect) *proj* : projection name

inst : instance id (i.e. a 5-digit string)

Returns *polytext* : gml text

`qrySelectFromArchive` (*roi*, *spatialrel*, *proj*)
Given a table name (with polygons, from/todates), determine the scenes that cover the area (spatialrel = contains or intersects) from start (str that looks like iso date) to end (same format).

Eventually include criteria: subtype - a single satellite name: ALOS_AR, RADAR_AR, RSAT2_AR (or ANY)

beam - a beam mode

comes back with - a list of images+inst - the bounding box

Parameters *roi* : region of interest file

spatialrel : spatial relationship (i.e. ST_Contains or ST_Intersect)

proj : projection name

Returns *copylist* : a list of images + inst

instimg : a list of only the images of that instance id

`qrySelectFromLocal` (*roi*, *spatialrel*, *proj*)
Determines the scenes that cover the area (spatialrel = contains or intersects) from start (str that looks like iso date) to end (same format).

Eventually include criteria: subtype - a single satellite name: ALOS_AR, RADAR_AR, RSAT2_AR (or ANY)

beam - a beam mode

comes back with - a list of images+inst - the bounding box

Parameters *roi* : region of interest

spatialrel : spatial relationship (i.e. ST_Contains or ST_Intersect)

proj : projection

Returns *copylist* : a list of images + inst

instimg : a list of only the images of that instance id

`sql2numpy` (*sqlArray*, *dtype='float32'*)
Comming from SQL queries, arrays are stored as a list (or list of lists) Defaults to float32

Parameters *sqlArray* : an sql friendly array

dtype : default type (float32)

Returns *list* : list containing the arrays

`updateFromArchive` (*archdir*)
Goes to CIS Archive and updates the tblArchive in postGIS database Assumes dbase postgis exists and that tblArchive does as well - overwrites!

Can be extended to import from other archives (Derek's ASF archive)

Parameters *archdir* : archive directory

`updateROI` (*inFile*, *path*, *proj*)
Goes to a shapefile named inFile and updates the postGIS database Assumes dbase postgis exists and that outTable does as well - this overwrites!

Parameters *inFile* : basename of a shapefile

path : full path to inFile

proj : projection name

Required name, inst, obj, type, fromdate, todate(optional)

fromdate - a valid time_start (ie it is here, when was it here?)

todate - a valid time_end (ie it is here, when was it here?)

inst - an instance id (unique for entire project) - Nominally a 5-digit string

Optional refimg - a reference image name (ie how do you know it was here)

type - an ice type (ie ice island, ice shelf, mlsi, fyi, myi, epishelf, open water)

subtype - an ice subtype (ie ice island could be iced firn, basement; open water could be calm, windy)

comment - a comment field

name - a name (Target7, Ayles)

obj - an object id tag (to go with name but very systematic: 2342, and if it splits 2342_11 & 2342_12)

`update_NTAI_FLUX_ROI` (*inFile, path, proj*)

Goes to a shapefile named inFile and updates the postGIS database Assumes dbase postgis exists and that outTable does as well - this overwrites!

Parameters *inFile* : basename of a shapefile

path : full path to inFile

proj : projection name

Required name, inst, obj, type, fromdate, todate(optional)

fromdate - a valid time_start (ie it is here, when was it here?)

todate - a valid time_end (ie it is here, when was it here?)

inst - an instance id (unique for entire project) - Nominally a 5-digit string

Optional refimg - a reference image name (ie how do you know it was here)

type - an ice type (ie ice island, ice shelf, mlsi, fyi, myi, epishelf, open water)

subtype - an ice subtype (ie ice island could be iced firn, basement; open water could be calm, windy)

comment - a comment field

name - a name (Target7, Ayles)

obj - an object id tag (to go with name but very systematic: 2342, and if it splits 2342_11 & 2342_12)

4.3 Discover Metadata

Now merged with createImg.

discoverMeta.py

Created on Tue Nov 12 11:05:44 2013 **@author:** Sougal Bouh Ali

`discoverMetaV2.addFiletotblMetadata` (*fname, imgname, sattype, granule, unzipdir, db*)

Adds the image file metadata to tblmetadata table in the specified database.

Parameters *fname* : image filename (i.e. R1_980705_114117.img OR product.xml)

imgname : image name (i.e. R1_980705_114117)

sattype : satelite platform

granule : granule name

unzipdir : unzip directory

db : database connection

`discoverMetaV2.`**`createHeader`**`()`

Creates the log file header

`discoverMetaV2.`**`discoverMeta`**`(`*path, pattern, db*`)`

Locates satelite image raw data files (zipfiles) using a *pattern* in *path* search method, and then calls addFiletotblMetadata() to add the image file metadata to tblmetadata table in the specified database.

Parameters *path* : directory tree to scan

pattern : file pattern to discover

db : database connection

4.4 Image Processing

imgProcess.py

Created on ??? Jul ? ??:??:?? 2009 **@author:** Derek Mueller

This module creates an instance of class Img and opens a file to return a gdal dataset to be processed into an amplitude, calibrated, noise or theta (incidence angle) image, etc. This image can be subsequently projected, cropped, masked, stretched, etc.

Modified on ??? Feb ? ??:??:?? 2012 **@reason:** Repackaged for r2convert **@author:** Derek Mueller

class `imgProcess.`**`Img`**`(`*fname, path, meta, imgType, imgFormat, zipname*`)`

This is the Img class for each image. RSAT2, RSAT1 (ASF and CDPF)

`applyStretch``(`*stats, procedure='std', sd=3, bitDepth=8, sep='tog'*`)`

Given stats... will stretch a multiband image to the dataType based on procedure (either sd for standard deviation, with +ve int in keyword sd, or min-max, also a linear stretch).

!!A nodata value of 0 is used in all cases!!

!!For now, dataType is byte and that's it!!

Note: gdal_translate -scale does not honour nodata values See: http://trac.osgeo.org/gdal/ticket/3085

Have to run this one under the imgWrite code. The raster bands must be integer, float? or byte and int data assumed to be only positive. Won't work very well for dB scaled data (obviously) it is important that noData is set to 0 and is meaningful.

sep = separate: applies individual stretches to each band (=better visualization/contrast)

tog = together: applies the same stretch to all bands (looks for the band with the greatest dynamic range)

(=more correct)

For further ideas see: http://en.wikipedia.org/wiki/Histogram_equalization

`cleanFiles` (*levels=['crop']*)
Removes files that have been written.

Input a list of items to delete: raw, nil, proj,crop, final

`cropBig` (*llur, subset*)
Here we have a way to crop that will expand the area of an image. However, this uses gdalwarp - and resampling/offsetting could skew result - by a fraction of a pixel obviously, but still..

Parameters *llur* : list/tuple of tuples in projected units

`cropImg` (*ullr, subset*)
Given the cropping coordinates, this function tries to crop in a straight-forward way. If this cannot be accomplished (likely because the corner coordinates of an image are not known to a sufficient precision) then gdalwarp (cropBig) will do the job.

Parameters *ullr* : upper left and lower right coordinates *subset* :

`cropSmall` (*urll, subset*)
This is a better way to crop b/c no potential for warping... However, this will only work if the region falls completely within the image.

Parameters *ullr* : list/yuple of tuples in projected units

`decomp` (*format='imgFormat'*)
Takes an input ds of a fully polarimetric image and writes an image of the data using a decomposition - could be 1) pauli TODO: 2) freeman 3) cloude

Differs from imgWrite b/c it ingests all bands at once...

`fnameGenerate` (*projout=None, subset=None, band=None*)
Decide on some parameters based on self.imgType we want...

`getAmp` (*datachunk*)
return the amplitude, given the amplitude... but make room for the nodata value by clipping the highest value...

`getBandData` (*band*)
opens an img file and reads in data from a given band assume that the dataset is small enough to fit into memory all at once

`getImgStats` (*inst*)
Opens a raster and calculates (approx) the stats returns an array - 1 row per band cols: band, dynamicRange, dataType, nodata value, min, max, mean, std

`getMag` (*datachunk*)
return the magnitude of the complex number

`getNoise` (*n_lines*)
For making an image with the noise floor as data

`getPhase` (*datachunk*)
Return the phase (in radians) of the data (must be complex/SLC)

`getSigma` (*datachunk, n_lines*)
Calibrate data to Sigma Nought values (linear scale)

`getTheta` (*n_lines*)
For making an image with the incidence angle as data

`imgWrite` (*format='imgFormat', stretchVals=None*)
Takes an input ds and writes an image.

self.imgType could be 1) amp, 2) sigma, 3) noise, 4) theta all bands are output (amp, sigma)

Also used to scale an integer img to byte with stretch, if stretchVals are included

`makePyramids` ()
Uses gdaladdo to make pyramids aux style

`maskImg` (*mask*, *vectdir*, *side*, *imgType*)
Masks all bands with gdal_rasterize using the 'layer'

side = 'inside' burns 0 inside the vector, 'outside' burns outside the vector

Note: make sure that the vector shapefile is in the same proj as img (Use reprojSHP from ingestutil)

Parameters *mask* : *vectdir* : *side* : *imgType* :

`openDataset` (*fname*, *path=''*)
Opens a dataset with gdal

Parameters *fname* : filename

`projectImg` (*projout*, *projdir*, *format=None*, *resample='bilinear'*)
Looks for a file, already created and projects it to a vrt file.

Parameters *projout* : projection base name *projdir* : path to the projection

NOTE THE PIXEL SIZE IS NOT PROSCRIBED! (it will be the smallest possible)

`reduceImg` (*xfactor*, *yfactor*)
Uses gdal to reduce the image by a given factor (i.e, factor 2 is 50% smaller or half the # of pixels) and saves as a temporary file and then overwrites.

Parameters *xfactor* : float *yfactor* : float

`stretchLinear` (*datachunk*, *scaleRange*, *dynRange*, *minVal*, *offset=0*)
Simple linear rescale: where min (max) can be the actual min/max or mean+/- n*std or any other cutoff

Note: make sure min/max don't exceed the natural limits of dataType takes a numpy array datachunk the range to scale to, the range to scale from, the minVal to start from and an offset required for some stretches (see applyStretch keyword sep/tog)

`vrt2RealImg` (*subset=None*)
When it is time to convert a vrt to a tiff (or even img, etc) use this

4.5 Metadata

meta.py

Created on ??? Jul ? ??:??:?? 2009 **@author:** Derek Mueller

This module creates an instance of class Meta and contains functions to query raw data files for metadata which is standardized and packaged for later use, output to file, upload to database, etc.

Note: Functions to read CEOS files and handle splines were inspired by or copied from IAPro.

class `metaV2.`**`Meta`** (*granule*, *imgname*, *path*, *sattype*)
This is the metadata class for each image RSAT2, RSAT1 (ASF and CDPF)

`clean_metaASF` (*result*)
Takes meta data from origmeta and checks it for completeness, coerces data types splits values, if required and puts it all into a standard format

NOT TESTED!!

clean_metaCDPF (*result*)
Takes meta data from origmeta and checks it for completeness, coerces data types splits values, if required and puts it all into a standard format

createMetaDict (*location*)
Creates a dictionary of all the metadata fields for an image this can be written to file or sent to database

Note that the long boring metadata fields are not included

getCEOSmetafile ()
Get the filenames for metadata

getCornerPoints ()
Given a set of geopts, calculate the corner coords to the nearest 1/2 pixel. Assumes that the corners are among the GCPs (not randomly placed)

getDimgname ()
Create a filename that conforms to my own standard naming convention: yyyymmdd_HHmmss_sat_beam_pol...

getMoreGCPs (*n_gcps*)
If you have a CDPF RSat1 image, gdal only has 15 GCPs Perhaps you want more? If so, use this function. It will grab all the GCPs available (3 on each line) and subselect n_gcps of these to return.

The GCPs will not necessarily be on the 'bottom corners' since the gcps will be spaced evenly to get n_gcps (or more if not divisible by 3) If you want corners the only way to guarentee this is to set n_gcps = 6

getRS2metadata ()
Open a Radarsat2 file and get all the required metadata

get_ceos_metadata (**file_names*)
Take file names as input and return a dictionary of metadata file_names is a list or strings or a string (with one filename)

getgdalmeta ()
Open file with gdal and get metadata

Ret gdal_meta

saveMetaFile (*dir=''*)
Makes a text file with the metadata ToDO: MAKE THIS AN XML FILE

metaV2.**byte2int** (*byte*)
Reads a byte and converts to integer

metaV2.**date2doy** (*date, string=False, float=False*)
Give a python datetime and get an integer or string doy fractional doy returned if float=True

metaV2.**datetime2iso** (*datetimeobj*)
Return iso string from a python datetime

metaV2.**doy2date** (*year, doy*)
Give a float, integer or string and get a datetime

metaV2.**getEarthRadius** (*ellip_maj, ellip_min, plat_lat*)
Calculates the earth radius at the latitude of the satellite from the ellipsoid params

metaV2.**getGroundRange** (*slantRange, radius, sat_alt*)
Finds the ground range from nadir which corresponds to a given slant range must be an slc image, must have calculated the slantRange first

metaV2.**getSlantRange** (*gsr, pixelSpacing, n_cols, order_Rg, groundRangeOrigin=0.0*)
gsr = ground to slant range coefficients -a list of 6 floats pixelSpacing - the img. res., n_cols - how many pixels in range ground range orig - for RSat2 (seems to be zero always)

Valid for SLC as well as SGF

metaV2.**getThetaPixel** (*RS, r, h*)
Calc the incidence angle at a given pixel

metaV2.**getThetaVector** (*n_cols, slantRange, radius, sat_alt*)
Make a vector of incidence angles in range direction

metaV2.**get_data_block** (*fp, offset, length*)
gets a block of data from file

metaV2.**readdate** (*date, sattype*)
Takes a rsat2 formated date 2009-05-31T14:43:17.184550Z and converts it to python datetime

4.6 Utility

utility.py

Created on Tue Feb 12 20:04:11 2013 **@author:** Cindy Lopes

Modified on Sat Nov 23 14:49:18 2013 **@reason:** Added writeIssueFile and compareIssueFiles **@author:** Sougal Bouh Ali

Modified on Sat Nov 30 15:37:22 2013 **@reason:** Redesigned getFilname, getZipRoot and unZip **@author:** Sougal Bouh Ali

utilityV2.**compareIssueFiles** (*file1, file2*)
Compares 2 clean issue files generated by writeIssueFile() and generates a separate file containing the list of matched & unmatched files in 2 files.

Parameters *file1* : name of the first text file with extension to be compared (i.e. textfile.txt)

file2 : name of the second text file with extension to be compared (i.e. textfile.txt)

utilityV2.**deltree** (*dirname*)
Delete all the files and sub-directories in a certain path

Parameters *dirname* :

utilityV2.**getFilename** (*zipname, unzipdir*)
Given the name of a zipfile, return the name of the image, the file name, and the corresponding sensor/platform (satellite).

Parameters *zipname* : *unzipdir* :

Returns *fname* : *imgname* : *sattype* :

utilityV2.**getZipRoot** (*zip_file, tmpDir*)
Looks into a zipfile and determines if the contents will unzip into their own subdirectory or not, then return the unzipdir and zipname.

Parameters *zip_file* : *tmpDir* :

Returns *unzipdir* : *zipname* :

utilityV2.**getdBScale** (*power*)
Convert a SAR backscatter value from the power scale to the dB scale

Note: power must be a scalar or an array of scalars,negative powers will throw back NaN.

Parameters *power* :

Returns *sig* :

`utilityV2.`**`reprojSHP`**(*in_shp*, *vectdir*, *proj*, *projdir*)

Opens a shapefile, saves it as a new shapefile in the same directory that is reprojected to the projection wkt provided.

Note: this could be expanded to get polyline data from polygon data for masking lines (not areas) ogr2ogr -nlt MULTILINESTRING

Parameters *in_shp* : *vectdir* : *proj* : *projdir* :

Returns *out_shp* : name of the proper shapefile

`utilityV2.`**`unZip`**(*zip_file*, *unzipdir*, *ext='all'*)

Unzips the zip_file to unzipdir with python's zipfile module.

"ext" is a keyword that defaults to all files, but can be set to just extract a leader file L or xml for example.

Parameters *zip_file* : *unzipdir* :

`utilityV2.`**`wkt2shp`**(*shpname*, *vectdir*, *proj*, *projdir*, *wkt*)

Takes a polygon defined by well-known-text and a projection name and outputs a shapefile into the current directory

Parameters *shpname* :

vectdir :

proj :

projdir :

wkt :

`utilityV2.`**`writeIssueFile`**(*fname*, *delimiter*)

Generates a clean list of zipfiles, when given an Issue File written by scripts.

Parameters *fname* : name of the text file with extension to be cleaned(i.e. textfile.txt)

delimiter : separator used to split zipfile from unwanted errors (i.e. use most common " / " before zipfile)

A01 original for CI2D3

```
                                    List of databases
       Name        |  Owner    | Encoding |   Collate   |    Ctype    |   Access pr
ivileges
-------------------+-----------+----------+-------------+-------------+------------
----------
 ~~CDPF~~              | sali      | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 ~~CI2D3~~             | sali      | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 ~~CIS~~               | sali      | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 CIS_archive       | sali      | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 ~~PDC~~               | sali      | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 RS2               | sali      | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 Ron               | rsaper    | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 acrawford_db      | sali      | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 ci2d3_alpha       | rsaper    | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 postgres          | postgres  | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 r1                | sali      | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 r2                | sali      | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 sali_test_db      | sali      | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 template0         | postgres  | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres
          +
                   |           |          |             |             | postgres=CT
c/postgres
 template1         | postgres  | UTF8     | en_US.UTF-8 | en_US.UTF-8 | postgres=CT
c/postgres+
                   |           |          |             |             | =c/postgres
 template_postgis  | postgres  | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 ~~test_db~~           | sali      | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 wirlcis           | sali      | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 wirlsar           | sali      | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
(19 rows)
```

RS2 — beacon + flex

ci2d3_alpha — prototype

r1, r2 — discover latest meta.

template_postgis — has SRIDs.

\tank\ica\data\sar\r1
r2

CIS Archive -

tblmetadata

this is the master database

SRIDs → 96718 - Lec.

[Scan]

path : → } not parallel Specify Directory to scan.

query : → pass a qry too!

file : → 1 = parallel. → use head. sh

Φ = normal.

no cleaning.

7937
8012

python 2.7

$\boxed{\text{paramiko}}$ → Not working
- security features

→ $\boxed{\text{SSH}}$ → errors

→ shp2pgsql.py.

ogr2ogr → $\boxed{\text{workaround}}$

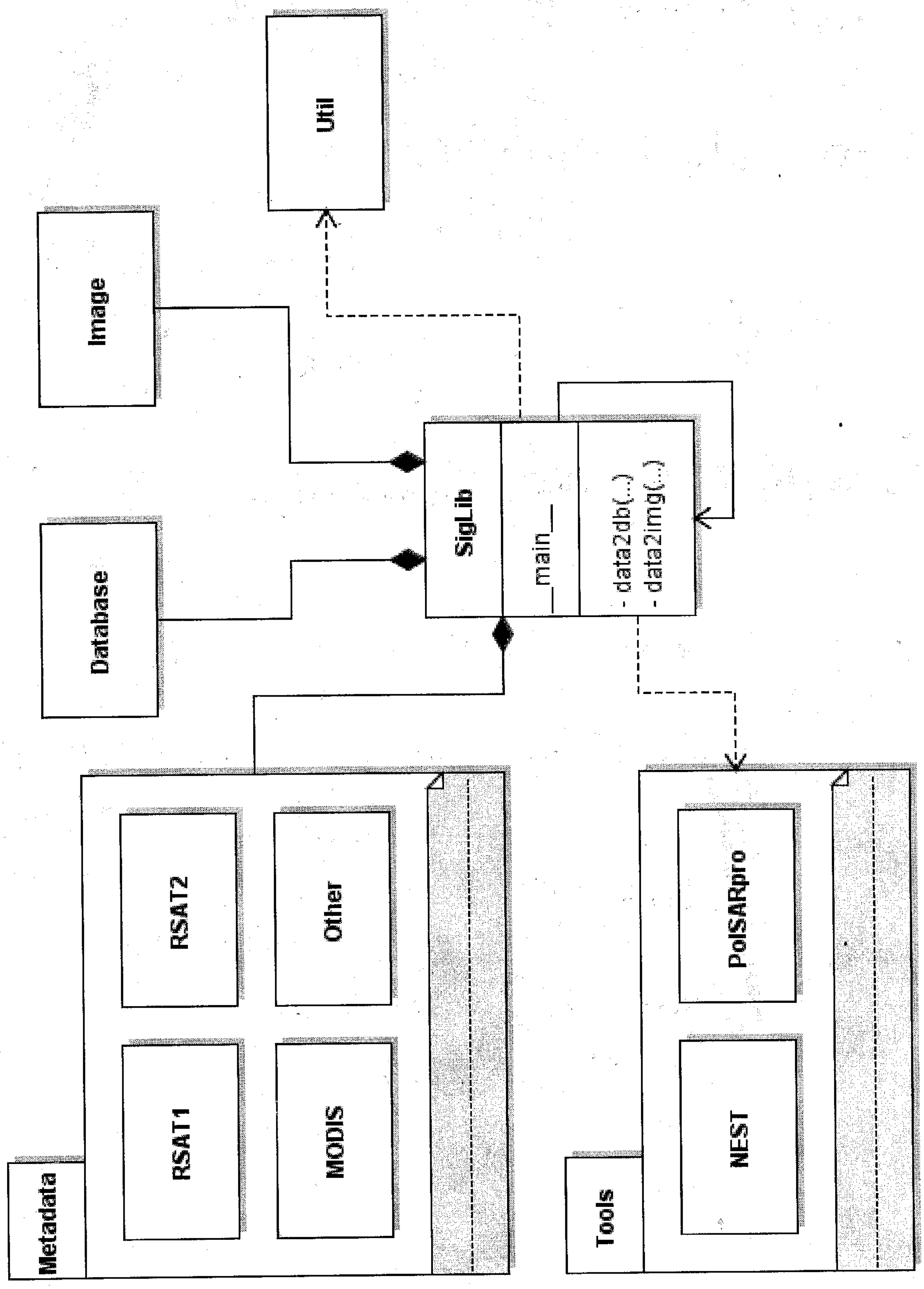
/tank/HOME/sali/CIAD3 /shp2pgsql.py.

C.

DiscoverMeta fails → r1 r2 → /tank/HOME/sali/ Siglib/logs

look here
or
Run query-scan. $\boxed{\text{not parallel}}$
to get output.

Create logs.


Util
Image
Database
SigLib
__main__
- data2db(...)
- data2img(...)
Metadata
RSAT1
RSAT2
MODIS
Other
Tools
NEST
PolSARpro