Notes for maintaining the lds_topo_reblocker code

Overview

Simple version. The reblocker reads submitted shapefiles and pushes them to a PostgreSQL instance on an LI geoserver. The database, taking each layer in turn, cross compares the features in the uploaded layer determining if any two touch on a mapsheet boundary. If they do a new feature, replacing the two (or more) old features, is created. The result is returned to the client as a shapefile.

Client (Python)

The main script the user interacts with is the LayerReader.py. Using command line switches the user selects a shapefile to reblock and applies options that dictate the extent of the processing, indicate ID naming or release a processed layer as complete.

LayerReader.py
The LayerReader is the main command line interface module.

main(): Main command processing function. Arguments are parsed using getopt and set processing flow and assigns options/flags. Processing is redirected in this function to utility functions (crop, release) or reblock.

convertImpl(): This is the main implementation function that does layer reblocking. It also reads args to determine the extent of reblocking to undertake: Upload files only, download processed files or just process stalled files. Depending on action selection this function instantiates a SFDS (ShapeFile Data Source) and/or a PGDS (PostgreSQL DataSource) instance and passes them to a LayerReader class.

LayerReader(): The LayerReader class provides a transfer method calling read/write on Source and Destination instances. It is also used to trigger the reblocking function call on the database.

SFDS(_DS): This is the shapefile processing class. It provides read and write functionality. The write functionality wraps a writeLayer function and a writeFeature function for when GDAL fails.

PGDS(_DS): This is the postgres processing class. It provides read and write functionality. The write functionality wraps a writeLayer function and a writeFeature function for when GDAL fails. Connection parameters are hardcodes but credentials will come from a credentials file.

GdalErrorHandler(): This class is designed trap GDAL errors that get sent to stderr/stdout rather than returned as exceptions. It relies on the @captureGdalError decorator being applied to any potential error functions.

ReblockerUI.py
The UI is a Tkinter based user interface. Its not regularly maintained.

ReblockerServer.py
The Reblock Server is a WiP module that is intended to be run in a client/server environment as an API/Webservice.

Config.py
This is a wrapper module for the Python ConfigParser

Constant.py
The Constant module reads a config file and sets its attributes from this file

Server (PostgreSQL)

The functions comprising this part of the application are contained in the nbrfnc.sql file. These functions perform a number of jobs including joining cut features and managing feature IDs.

reblockall(): This is the main entry point function for the reblocker. It queries the available data tables and processes reblock candidates.

rbl_init(): This function sets up the tables and permissions.

reblocksplitter(): The reblocksplitter function performs reblocking duties between the regions defined in the cropregions layer (NB crop regions are user defined regions across which there are no continuous features. They are used to divide the processing task into more manageable groupings. The default/provided cropregions splits NZ between the North and South islands). The reblocksplitter also divides the processing task between the mapsheet boundaries running North-South and those running East-West. Again, this is to reduce processing load.

reblockfaster(): The reblockfaster function is the workhorse function and performs a number of operations. generate_rbl is the first function call and generates a list of reblock candidates. generate_aggrbl then aggregates all coincident joins. Then starting with the un-joined features the joined features are added to the result table after being assigned a new FID. New FID assignment is performed using the ufid_generator function.

generate_rbl(): This function creates the reblocklist by initially filtering and then performing a match to see if feature cut ends match.
This query performs the following successive operations.
1. Divide features between NI and SI
2. Round/snap edge points so they match boundry lines
3. Determine whether edge points touch mapsheet boundaries
4. Filter only touching points
5. Cross match all layer features to see if they touch each other
6. Output 1-to-1 list of joins to perform
e.g. $A \cap B \rightarrow$ [A,B], $A \bar{\cap} C \rightarrow$ [], $B \cap C \rightarrow$ [B,C]

generate_aggrbl(): The generate_aggrbl function is a recursive function that uses a custom aggregate to accumulate unique FIDs into arrays of linked features. The linked features are filtered down to a unique list for output.
e.g. [A,B]+[B,C] $\rightarrow$ [A,B,C]

ufid_generator(): This function checks the rbl_associations table to see if the combination of component FIDs has been seen before. If it has the matching FID is returned and if not a new one is generated and stored in the rbl_associations table.