Image Calibration Pipeline Test Plan

1. Create a branch from the master code line called ‘calibrationpipeline’.
   1. Push new code only to the branch:   
      > git checkout -b calibrationpipeline   
      > git status # to verify that you are the branch  
      > git commit -a. # create the commit  
      > git push -u origin calibrationpipeline # push to the branch
   2. After the code has passed tests, it can be merged back into master:  
      > git merge origin/calibrationpipeline
2. Create a set of test tables in SQLite that mirror the production tables.
   1. fits\_test.db for the name of the test database
   2. same table names as for fits.db: fits and fits\_by\_target
   3. copy the catalog\_by\_target table from fits.db to fits\_test.db so that we don’t have to switch database connections to use it. The code in catalog.py can be updated to include creating the catalog\_by\_target table from the database.messier file.
   4. Create starting content by restoring to the fits\_test database from the fits database. Keep the backup of fits for resetting the test environment later, as needed.
3. All programs must have a TEST mode
   1. This can be an argument so that it is easy to run in the TEST mode or it can be a TEST variable set to TRUE or FALSE in the code so that it’s NOT easy to run the test mode, forcing the tester to be very intentional about test runs.
4. Create a test folder
   1. /home/nas/EagleTEST
   2. The test directory structure will mirror the production directory structure exactly so that when run in TEST mode the files are created and moved in the test folder and in production they are created and moved in the production folder. Only the top folder will be different.
   3. Populate the top folder with the existing files from /home/nas/Eagle as a starting point.
5. Create a reset program
   1. Remove all the files in the reduced folder structure
   2. Removes all data from the test database and restores with starting content
6. Create a regression test program
   1. Identify a set of starting data and a set of expected outputs from running the program
   2. Report on any missing outputs
   3. Report on any outputs that are incorrect
   4. Report on any outputs that are not expected
   5. Existing outputs that should not change need to be checked
   6. New outputs that should be present need to be checked
7. Establish a code push and test process
   1. When code is pushed to the branch, the regression test should be run.
   2. All issues identified by the regression test should be documented and fixed.
   3. Once fixes are in place, the test environment should be reset and the regression test run again.
   4. Repeat this process until all reported issues by the regression test are resolved.