NHSC/PACS Web Tutorials
Running PACS Photometer pipelines

PACS-401

Level 2.5 Map-Making with MADmap

for HIPE 9.0 user release Version

NASA Herschel Science Center

Introduction

NHSC PACS Web Tutorial

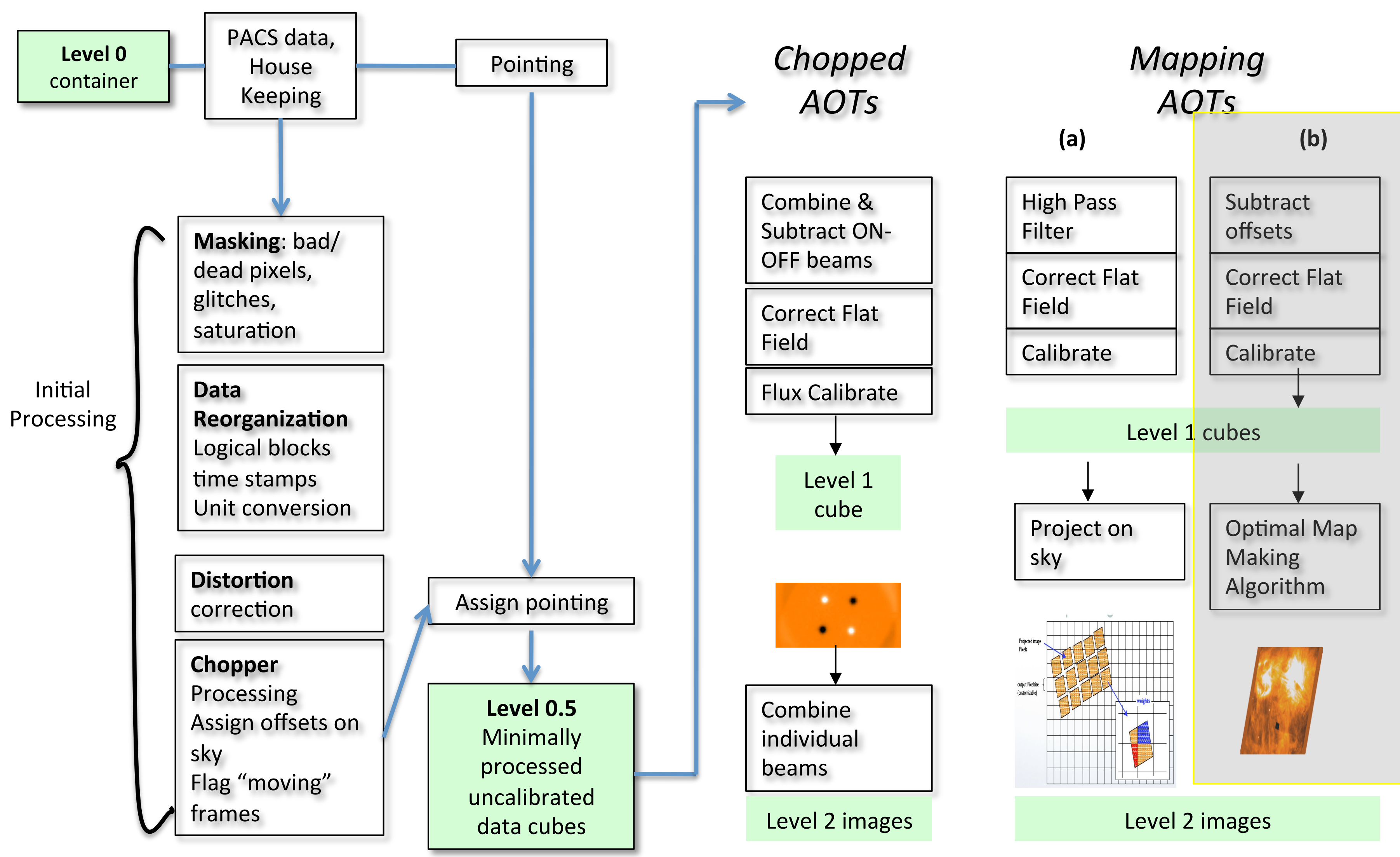
- This tutorial provides a walk-through from Level 0 to 2.5 processing using the MADmap branch of the PACS photometer pipeline.
- The tutorial follows the ***ipipe*** script:

  L25_scanMapMadMap.py

  NOTE: This tutorials now starts at PACS Level 1 product

- At the end of the tutorial, you will have created a PACS map from individual bolometer readouts using the optimal map-mapping algorithm MADmap.

Pipeline section covered here



http://nhsc.ipac.caltech.edu/helpdesk/index.php

Documentation Reference

- PACS data reduction guide, chapter 9
  - PACS scanmap reduction using MADmap
- **PACS-101**: Introduction to PACS tutorials
- **PACS-103**: Accessing & Storing PACS data
- **PACS-104**: Using iPipe scripts
- **PACS-201**: Level 0 to 1 processing of PACS photometer data

Pre-requisites:

1. You should have completed the following tutorials:
   - ***PACS-101***: *How to use these tutorials.*
   - ***PACS-104***: *How to access and use ipipe data reduction scripts.*
   - ***PACS-201***: *Level 0 to Level 1 processing*
2. HIPE 9.0 user-release
3. The example dataset for RCW 120. The data should be placed in a local pool with the OBSID as the pool name. **You will need the full path name to this directory**.

Processing Overview

Step 1

Check script and software pre-requisites

Step 2

Loading ipipe script L25_scanMapMadMap.py

Step 3

Pre-amble and script parameters

Step 4

Identify the data to process

Step 5

Making sense of the main processing loop

Processing (Cont.)

Step 6

MADmap pre-processing (post Level 1)

Step 7

Remove correlated signal drifts

Step 8

Create MADmap ToD product

Step 9

Create the "naive" and optimal maps

Step 10

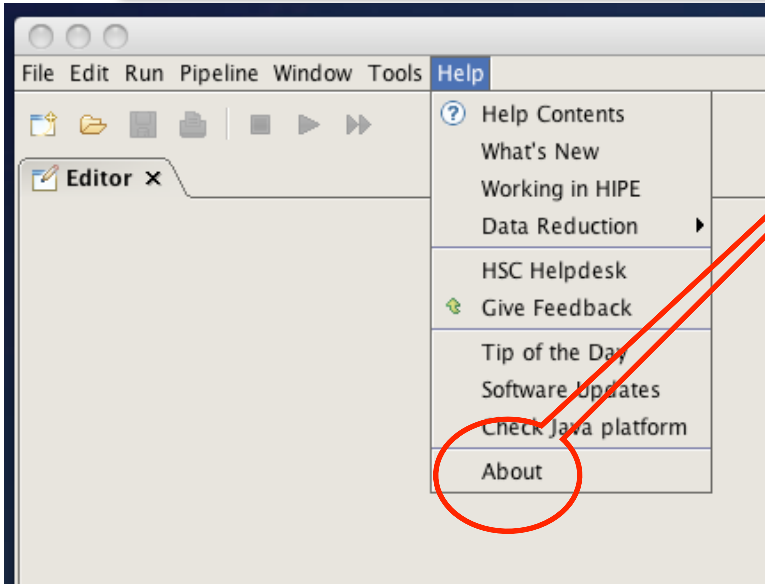
Point-Source artifact correction

Step 1

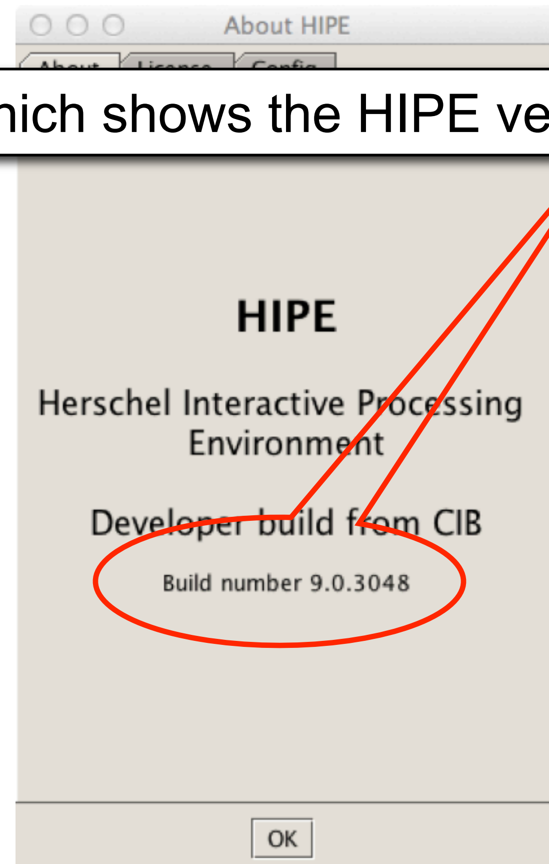
Check your software version

Check # 1: HIPE 9.0 build ....

From the "Help" menu, select "About"



Which shows the HIPE version



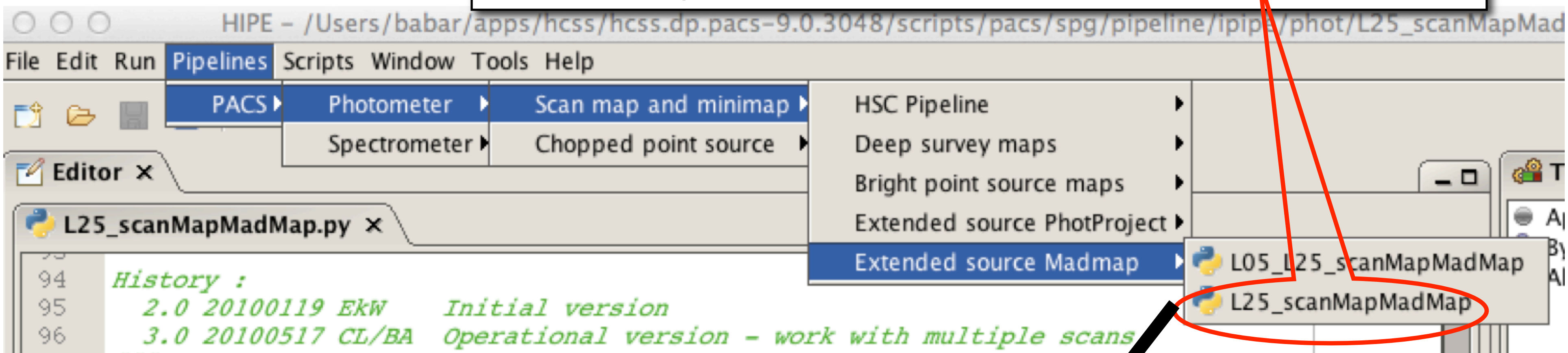
If your Build number does not start with 9.0, stop and upgrade.

Step 2

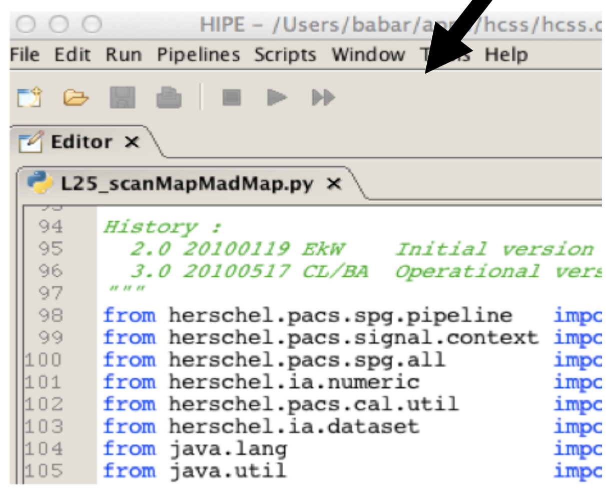
Load ipipe script "L25_scanMapMadMap.py"

Step 2: Load ipipe script

From the pipeline menu, make the selections as shown to get to “L25_scanMapMadMap”



If you successfully loaded the script, it’ll appear as a folder tab under the Editor window.



Step 2: **Warnings**

- You should always save the template ipipe script under a new name before making changes to prevent accidentally overwriting and destroying the original templates.

- See **PACS-104** for details.

Step 3

Pre-amble and script parameters

The preamble

Highlight and execute the block of import and definition statements with the single green arrow.

Editor ×

L25_scanMapMadMap.py ×

```
97   """
98   from herschel.pacs.spg.pipeline      import *
99   from herschel.pacs.signal.context    import *
100  from herschel.pacs.spg.all           import *
101  from herschel.ia.numeric             import *
102  from herschel.pacs.cal.util          import *
103  from herschel.ia.dataset             import LongParameter
104  from java.lang                       import Boolean
105  from java.util                       import ArrayList
106
107  from herschel.pacs.spg.phot.madmapUtils import *
108  #
```

The initial lines (97 in the case of example shown) are comments and may be skipped.

The import statement define java classes to be used later.

Preamble (cont.)

The next set of lines define the PACS bolometer channel to work on.

```
#       > camera = 'blue'
#     the try/except here
try:
  camera
except NameError:
  camera = 'blue'
#
```

Replace these lines with this one, which selects the red (long-wavelength) PACS channel.

```
#     the try/exce

camera = 'red'

#
```

Execute this statement by positioning the marker next to the statement and clicking on the single green arrow.

See **PACS-102** for reminders

Parameter Summary

Parameter	Description	Recommend Value
verbose	Print processing step details.	Boolean(1)
doplot	For signal drift correction, show the plot of best-fit model.	Boolean(1)
outdir	Where to save the output from 'doplot' and final maps and reduced data.	A valid directory on your system.
globalDriftModel	Option for drift correction. See step 6.	1
modelOrder	Polynomial order. See step 6.	2
ignoreFirst	Mask and remove this many readouts from the start of the observation	2000 for large datasets, 100-500 small datasets
envSize	Envelope size for IInd level deglitcher.	10
nsigma	Glitch rejection criteria	20
scale	Scale of the output map pixels compared to PACS native pixels (1=use native pixel scaling).	1
doPGLScorrection	Whether or not to do MADmap point-source artifact correction. See Step 11.	True, if bright point sources are present. False, otherwise.
PGLS_iterations	The number of iterations in the point-source artifact corrector algorithm. See Step 11.	

Point Source Artifact Correction

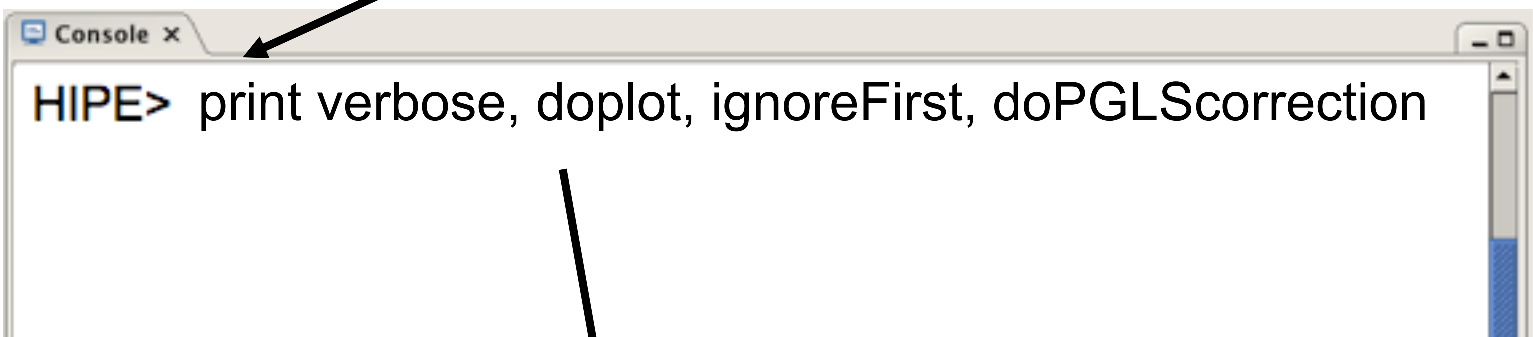
```
scale              = 1.
# Point source artifacts correcti
try:
  doPGLScorrection
except NameError:
  doPGLScorrection = False
try:
  PGLS_iterations
except NameError:
  PGLS_iterations   = 5
#
```

Set to ‘False’ for this demo.

Highlight & execute this stanza.

Check # 2: Preamble okay?

Execute the following line either by typing in console or editing your script.



The output should contain the values you set in Step 3.

Step 4

Identify the data (scan and cross-scan) for processing.

Step 4

Execute the following lines in your console or edit them in the script.

The scan and cross-scan OBSID pairs are identified in a jython vector. The values are long integers.

```
obsids  = [1342185553,1342185554]
pooldir = "/Users/babar/projects/NHSC/workshops/2011-12-Internal/"
obsList = [ getObservation(obsids[0], poolLocation=pooldir), \
            getObservation(obsids[1], poolLocation=pooldir) ]
```

pooldir is the full path name to where the data pools are stored. See pre-requisites for this tutorial and PACS-102

The nested commands in the last line(s) read and populate a vector of observation contexts. This identifies your data to HIPE.

Check # 3: Data is properly loaded

Highlight and execute the following lines in your script.

```
#
# *********************************************************************
#         Processing PER OBSID
# *********************************************************************
#
ra                        = [ obsList[0].meta["ra"].double ]
dec                       = [ obsList[0].meta["dec"].double ]
print "ra :", ra
print "dec:", dec
#
first = 1
```

If there was an error, you (likely) already got a notification. This step ensures that there are no problems with the pools themselves.
The output should be the programmed values for the right ascension and declination of the object in the first OBSID.

Step 5

MADmap pre-processing

The pre-processing loop:

A. For each observation in your scan and cross-scan pair, the following processing steps are executed:
   - Step 5: Level 0 to 1 processing.
   - Step 6: Post level 1, MADmap pre-processing.
   - Step 7: Remove correlated signal drift

B. After the processing steps, on the first pass through the loop a super frames structure is created.

C. On the next pass the cross-scan data is appended to the super frames structure

The Processing Loop

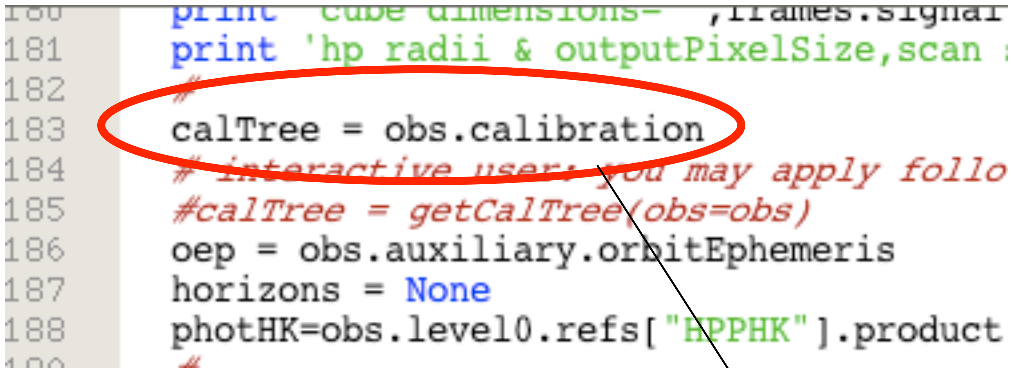
Is this the first time through?

A. Step 5

```
#
first       = 1
firstInvntt = True
#
for obs in obsList:
  obsidStr = str(obs.meta["obsid"].value)
  #L0.5 - L1
  pp = obs.auxiliary.pointing
  if camera=='blue':
     frames=obs.level0.refs["HPPAVGB"].product.refs[0].product
     hpfradius1 = 100  # Radius used for highpass filtering to get mask
     hpfradius2 = 100  # Radius used for highpass filtering in the 2nd pass
     outputPixelSize = 3.2
     if frames.meta.containsKey("mapScanLegLength") :
        if frames.meta["mapScanLegLength"].value <= 5.0 :
           hpfradius2 = 15
           outputPixelSize = 1.0
  elif camera=='red':
     frames=obs.level0.refs["HPPAVGR"].product.refs[0].product
     hpfradius1 = 100  # Radius used for highpass filtering to get mask
     hpfradius2 = 100  # Radius used for highpass filtering in the 2nd path
     outputPixelSize=6.4
     if frames.meta.containsKey("mapScanLegLength") :
        if frames.meta["mapScanLegLength"].value <= 5.0 :
           hpfradius2 = 25
           outputPixelSize=2.0
  if obs.cusMode=="SpirePacsParallel":
     speed = frames.meta["mapScanRate"].value
     if speed=="slow":
        lowscanspeed  = 15.
        highscanspeed = 25.
     elif speed == "fast":
        lowscanspeed  = 54.
        highscanspeed = 66.
  elif obs.cusMode=="PacsPhoto":
     speed = frames.meta["mapScanSpeed"].value
     if speed=="medium":
        lowscanspeed  = 15.
        highscanspeed = 25.
     elif speed =="high":
        lowscanspeed  = 54.
        highscanspeed = 66.
  print 'cube dimensions= ',frames.signal.dimensions
  print 'hp radii & outputPixelSize,scan speed ',hpfradius1,hpfradius2,outputPix
  #
  calTree = obs.calibration
  # interactive user: you may apply following e.g. to get the most recent calibr
  #calTree = getCalTree(obs=obs)
```

```
invnttVersion =
if ( invnttVers
   print " !! T
# Check INVNTT
if (firstInvntt
   invnttVersio
   obsidStrFirst       = obsidStr
   firstInvntt         = False
else :
   if (invnttVersion != invnttVersionFirst) :
      print " !! ObservationCointext OBSID pairs contain different
      print " OBSID : " + obsidStr + " / " + str(invnttVersion) +
#
#
nframes = frames.dimensions[2]
print "Number of frames = ", nframes
#
# Determine the band used in the moment just the middle one of an
band = frames.status["BAND"].data[frames.dimensions[2] /2]
```

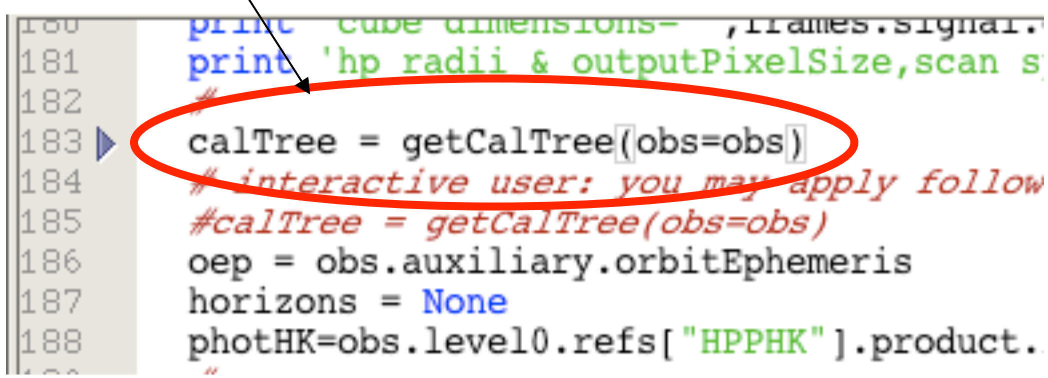
Use the proper calibration tree.



The recommended calibration tree comes with your install of HIPE.

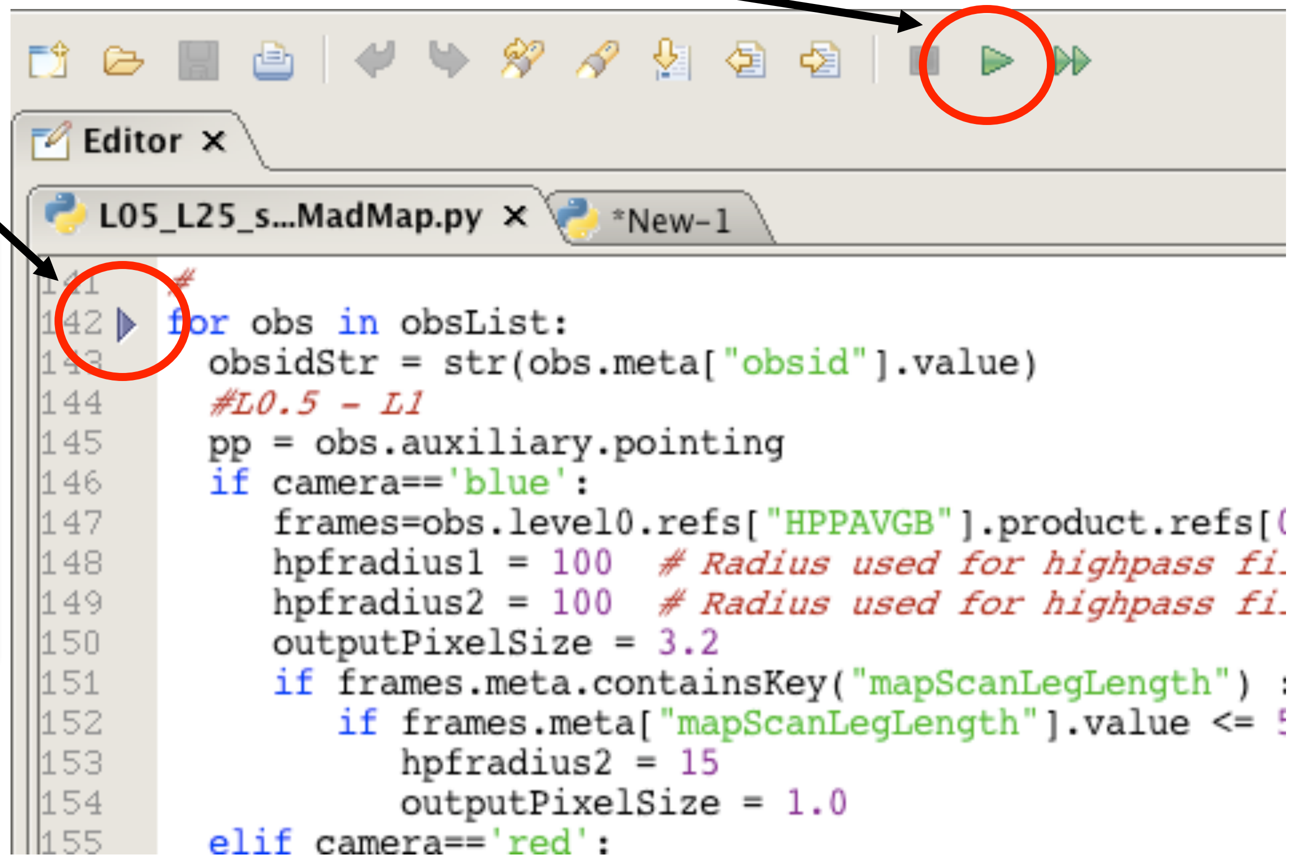
Find, then change the call to populate the calibration file structure as shown.

This should be near line 156.



Executing the loop

Position then execute with the single green arrow.



You are encouraged to read about the steps performed in the loop in the next few slides before executing the loop.

Check # 4: Position cubes exist

NHSC PACS Web Tutorial

Issue this command in the console window

Console

```
HIPE>
HIPE> print frames
{description="Frames", meta=[type, creator, creationDate, description, instrument,
modelName, startDate, endDate, formatVersion, detRow, detCol, camName, relTimeOffset, Apid,
subType, compVersion, algoNumber, algorithm, compNumber, compMode, dxid,
qflag_pacs_phot_red_FailedSPUBuffer, qflag_pacs_phot_blue_FailedSPUBuffer, RemovedSetTime,
blue, chopAvoidFrom, chopAvoidTo, dec, dither, fluxExtBlu, fluxExtRed, fluxPntBlu,
fluxPntRed, lineStep, m, mapRasterAngleRef, mapRasterConstrFrom, mapRasterConstrTo,
mapScanAngle, mapScanAngleRef, mapScanConstrFrom, mapScanConstrTo, mapScanCrossScan,
mapScanHomCoverage, mapScanLegLength, mapScanNumLegs, mapScanSpeed, mapScanSquare, n,
naifid, obsOverhead, pointStep, ra, repFactor, source, fileName, obsid, obsType, obsCount,
aorLabel, aot, cusMode, equinox, missionConfig, naifId, object, obsMode, odNumber, origin,
raDeSys, telescope, level, isInterlaced], datasets=[Signal, Status, Mask, BlockTable,
History, Ra, Dec, Noise], history=Available}
HIPE>
```

Look for "dataset"s Ra and Dec

Your output will likely look slightly different but you should NOT get an error message and the important "ra" and "dec" datasets exist in your "frames" object.

Step 6

Post level 1 MADmap pre-processing

MADmap preprocessing

Executing the loop will automatically execute these steps for both OBSIDs.

Cleanup unstable frames at the start of observation.

```
#
frames = photMadMapIgnoreFirst(frames, ignoreFirst)
#
frames = photAssignRaDec(frames, calTree=calTree)
#
frames = photOffsetCorr(frames)
#
```

Assign pointing to each and every pixel and readout

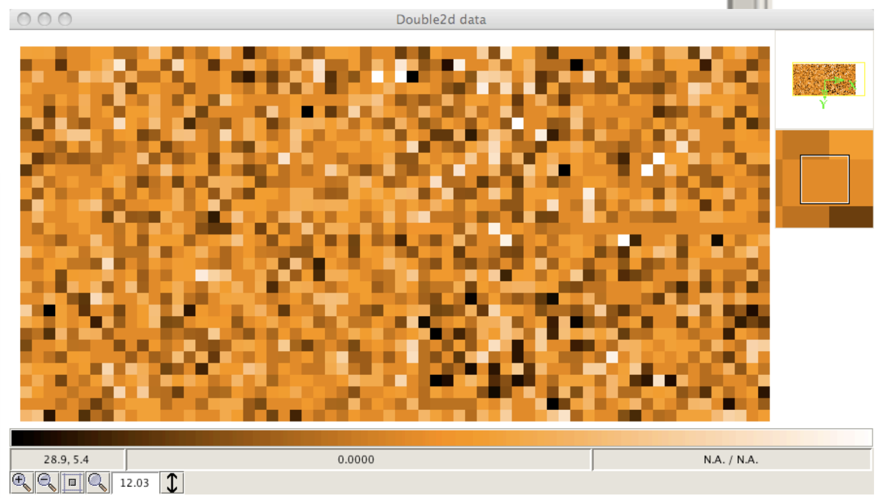
Apply pixel-to-pixel electronic offset correction.

Check # 5: Offsets are removed

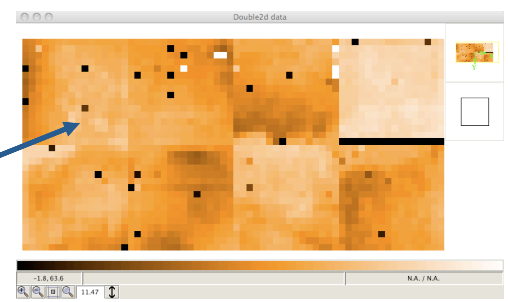
Type this command

Console

```
HIPE> Display( frames.signal[:,:,100] )
```



Expected Output:
With proper offset removal, the image shows a relatively constant signal. Note: extremely bright sources may also be observed on a single image.



An example of improper or no pixel-to-pixel offset correction.

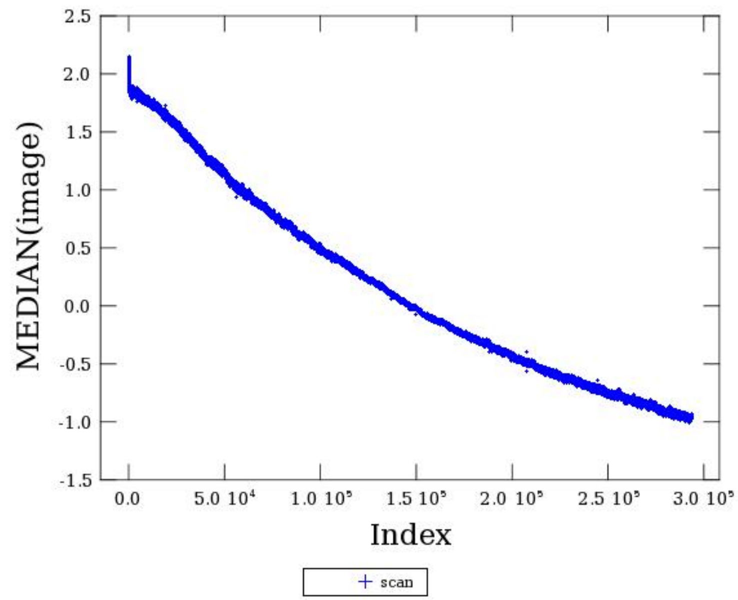
Step 7

Remove Correlated Signal Drifts

Background

NHSC PACS Web Tutorial

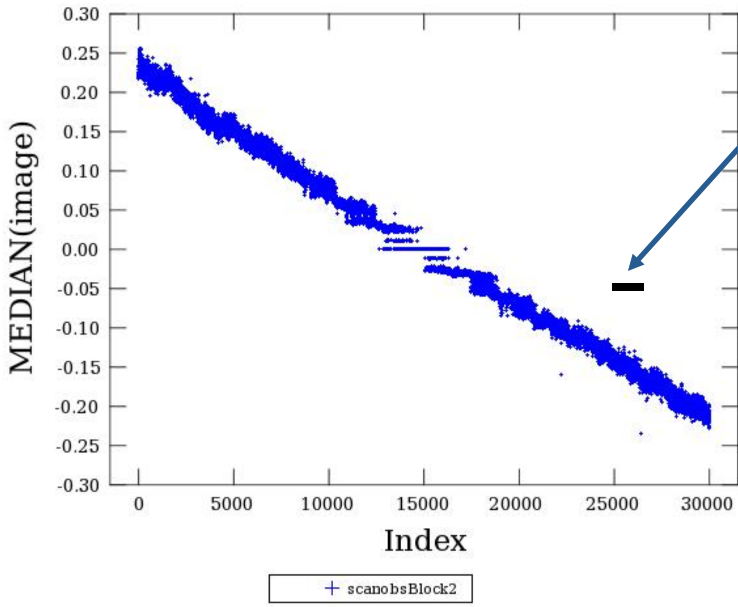
PACS' correlated signal drift.



This Figure illustrates what is meant by both correlated and drift for PACS signal. The Figure shows the median value of the bolometer array as a function of readout index. The monotonic signal showing a decay in intensity is commonly observed in PACS' image cubes, and is thought to be related to focal plan temperature drifts.

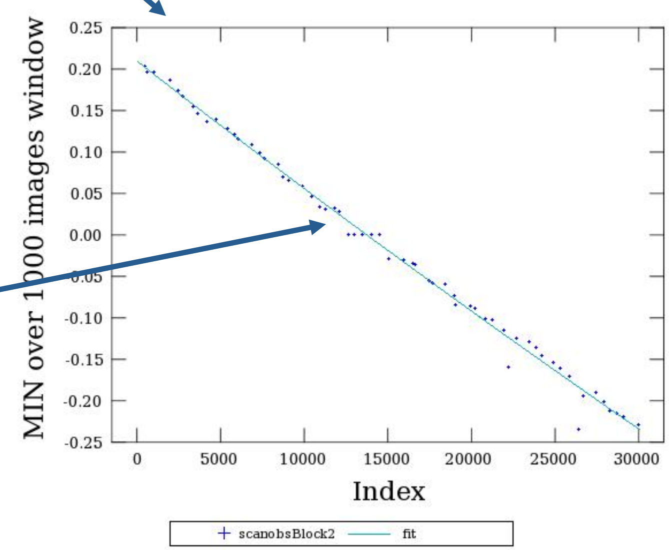
Background

Mitigating the signal drift.



Divide the array median in bins of N readouts. N is typically 1000 readouts.

Take the minimum value in each N readout bins.

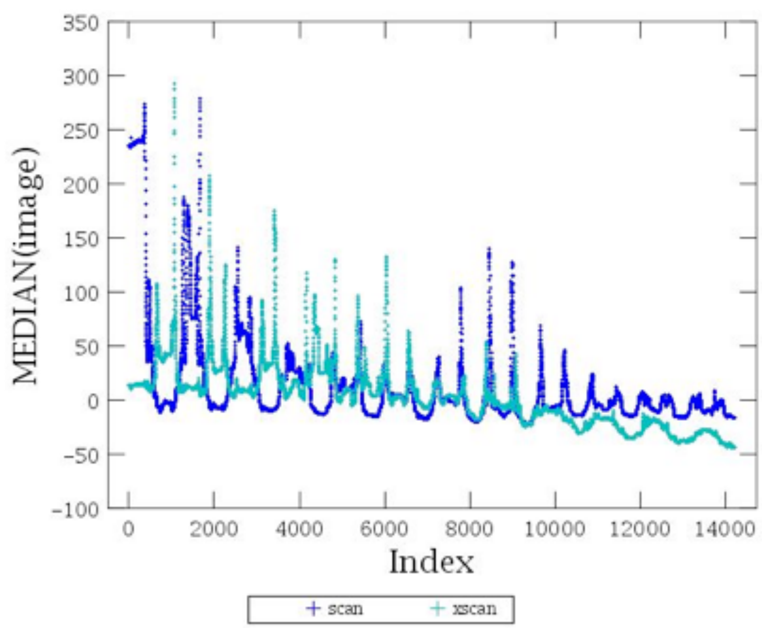


Fit the resulting curve with a polynomial.

http://nhsc.ipac.caltech.edu/helpdesk/index.php

Background

If the sources are weak (i.e. do not produce significant signal in a single image) it may be sufficient to fit the median values directly. However, for strong sources, the minimum approach becomes necessary.



An observation with strong sources. The minimum values still manage to trace the overall drift fairly accurately.

Documentation

- PACS data reduction guide, chapter 9

Step 7

The drift correction is automatically applied to the data when the main loop is exectued.

The ***photGlobalDriftCorrection*** *module allows several options for fitting and removing the drift.*
See: PDRG chapter 7
Or type
Print photGlobalDriftCorrection
In the HIPE window.

The most important parameters

model=1

This is the default and uses the minimum of the bins as discussed above.

polyOrder=3

Sets the order of the fitted polynomial

binSize=1000

Sets the size of the bin from which the minimum value is determined.

Step 8

Create MADmap ToD product

Step 8

NHSC PACS Web Tutorial

Execute the single line

```
#
#
# Make the Time Ordered Data array
tod = makeTodArray(joinframes, calTree=calTree, scale=scale)
# To scale output pixle size or to rotate the final map:
# tod = makeTodArray(joinframes,calTree=calTree,scale=myscale,crota.
#
```

The ToD stands for Time-ordered-Data and is the internal format used by MADmap.
In fact, makeTodArray will create a binary file in your temporary area that has the rearranged PACS signal in the proper format.

- *The scale parameter selects the size of the output sky grid relative to the nominal PACS pixel sizes. E.g. scale=0.5 for PACS blue channel will result in final pixel sampling of 1.6"/pixel.*

Step 9

Create Naive and Optimal Maps

Step 9

Select and execute this block of commands

```
maxRelError    = 1.e-5
maxIterations = 500

naivemap = runMadMap(tod,calTree,maxRelError,maxIterations,True)
madmap=runMadMap(tod,calTree,maxRelError,maxIterations,False)
```

Documentation Reference:

PDRG Chapter 9

Both the naive and optimal maps are created with the same call. The last parameter is set to 'True' for naive map and 'False' for optimal map. MADmap uses maximum likelihood and conjugate gradient solvers to find the optimal solution. The parameters maxRelError and maxIterations control both the convergence tolerance and the number of iterations in finding the optimal solution. See the above reference for details.

Check # 8: Output map

Issue this command in the console window

Console

```
HIPE> print madmap
```

Expected output:
The output from madmap or naivemap making is a simpleImage product class with several datasets.

```
HIPE> print madmap
{description="MadMap", meta=[type, creator, creationDate, description, instrument, modelName, startDate,
endDate, formatVersion, wavelength], datasets=[image, error, exposure, History], history=Available}
HIPE>
```

Step 10

Correct the final map for point source artifacts

Step 10

Execute the block of lines

```
#
# Do point source artifacts correction if requested
if doPGLScorrection:
   print "do point source artifacts correction"
   correctedmap = photCorrMadmapArtifacts(joinframes, tod, madmap, PGLS_iterations)
   #Display(correctedmap)
#
```

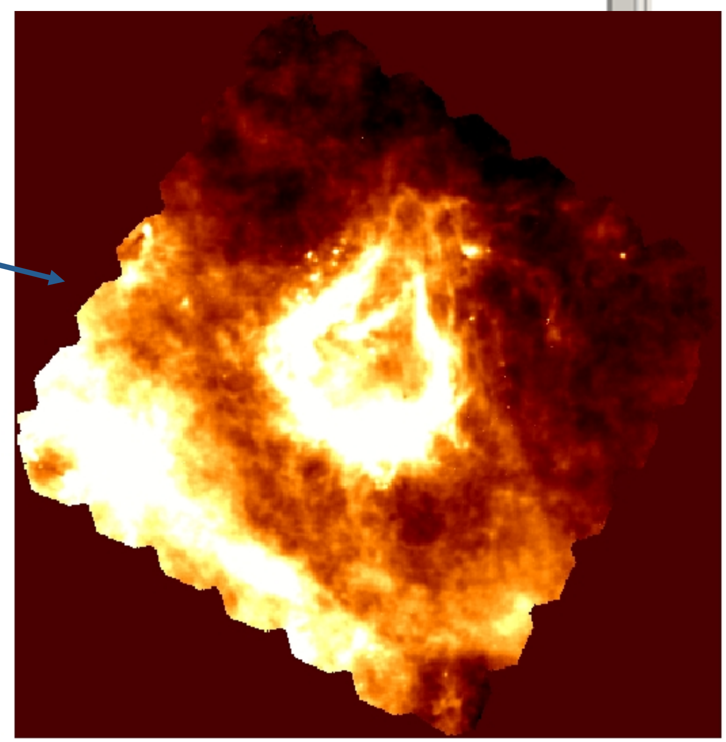
The doPGLScorrection flag is set at the beginning of the script. If set, the correctedmap variable will contain the artifact free map.
See PDRG Section 9.5 for details.
The number of interations for the PGLS algorithm are set in the PGLS_iterations variable (at the start of the script).

Check # 9: Display the final map

Issue this command in the console window

```
HIPE> Display(madmap)
```

Expected output:
A mosiac of all images in your PACS data cube.



See PACS-202 for how to manipulate Display to show different planes in the simpleImage.