DUNES: Data reduction

This document serves as a summary of the data reduction process and parameter values adopted for the *Herschel* (Pilbratt et al., 2010) archive's PACS and SPIRE images made by the Open Time Key Programme DUNES (DUst around NEarby Stars) [1]. We refer the user to the DUNES survey paper, "DUst around NEarby Stars: The survey observational results" (Eiroa et al., 2013) wherein much of the information presented here is illustrated and many of the caveats explained.

Jonathan P. Marshall: jonathan.marshall@uam.es

Observations

PACS (Poglitsch et al., 2010) scan map observations of all 133 DUNES targets in 130 fields (due to the doubling-up on several binary sources) were taken with the 100/160 channel combination. Some 70/160 observations were taken for a sub-set of these stars based on their expected 100 µm flux (in the case of faint sources) or the known presence of a bright disc. We note that additional PACS 70/160 observations for some DUNES resolved disc targets are available through the DEBRIS programme (e.g. HIP 22263, HIP 62207), with which DUNES shares 106 sources. Following the recommended parameters laid out in the scan map release note[2] each scan map consisted of 10 legs of 3′ length, with a 4″ separation between legs, scanning at the medium slew speed (20″ per second). Each target was observed at two array orientation angles (70°and 110°) to improve noise supression and assist in the removal instrumental artefacts and gl itches from the images after mosaicking.

SPIRE (Griffin et al., 2010) small map observations were taken of 20 DUNES targets based on the expected target brightness at sub-mm wavelengths. Each SPIRE observation was composed of either two or five repeats (equivalent on-source time of either 74 or 185 s) of the small scan map mode[3], producing a fully sampled map covering a region 4 ′ around the target.

Data reduction

All data reduction was carried out in HIPE *Herschel* Interactive Processing Environment (Ott et al., 2010) user release version 10.0.0, PACS calibration version 45 and SPIRE calibration version 10 for the images presented here. Although the HIPE version and calibration versions are different to those with which the results in the survey paper were derived, the PACS photometry calibration has been stable

[1] http://www.mpia-hd.mpg.de/DUNES/
[2] See PICC-ME-TN-036 for details.
[3] See herschel.esac.esa.int/Docs/SPIRE/html/spire_om.pdf for details.

since version 32 (used to create the images in the survey paper) and . This change should not affect the validity of either the material presented in the archive or the results presented in Eiroa et al., 2013.

The individual PACS scans were processed with a high pass filter to remove background structure, using high pass filter radii of 15 frames at 70 μm, 20 frames at 100 μm and 25 frames at 160 μm, suppressing structure larger than 62″, 82″ and 102″ in the final images, respectively. In the highpass filtering process, sources were identified from the level 2 pipeline product using SExtractor and masked during a second pass through the data reduction with a 10″ radius for all three wavebands. A region centered on the expected target position was masked automatically with a 15″ radius (if point-like) or 30″ (if extended), again invariant of wavelength. For comparison, the largest disc in the DUNES survey had a FWHM extent of 18″×10″ at 100 μm. Deglitching was carried out using the second level spatial deglitching task. At 70 and 100 μm the two individual PACS scans were mosaicked to reduce sky noise and suppre ss $1/f$ striping effects from the scanning. At 160 μm typically only the two longest exposure scans (100/160) were combined unless the 70/160 combination were of similar duration (e.g. in the case of HIP 17439). This is due to the large sky noise values at 160 μm for single repetitions typically observed in thhe 70/160 scans (~ 9 mJy) compared to the noise in the longer duration 100/160 scans (~ 2-4 mJy) along with the faintness of the sources (~ 5–10 mJy) resulting in the combination of all four available scans simply adding noise to the longer duration 100/160 mosaic. The final image scales of the mosaics used for analysis are 1″ per pixel at 70 and 100 μm and 2″ per pixel at 160 μm, compared to native instrument pixel sizes of 3.2″ at 70 and 100 μm and 6.4″ at 160 μm.

For the SPIRE observations, the small maps were created using the standard pipeline routine in HIPE, using the naive mapper option. Image scales of 6″, 10″ and 14″ per pixel were used at 250, 350 and 500 μm.

Script

The script used to produce the images submitted to the archive is presented in Appendix A. The script was based on the pipeline script included with HIPE 10, with some variations following discussion with the HSC and PACS data reduction experts regarding the particular science goals of DUNES:

- The script was a batch process working on all sources in a given input table.
- The revised pointing product for each observation were used to correct for the astrometry errors induced by the star tracker distortions.

- Sources were identified in the individual scans using SExtractor during the reduction process and masked accordingly.
- A drop size (pixfrac parameter) of 1.0 was used since the method of background measurement was insensitive to the presence of correlated noise.
- Highpass filter widths of 15, 20 and 25″ were used rather than the standard 15, 15 and 25″ for 70, 100 and 160 μm.
- Over-sampled pixel sizes of 1″ at 70 and 100 μm and 2″ at 160 μm are used for the mosaic image scales used to measure the results. Native scale images are also available.

Description

Keywords

A list of the keywords added to the mosaic metadata and their definitions is presented in Table 1. Many of the keywords in the scan and mosaic meta data provided are taken directly from the original observation and are therefore not listed here, being self explanatory (e.g. the HIPE and calibration versions, WCS information, position and roll angle of the telescope, observation ids, etc.).

Table 1: Header parameters added to the FITS image metadata.

Keyword	Meaning
"PROGRAMME"	OTKP_DUNES: data observed for the DUNES programme.
"OBSERVER"	ceiroa: All observations can be found in the Herschel Science Archive searching by observer.
"PID"	Programme ID: OTKP_ceiroa_1 (main programme) or SDP_ceiroa_3 (science demonstration phase observations).
"TYPE"	Type of image: Scan map or Mosaic.
"OBS_ID"	Observation ID (for scans).
"OBS00x"	Observation IDs (for mosaics, where x is the scan number).
"DESC"	Description of contents: typically image, error, coverage and mask frames.
"T_ONSRC"	On-source time, as calculated in HSpot.
"PIXFRAC"	Pixel fraction parameter for the drop size in image reconstruction.
"PIXSIZE"	Pixel size of the image.
"HPFWIDE"	High pass filter width parameter.
"LEG_LENG"	Leg length of the observation, in arcmin.
"LEG_SEPN"	Leg separation of the observation, in arcsec.
"SLEW_SPD"	Slew speed of the telescope, in arcsec/second.

Frames

The DUNES mosaics contain four separate frames, namely the ‘image’, ‘exposure’, ‘error’ and ‘flag’. The ‘image’ frame contains the final mosaic (or scan) product from the DUNES data reduction script. The ‘exposure’ frame contains the number of readout measurements taken by PACS at each point in the ‘image’ frame. The ‘error’ frame, derived from the ‘exposure’ frame using the *photCoverage2Noise* procedure is a measurement of the uncertainty in the ‘image’ mosaic. Finally, the ‘flag’ frame shows the regions of the ‘image’ frame that were actually observed (‘0’ values, c.f. off-image parts of the mosaic which are denoted by ‘1’).

Bibliography

Eiroa, C., et al., 2013, A&A, 555, 11
Griffin, M.J., et al., 2010, A&A, 518, L3
Ott, S., 2010, in Astronomical Data Analysis Software and Systems XIX, ASP Conference Series, Vol. 434., p. 139
Pilbratt, G.L., et al., 2010, A&A, 518, L1

Appendix A: The script

```
#IMPORT PACS CALIBRATION
from herschel.pacs.spg            import *
from herschel.pacs.spg.common     import *
from herschel.pacs.spg.phot       import *
from herschel.pacs.spg.pipeline   import *
from herschel.pacs.signal.context import *
from herschel.pacs.signal         import SlicedFrames
from herschel.pacs.cal            import GetPacsCalDataTask
from herschel.ia.dataset          import LongParameter
from herschel.pacs.spg.all        import *
from herschel.pacs.signal import MapIndex
from herschel.pacs.spg.phot import MapIndexTask
from herschel.pacs.spg.phot import IIndLevelDeglitchTask

import os
import commands
import herschel.store.auth

#HARD WIRED DIRECTORIES
root_dir = '/home/jmarshall/mydata/dunes/redux_4/'
pp_dir = '/home/jmarshall/mydata/dunes/pointing/revised_pointing_products/'
#READ IN TARGET ID AND OBSIDS TO ARRAYS FROM ASCII FILE
ascii=AsciiTableTool()
ascii.template=TableTemplate(6,names=["target","obs1","obs2","ot","od","rslv"],\
                             types=["String","Long","Long","Integer","Integer","Integer"])
table=ascii.load("~/dir/dunes_input_table.txt")
#NUMBER OF TARGETS (OR WAVELENGTH PAIRS) TO BE ANALYZED
ntgt = len(table["target"].data)
#BREAK DOWN DATA TABLE BY TARGET, BLUE_CHANNEL, OBS1, OBS2
target_names = table["target"].data
#SCAN AND CROSS-SCAN OBSIDS
```

```
scan_obsids = table["obs1"].data
xscan_obsids = table["obs2"].data
onsource_time = table["ot"].data
opday = table["od"].data
rsv = table["rslv"].data
#ACTUAL SOURCE POSITION IN EACH FIELD
#Empty array for roll angles - at most a mosaic will have 4 observations
rollangle=[0.0,0.0,0.0,0.0]
#USERNAME AND PASSWORD (user will need to add their own!)
your_user_name = "yourusername"
your_password = "yourpassword"
login_usr = "hcss.ia.pal.pool.hsa.haio.login_usr"
login_pwd = "hcss.ia.pal.pool.hsa.haio.login_pwd"
Configuration.setProperty(login_usr,your_user_name)
Configuration.setProperty(login_pwd,your_password)

#Drop size for pixels
pixfrac = 1.0

#Radius for high pass filter masking
radius = 15.0

#used by SEXTRACTOR to identify sources for masking from the level 2 image
#assumed FWHM are the mean of the major and minor axis for 20\arcsec/s scans
#at each wavelength [70,100,160]
fwhm = [5.61,6.79,11.36]

#LOOP OVER ALL OBSERVATION PAIRS IN THE LIST
for nn in range(0,len(target_names)):
    # Obsids to use in making the 70 or 100 micron mosaic
    n_obs_blue = [scan_obsids[nn],xscan_obsids[nn]]
    # Obsids to use in making the 160 micron mosaic
    n_obs_red = [scan_obsids[nn],xscan_obsids[nn]]
    #If both 70/160 and 100/160 observations available, you can combine all 4 160um maps
    #be careful of offsets between epochs and the different observing times of the 70/160
    #and 100/160 scans, though.
    #if(nn >= 1)and(target_names[nn-1] == target_names[nn]):
    #  n_obs_red = [scan_obsids[nn-1],xscan_obsids[nn-1],scan_obsids[nn],xscan_obsids[nn]]

    # Name of the HIPE pool you saved the data to (same as target name)
    targetid = target_names[nn]
    # Is the disc resolved?
    resolved = rsv[nn]
    # Location of the HIPE pools on your computer
    pooldir = "~/.hcss/lstore/"
    #on source time
    tos = onsource_time[nn]
    od = opday[nn]
    # set some blank arrays
    map_wcs_red = []
    map_wcs_blue= []
    # define the channels to be processed
    channel  = ["blue","red"]
    bandfwhm = [0,fwhm[2]]
    # set blue channel names and creat directory for files
    bluobs=getObservation(scan_obsids[nn],verbose=True,useHsa=True,instrument='PACS')
    pacs_blue = bluobs.meta["blue"].value
    del(bluobs)
    if(pacs_blue == 'blue1'):
      blueid = "pacs70"
      channel[0] = "blue"
      bandfwhm[0] = fwhm[0]
```

```
if(pacs_blue == 'blue2'):
  channel[0] = "green"
  blueid = "pacs100"
  bandfwhm[0] = fwhm[1]
bandnames= [blueid,"pacs160"]
### ADD IN DIRECTORIES FOR DATA FILES TO GO IN ###
# output directory, attempt to make directories
os.system('mkdir '+root_dir+targetid+"/")
os.system('mkdir '+root_dir+targetid+"/"+blueid+"/")
direc = root_dir+targetid+"/"+blueid+"/"

### ACTUAL DATA REDUCTION STARTS HERE ###

for i in range(0,len(channel)):

  color = channel[i]

  if color=='red':
    n_obs = n_obs_red
  elif color=='green':
    n_obs = n_obs_blue
  elif color=='blue':
    n_obs = n_obs_blue

#BLANK ARRAYS FOR VARIOUS THINGS
  scanlist = []
#RESET MAPS TO AVOID PIXEL SIZE PROBLEMS
  map_wcs = []

  for j in range(len(n_obs)):
    obsid = n_obs[j]
  # observation id :
  #  get observation context from HSA:
    obs = -1
  #proprietary files
    if targetid == 'hip7978' and pacs_blue_channel[nn]==2:
      obs=getObservation(obsid,poolName=targetid, poolLocation=pooldir,verbose=True,\
                         useHsa=False,instrument='PACS')
    if obs == -1:
      obs=getObservation(obsid,verbose=True,useHsa=True,instrument='PACS')
  #TARGET LOCATION FROM META DATA
    tgt_pmra = obs.meta["pmRA"].value
    tgt_pmdec = obs.meta["pmDEC"].value
    rasource=obs.meta["raNominal"].value
    decsource=obs.meta["decNominal"].value
    cosdec=COS(decsource*Math.PI/180.)

    rollangle[j] = obs.meta["posAngle"].value
    if color=='red':
      rad1=150
      rad2=150
      outpixsz=2.0
      hpfrad1=100
      hpfrad2=25
      bandid = bandnames[1]
      wavelength = '160 microns'
      map_wcs_red=Wcs(cunit1="Degrees",cunit2="Degrees",cdelt1=-1.0*outpixsz/3600.,\
        cdelt2=outpixsz/3600.,crota2=0.,crpix1=rad1,crpix2=rad2,\
        crval1=rasource,crval2=decsource,ctype1="RA---TAN",ctype2="DEC--TAN",\
        equinox=2000.0)
      map_wcs_red.setParameter("naxis1",2*rad1,"naxis1")
      map_wcs_red.setParameter("naxis2",2*rad2,"naxis2")
```

```
        elif color=='green':
          rad1=300
          rad2=300
          outpixsz=1.0
          hpfrad1=100
          hpfrad2=20
          n_obs = n_obs_blue
          bandid = bandnames[0]
          wavelength = '100 microns'
          map_wcs_blue=Wcs(cunit1="Degrees",cunit2="Degrees",cdelt1=-1.0*outpixsz/3600.,\
            cdelt2=outpixsz/3600.,crota2=0.,crpix1=rad1,crpix2=rad2,\
            crval1=rasource,crval2=decsource,ctype1="RA---TAN",ctype2="DEC--TAN",\
            equinox=2000.0)
          map_wcs_blue.setParameter("naxis1",2*rad1,"naxis1")
          map_wcs_blue.setParameter("naxis2",2*rad2,"naxis2")
        elif color=='blue':
          rad1=300
          rad2=300
          outpixsz=1.0
          hpfrad1=100
          hpfrad2=15
          n_obs = n_obs_blue
          bandid = bandnames[0]
          wavelength = '70 microns'
          map_wcs_blue=Wcs(cunit1="Degrees",cunit2="Degrees",cdelt1=-1.0*outpixsz/3600.,\
            cdelt2=outpixsz/3600.,crota2=0.,crpix1=rad1,crpix2=rad2,\
            crval1=rasource,crval2=decsource,ctype1="RA---TAN",ctype2="DEC--TAN",\
            equinox=2000.0)
          map_wcs_blue.setParameter("naxis1",2*rad1,"naxis1")
          map_wcs_blue.setParameter("naxis2",2*rad2,"naxis2")

        print ""
        print "Reducing OBSID:", obsid, " (", i+1, "/", len(n_obs), ")"

      ############ SETTINGS ##########################################################
      #extract the frames from the observation context "obs"
        if color=='blue' or color=='green':
          frames=obs.level0.refs["HPPAVGB"].product.refs[0].product
          map_wcs = map_wcs_blue
        elif color=='red':
          frames=obs.level0.refs["HPPAVGR"].product.refs[0].product
          map_wcs = map_wcs_red

# ------------------------------------------------------------------------------------
# Extract the PointingProduct
        pp = obs.auxiliary.pointing
# correct pointing product according to revised pps from ESAC
# this is no longer required after v9.1.0 - the revised pointing is automatically included
#        if(od >= 320) and (od <= 761):
#          pp = fitsReader(pp_dir+'/pointing_od_0'+str(od)+'.fits')
# Extract the calibration tree
        calTree = getCalTree(version=45)
# set the calibration tree as default
        GetPacsCalDataTask.setDefaultCalTree(calTree)
# Extract Housekeeping parameters
        photHk = obs.refs["level0"].product.refs["HPPHK"].product.refs[0].product["HPPHKS"]
# Extract OrbitEphemeries
        orbitEphem = obs.auxiliary.orbitEphemeris

#######################
# LEVEL 0 -> LEVEL 0.5
#######################
```

```
        frames = filterSlew(frames)
        frames = findBlocks(frames,calTree=calTree)
        frames = detectCalibrationBlock(frames)
        frames = removeCalBlocks(frames,useBbid=1)
        frames = photFlagBadPixels(frames,calTree=calTree,scical="sci",keepall=False)
        frames = photFlagSaturation(frames, calTree=calTree, hkdata=photHk)
        frames = photConvDigit2Volts(frames, calTree=calTree)
        frames = convertChopper2Angle(frames, calTree=calTree)
        frames = photAddInstantPointing(frames, pp, orbitEphem = orbitEphem,calTree = calTree)
# DELETE UNNECESSARY VARIABLES
        del(pp,photHk,orbitEphem)
# Save the level 0.5 data as a frames file
        savefile = direc+"frame_"+"_" + str(obsid) + "_" + bandid + "_" + "Level_0.5.fits"
        simpleFitsWriter(frames,savefile)
#######################
# LEVEL 0.5 -> LEVEL 1
#######################
        frames = photRespFlatfieldCorrection(frames, calTree = calTree)
# Save the level 1 data as a frames file
        savefile = direc+"frame_"+"_" + str(obsid) + "_" + bandid + "_" + "Level_1.fits"
        simpleFitsWriter(frames,savefile)
#####################
# LEVEL 1 -> LEVEL 2
#####################
# Mask the central region
        ralist = Double1d(1)
        ralist[0] = rasource
        declist = Double1d(1)
        declist[0] = decsource
        rad = Double1d(1)
        rad[0] = 15.0

# Use bigger rad size for large discs in the sample
        if resolved == 1:
          rad[0] = 30.0
# Create a list of sources in the field based on the level2 image from the obs
        if(color == 'red'):
          map=obs.refs["level2"].product.refs["HPPPMAPB"].product
        if(color == 'blue' or color=='green'):
          map=obs.refs["level2"].product.refs["HPPPMAPR"].product
        sourceList = sourceExtractor(image=map, algorithm="sussextractor", detThreshold=10.0,\
            fwhm=bandfwhm[i], pixelRegion=1.5,fluxPriorsLambda=0.0, fitBackground=True,\
            useSignalToNoise=False, fluxPriorsMin=1.0E-4, fluxPriorsMax=1.0E8,getFilteredMap=False,\
            getPrf=False, doApertureCorrection=True)
        ralist = ralist.append(sourceList["sources"]["ra"].data)
        declist = declist.append(sourceList["sources"]["dec"].data)
        radlist = [10.0]*(len(ralist)-1)
        radlist = Double1d([float(s) for s in radlist])
        rad = rad.append(radlist)

        from herschel.pacs.spg.phot import MaskFromCatalogueTask
        mfc = MaskFromCatalogueTask()
        mask = mfc(map, ralist, declist, rad, copy = 1)
        frames  = photReadMaskFromImage(frames, si=mask, extendedMasking=True,\
            maskname="HighpassMask")

# Project hpfmask into an image
        frames_masked = frames.copy()
        objectMask = frames_masked.getMask('HighpassMask').copy()
        frames_masked.setSignal(Double3d(objectMask))
        frames_masked = deactivateMasks(frames_masked,String1d(["HighpassMask"]))
        hpfmask = photProject(frames_masked,calTree=calTree,outputPixelsize=outpixsz,\
             pixfrac=pixfrac,wcs=map_wcs)
        Display(hpfmask)
        hpfmask.image[hpfmask.image.where(hpfmask.image > 0.0)] = 1.0
        hpfmask.image[hpfmask.image.where(hpfmask.coverage < 0.1)] = -1.0
        # Save the hpfmask image for later use (add to Mosaic product)
        outfile = direc+ "hpfmask_Scan_"+targetid+"_" + str(obsid) +"_"+ bandid +"_hpf"+\
             str(hpfrad2)+".fits"
        simpleFitsWriter(hpfmask,outfile)

# Delete unneccesary parameters
        del(ralist,declist,rad,map,frames_masked,objectMask)

# Run the high pass filter on a frames class, where the sources are masked (using HighpassMask)
        frames  = maskedHighpassFilter(frames,hpfrad2,maskname="HighpassMask",\
                  interpolateMaskedValues=True)

        frames  = filterOnScanSpeed(frames,limit=10)

# spatial deglitching - define properties
        from herschel.pacs.spg.phot.deglitching.map import MapDeglitchTask
        s = Sigclip(nsigma=30,behavior="clip",outliers="both",mode=Sigclip.MEDIAN)
# deglitch map
        mdt = MapDeglitchTask()
        deg = mdt(frames,deglitchvector='timeordered',maskname='2nd level glitchmask',algo=s)
#        frames_masked = frames.copy()
#        objectMask = frames_masked.getMask('2nd level glitchmask').copy()
#        frames_masked.setSignal(Double3d(objectMask))
#        frames_masked = deactivateMasks(frames_masked,String1d(["2nd level glitchmask"]))
# Project the glitch mask into an image
#        glitchmask = photProject(frames_masked,calTree=calTree,outputPixelsize=outpixsz,\
#                pixfrac=pixfrac,wcs=map_wcs)
#        glitchmask.image[glitchmask.image.where(glitchmask.image > 0.0)] = 1.0
#        glitchmask.image[glitchmask.image.where(glitchmask.coverage < 0.1)] = -1.0
#        Display(glitchmask)
# Save the glitchmask image for later use (add to Mosaic product)
#        outfile = direc+ "glitchmask_Scan_"+targetid+"_" + str(obsid) +"_"+ bandid +"_hpf"+\
#             str(hpfrad2)+".fits"
#        simpleFitsWriter(glitchmask,outfile)
# Delete unneccesary parameters
#        del(frames_masked,objectMask)

#################
# Make the image
#################
        image = photProject(frames,calTree=calTree,outputPixelsize=outpixsz,pixfrac=pixfrac,\
           wcs=map_wcs)
        Display(image)
# Create error map
        photCoverage2Noise(image, hp = hpfrad2, pixfrac = 1.0)
# Add metadata to image
        image["hpfmask"] = hpfmask["image"]
#        image["glitchmask"] = glitchmask["image"]
        image.meta["PROGRAMME"] = StringParameter("OTKP_DUNES")
        image.meta["OBSERVER"] = StringParameter("ceiroa")
# Exact observing programme for the obsid
        obs_programme = "OTKP_ceiroa_1"
        if(obsid == 1342195666 or obsid == 1342187328 \
           or obsid == 1342187142 or obsid == 1342187141):
           obs_programme = "SDP_ceiroa_3"
        image.meta["PID"] = StringParameter(obs_programme)
        image.meta["INSTRUMENT"] = StringParameter("PACS")
        image.meta["TYPE"] = StringParameter("SCAN IMAGE")
        image.meta["DESC"] = StringParameter("IMAGE COVERAGE AND UNCERTAINTY MAPS")
        image.meta["WAVELENG"] = StringParameter(wavelength)
        image.meta["TARGET"] = StringParameter(targetid)
        image.meta["PM_RA"] = StringParameter(str(tgt_pmra))
        image.meta["PM_DEC"] = StringParameter(str(tgt_pmdec))
        image.meta["REFFRAME"] = StringParameter("ICRS")
        image.meta["EQUINOX"] = StringParameter("2000.0")
        image.meta["RIGHTASC"] = StringParameter(str(rasource))
        image.meta["DECLINAT"] = StringParameter(str(decsource))
        image.meta["T_ONSRC"] = StringParameter(str(tos))
        image.meta["vHIPE"] = StringParameter("10.0.0")
        image.meta["vPACScal"] = StringParameter("45")
        image.meta["PIXFRAC"] = StringParameter("1.0")
        image.meta["PIXSIZE"] = StringParameter(str(outpixsz))
        image.meta["HPFWIDE"] = StringParameter(str(hpfrad2))
        image.meta["SCAN_ID"] = StringParameter(str(obsid))
        image.meta["posAngle"] = StringParameter(str(rollangle[j]))
        image.meta["arrAngle"] = StringParameter(str(obs.meta["mapScanAngle"].value)+" "+"deg")
        image.meta["leg_leng"] = StringParameter(str(obs.meta["mapScanLegLength"].value)+" "+"arcmin")
        image.meta["leg_sepn"] = StringParameter(str(obs.meta["mapScanCrossScan"].value)+" "+"arcsec")
        image.meta["slew_spd"] = StringParameter(str(obs.meta["mapScanSpeed"].value))
# Save image to fits file
        outfile = direc+ "Scan_"+targetid+"_" + str(obsid) +"_"+ bandid +"_hpf"+str(hpfrad2)+".fits"
        print "Saving file: " + outfile
        simpleFitsWriter(image,outfile)

# Delete unneccesary variables
        del(frames,obs,hpfmask)
        System.gc()

##############
# MOSAICKING
##############
    from java.util import ArrayList
    from herschel.ia.toolbox.image import MosaicTask

    for nc in range(len(channel)):
      print channel[nc]
# camera :
      camera = channel[nc]
      bandid = bandnames[nc]

      if camera=='red':
        outpixsz=2.0
        hpfwidth=25
        n_obs = n_obs_red
        bandid = bandnames[1]
        map_wcs = map_wcs_red
        wavelength = '160 microns'
      elif camera=='green':
        outpixsz=1.0
        hpfwidth=20
        n_obs = n_obs_blue
        bandid = bandnames[0]
        map_wcs = map_wcs_blue
        wavelength = '100 microns'
      elif camera=='blue':
        outpixsz=1.0
        hpfwidth=15
        n_obs = n_obs_blue
        bandid = bandnames[0]
        map_wcs = map_wcs_blue
        wavelength = '70 microns'

#Create a list of images to be mosaicked
      images = ArrayList()
#      glmap = ArrayList()
      hpmap = ArrayList()
      for nb in range(len(n_obs)):
        obsid=n_obs[nb]
        map_file = direc+ "Scan_"+targetid+"_" + str(obsid) + "_" + bandid + "_hpf"+\
              str(hpfwidth)+".fits"
#        gl2_file = direc+ "glitchmask_Scan_"+targetid+"_" + str(obsid) +"_"+ bandid +\
              "_hpf"+str(hpfwidth)+".fits"
        hpf_file = direc+ "hpfmask_Scan_"+targetid+"_" + str(obsid) +"_"+ bandid +"_hpf"+\
              str(hpfwidth)+".fits"
        map = simpleFitsReader(map_file)
#        gl2map = simpleFitsReader(gl2_file)
        hpfmap = simpleFitsReader(hpf_file)
        map.exposure = map.coverage
#        glmap.add(gl2map)
        hpmap.add(hpfmap)
        images.add(map)
# Mosaic the selected scans
      mosaic = MosaicTask()(images=images,oversample=0)
      hp_msc = MosaicTask()(images=hpmap,oversample=0)
#      gl_msc = MosaicTask()(images=glmap,oversample=0)

      for n in xrange(len(mosaic.image)):
        if mosaic["flag"].data[n]==1: mosaic["image"].data[n]=0

# Add mosaics of the masks used in the processing
      mosaic["hpfmask"] = hp_msc["image"]
#      mosaic["glitchmask"] = gl_msc["image"]
      Display(mosaic)
# Add metadata to mosaic image
      mosaic.meta["CREATOR"] = StringParameter("OTKP_DUNES")
      mosaic.meta["OBSERVER"] = StringParameter("ceiroa")
# Main programme = KPOT_ceiroa_1, SDP phase = SDP_ceiroa_3
      mosaic.meta["PROGRAMS"] = StringParameter("KPOT_ceiroa_1")
      mosaic.meta["INSTRUME"] = StringParameter("PACS")
      mosaic.meta["TYPE"] = StringParameter("MOSAIC IMAGE")
      mosaic.meta["DESC"] = StringParameter("IMAGE COVERAGE AND UNCERTAINTY MAPS")
      mosaic.meta["WAVELENG"] = StringParameter(wavelength)
      mosaic.meta["TARGET"] = StringParameter(targetid)
      mosaic.meta["PM_RA"] = StringParameter(str(tgt_pmra))
      mosaic.meta["PM_DEC"] = StringParameter(str(tgt_pmdec))
      mosaic.meta["REFFRAME"] = StringParameter("ICRS")
      mosaic.meta["EQUINOX"] = StringParameter("2000.0")
      mosaic.meta["RIGHTASC"] = StringParameter(str(rasource))
      mosaic.meta["DECLINAT"] = StringParameter(str(decsource))
      mosaic.meta["T_ONSRC"] = StringParameter(str(2*tos))
      mosaic.meta["vHIPE"] = StringParameter("10.0.0")
      mosaic.meta["vPACScal"] = StringParameter("45")
      mosaic.meta["PIXFRAC"] = StringParameter("1.0")
      mosaic.meta["PIXSIZE"] = StringParameter(str(outpixsz))
      mosaic.meta["HPFWIDE"] = StringParameter(str(hpfwidth))
# Loop over obsids to add details for each scan
      for ns in range(n_obs):
        mosaic.meta["SCAN_ID"+str(ns+1)] = StringParameter(str(n_obs[ns]))
        mosaic.meta["posAngle_"+str(ns+1)] = StringParameter(str(rollangle[ns]))
# Save image to fits file
      outfile = direc+ "Mosaic_"+targetid+"_" + camera +"_"+ bandid +"_hpf"+str(hpfwidth)+\
           "_pix"+str(outpixsz)+".fits"
      print "Saving file: " + outfile
      simpleFitsWriter(mosaic,outfile)

#DELETE LEVEL 1 .SAV FILES FROM DIRECTORY TO SAVE SPACE
    os.system('rm -rf '+root_dir+targetid+"/"+blueid+"/frame*.fits")
#DELETE MAP FILES FROM DIRECTORY TO SAVE SPACE (ONLY NEED SCANS AND MOSAICS)
    os.system('rm -rf '+root_dir+targetid+"/"+blueid+"/map*.fits")
#DELETE .HCSS POOL TO SAVE SPACE (THIS IS PROBABLY NOT WISE...)
    if targetid != 'hip7978':
      os.system('rm -rf '+pooldir+targetid)

    print "INFO: Finished reducing "+targetid+" "+blueid+"/160, target number "+str(nn+1)+" of "+str(ntgt)

    System.gc()
```