# Table of Contents

Preface ......................................................................................................................... viii
1. Introduction ................................................................................................................ 1
   1.1. Scope and Structure of this Data Reduction Guide .......................................... 1

2. SPIRE Launch Pad: Data Reduction Overview ...................................................... 4
   2.1. Data Launch Pad ................................................................................................. 4
      2.1.1. Obtaining SPIRE data from the archive and importing into HIPE ............. 4
      2.1.2. Looking at your Data .................................................................................. 4
   2.2. Photometer Launch Pad ...................................................................................... 5
      2.2.1. Does the Photometer observation data need re-processing? ....................... 5
      2.2.2. Re-processing with the User Pipeline Scripts ........................................... 5
      2.2.3. Further Analysis ......................................................................................... 6
   2.3. Spectrometer Launch Pad .................................................................................. 7
      2.3.1. Does the observation data need re-processing? .......................................... 7
      2.3.2. Re-processing with the User Pipeline Scripts ........................................... 8
      2.3.3. Further Analysis ......................................................................................... 9

3. Overview of Scripts in HIPE ................................................................................... 10

4. SPIRE Observation Context Data Structure .......................................................... 13
   4.1. Accessing SPIRE Data ....................................................................................... 13
      4.1.1. Understanding how HIPE accesses data ................................................. 13
      4.1.2. Accessing data directly from the Herschel Science Archive ................. 14
      4.1.3. Querying the SPIRE Observation Log ..................................................... 16
      4.2. SPIRE Observation Context Data Structure ................................................. 18
         4.2.1. Anatomy of a SPIRE Observation: Products, Pools, Storage, and Building
                Blocks ........................................................................................................... 18
         4.2.2. Linking it altogether: Introducing the Context ......................................... 19
         4.2.3. The Level 2.5 Data in the Herschel Science Archive ............................. 22
         4.2.4. The Level 3. Data in the Herschel Science Archive ............................... 24
         4.2.5. Looking at your Observation Context in HIPE .................................. 25
         4.2.6. What the Observation Context looks like on your hard disk .............. 28
   4.3. Pipelines, versions and the archive ................................................................... 29

5. SPIRE Calibration Data .......................................................................................... 31
   5.1. The SPIRE Calibration Context ....................................................................... 31
   5.2. The SPIRE Calibration Tree ............................................................................ 32
   5.3. SPIRE Calibration Product Editions ................................................................. 32
   5.4. Updating a Calibration Tree .............................................................................. 33
   5.5. The SPIRE Calibration Task GUI and Calibration Viewer ............................. 36
   5.6. Updating Individual Calibration Products ...................................................... 39
   5.7. Removing Calibration Products from the Tree ............................................... 40
   5.8. Which Calibration Tree Version was used to Process my Data? ..................... 40
   5.9. Further Information ......................................................................................... 41

6. SPIRE Photometer Mode Cookbook ....................................................................... 42
   6.1. SPIRE Scan Map and Data Structure ............................................................... 42
      6.1.1. A first look at your image maps (The Level 2 Data Product) .................... 42
      6.1.2. Saving a map as a FITS file and reading it in again ............................... 45
      6.1.3. Looking at observations with Level 2.5 and Level 3 products .............. 46
      6.1.4. Looking at the Level 1 Timeline Data ..................................................... 50
      6.1.5. SPIRE Serendipity Data at Level 1 ......................................................... 52
      6.1.6. Looking at the Level 0.5 Timeline Data ................................................. 55
      6.1.7. Looking at the Raw Level 0 Data ............................................................. 58
   6.2. SPIRE Point Source Mode Data Structure ...................................................... 59
7.3.12. Correct time domain phase in the detector signals ........................................ 200
7.3.13. Create a SPIRE Pointing product ......................................................... 200
7.3.14. Interpolate SDT and SMECT to create interferograms .................................. 200
7.3.15. Subtract the interferogram baseline ...................................................... 201
7.3.16. Apply second level deglitching ............................................................. 203
7.3.17. Correct interferogram phase .................................................................... 204
7.3.18. Fourier transform the interferograms ...................................................... 204
7.3.19. Fetch calibration files ............................................................................. 205
7.3.20. Remove out-of-band power ..................................................................... 206
7.3.21. Apply bright gain correction ................................................................... 206
7.3.22. Correct for instrument emission .............................................................. 207
7.3.23. Apply the extended source flux conversion ............................................ 207
7.3.24. Correct for telescope emission ............................................................... 208
7.3.25. Apply the point-source flux calibration .................................................. 209
7.3.26. Apply the low resolution correction ........................................................ 210
7.3.27. Average the spectra .............................................................................. 212
7.3.28. Apodization .......................................................................................... 212
7.3.29. Correct the frequency axis to LSR .......................................................... 213
7.3.30. Sort the meta data order .......................................................................... 213
7.3.31. Apply the extended calibration correction ............................................... 213
7.3.32. Add overlapping photometer observation IDs ........................................ 214
7.3.33. Save the Level-2 products as FITS ........................................................... 215
7.3.34. Create spectral cubes (mapping script) .................................................... 215
7.3.35. Update and store the Observation Context ............................................. 217
7.4. Pointing Considerations ............................................................................. 218
  7.4.1. SPIRE BSM Position .............................................................................. 220
  7.4.2. Correcting the pointing ......................................................................... 221
7.5. Recipes for faint and medium-strength sources .......................................................... 222
  7.5.1. Comparing point-source and extended-source calibration ....................... 223
  7.5.2. Optimising background subtraction ....................................................... 223
  7.5.3. Checking Spectral Noise ..................................................................... 228
  7.5.4. Comparing with the SPIRE Photometer .............................................. 230
7.6. Recipes for semi-extended sources .................................................................. 231
  7.6.1. Identifying partially-extended sources .................................................. 231
  7.6.2. Does my spectrum need correcting? ...................................................... 233
  7.6.3. Using the SECT GUI ........................................................................... 236
  7.6.4. Pointing considerations ...................................................................... 239
  7.6.5. Scripting with SECT ........................................................................ 239
  7.6.6. How much can I believe the SECT results? ......................................... 240
7.7. Recipes for bright sources (above 500 Jy) ....................................................... 241
7.8. Recipes for mapping observations ................................................................. 242
  7.8.1. Understanding the SPIRE beam and how it relates to mapping observations 242
  7.8.2. Check clipping ..................................................................................... 243
  7.8.3. Pointing Information ........................................................................ 244
  7.8.4. Restricting or expanding the data for gridding ....................................... 246
  7.8.5. The gridding algorithms ................................................................... 246
  7.8.6. Holes in the map: examining the coverage ............................................ 251
  7.8.7. Maps with faint continuum levels ......................................................... 252
7.9. Observations with few repetitions ................................................................. 253
  7.9.1. Removing data ................................................................................... 253
  7.9.2. Correcting data .................................................................................. 254
7.10. Comparison with the SPIRE photometer .................................................... 257
  7.10.1. The FTS Synthetic Photometry task .................................................. 257
  7.10.2. Point source observations ................................................................. 258
  7.10.3. Mapping observations ...................................................................... 259
7.11. Spectral Analysis ...................................................................................... 262
  7.11.1. Local Standard of Rest ................................................................. 262
7.11.2. Joining SLW and SSW data ......................................................... 263
7.11.3. Visualising SPIRE spectra ......................................................... 263
7.11.4. Spectral fitting ......................................................................... 264
7.11.5. Sinc model .............................................................................. 264
7.11.6. Gaussian model for apodized data ............................................ 265
7.11.7. Partially resolved lines ............................................................. 266
7.11.8. Interactive line fitting using the Spectrum Fitter GUI ............... 268
7.11.9. SPIRE Spectrometer Line Fitting script ...................................... 270
7.11.10. Calculating line fitting errors ................................................... 271
7.11.11. Line identification .................................................................. 272
7.11.12. Joining SPIRE and PACS point spectra .................................... 272
7.11.13. Comparing SPIRE and HIFI spectra ........................................ 272

7.12. Cube Analysis ............................................................................ 273

7.12.1. SPIRE FTS cubes ..................................................................... 275
7.12.2. Extracting spectra from a spectral cube ...................................... 276
7.12.3. Extracting a point source from a spectral cube ......................... 276
7.12.4. Matching spectra from SSW and SLW cubes ......................... 277
7.12.5. Convolving a cube to a different beam size .................................. 278
7.12.6. Cube fitting with the Spectrum Fitter GUI .................................. 283
7.12.7. Cube fitting and line intensity/velocity maps .................................. 283
7.12.8. Publication quality plots for mapping data .................................. 285
7.12.9. Comparison with photometer maps .......................................... 287

8. SPIRE Visualisation Tools ................................................................ 288

8.1. SDI/SDS Explorer ........................................................................ 288
8.1.1. Starting SDI/SDS Explorer ......................................................... 288
8.1.2. SDI/SDS Explorer Layout ........................................................... 288
8.1.3. Example 1: Plotting and Overplotting ........................................ 292
8.1.4. Example 2: Making a Thumbnail Image ....................................... 294
8.1.5. Example 3: Plotting complex data: real, imaginary, absolute, and phase ................................................................. 295

8.2. Detector Timeline Explorer (DTE) ............................................... 297
8.2.1. Starting DTE .................................................................. 297
8.2.2. DTE Layout ................................................................ 297
8.2.3. DTE Preferences ................................................................. 304
8.2.4. Example 1: Plotting functions ................................................. 304
8.2.5. Example 2: Browse through time across the timeline ............... 305
8.2.6. Example 3: Visualize the flagged detectors in the Array Display .......... 305

8.3. SPIRE Bolometer Finder .............................................................. 306
8.3.1. Purpose ........................................................................ 306
8.3.2. The BoloFinderTool .............................................................. 306
8.3.3. Interacting with the BoloFinder via Scripting ............................ 310
8.3.4. Specifying the map pixel via the command line ....................... 312
8.3.5. Visualising timelines with the plotAllData() method .................. 313
8.3.6. Identifying outliers with the Bolometer Finder Useful Script and the SPIRE Mask Editor ............................................. 314

8.4. SPIRE Mask Handling .................................................................. 315
8.4.1. SPIRE Mask Formalization ....................................................... 315
8.4.2. SPIRE Mask Editor Viewer ....................................................... 317
8.4.3. Other ways to view SPIRE masks ............................................ 321

8.5. Quality Control for SPIRE pipeline processed data ....................... 322
8.5.1. Quality Control for SPIRE pipeline processed data ....................... 322

9. Advanced HIPE Tips ...................................................................... 323
9.1. Saving an observation context to a pool on disk ............................ 323
9.2. Using TEMP STORAGE to overcome memory problems with HIPE ... 323
9.3. Filtering variables in the HIPE variable list .................................... 325

10. Glossary ......................................................................................... 327

11. Reprocessing with the SPIA .......................................................... 329
11.1. Overview ............................................................................... 329
11.2. Why reprocess with the SPIA? ................................................ 329
11.3. SPIA Setup and Workflows ................................................................. 330
11.4. Locating the SPIA tasks within HIPE .................................................. 333
11.5. Retrieving observations from the Herschel Science Archive into local pools .... 333
11.6. Loading your observation into your HIPE session ................................. 334
11.7. Downloading the SPIRE calibration tree ............................................. 334
11.8. Loading SPIRE calibration information into HIPE ................................. 335
11.9. Reprocessing your data to calibrated timelines .................................... 335
11.10. Making maps from one or more observations ...................................... 337
11.11. Inspecting your reprocessed maps ...................................................... 338
11.12. Inspecting concatenated calibrated timelines ..................................... 339
11.13. Correcting artifacts via manual inspection ........................................ 340
   11.13.1. Processing to Level0.5 and detecting artifacts ............................ 340
   11.13.2. Correcting artifacts via manual inspection .................................. 341
   11.13.3. Repairing artifacts and reprocessing to Level 1 .......................... 342
11.14. Saving the reprocessed results ........................................................ 343
11.15. Worked Examples ............................................................................. 344
   11.15.1. Map-making with turnaround data and destriping ....................... 344
   11.15.2. Destriping two large maps ......................................................... 344
12. References .......................................................................................... 345
   12.1. SPIRE DRG References .................................................................... 345
Preface

1. Versioning

On the front page of this manual is a version number. The HIPE version that the DRG corresponds to is given followed by a traditional versioning system (1.0, 1.1, 1.2, etc...), and the changes introduced with each version are detailed below. Also shown on the front page is the date of publication of the manual.

1.1. Changelog

The following was changed for 5.1

• The DRG was updated in response to the status of the data reduction pipeline for HIPE 14. the DRG was updated to reflect the changes to the new SPIRE Calibration Tree 14.0. Many minor changes and updates made to both the photometer and spectrometer sections.

The following was changed for 5.0

• The DRG was updated in response to the status of the data reduction pipeline for HIPE 14. the DRG was updated to reflect the changes to the new SPIRE Calibration Tree 14.3. The Photometer photometry section has been updated. The Spectrometer sections have been restructured to reflect the change in the SPeCTrometer Observation Context for all spectrometer observations.

The following was changed for 4.0

• The DRG was updated in response to the status of the data reduction pipeline for HIPE 13. the DRG was updated to reflect the changes to the new SPIRE Calibration Tree 13.1. The Photometer photometry section has been updated. The photometer zero-point correction section was updated. An explanation of the Photometer 2-pass pipeline has been added. The Spectrometer sections have been restructured to reflect the change in the SPeCTrometer Observation Context for all spectrometer observations.

The following was changed for 3.0

• The DRG was updated in response to the status of the data reduction pipeline for HIPE 12. the DRG was updated to reflect the changes to the new SPIRE Calibration Tree 12.2. The Photometer photometry section has been updated. The Spectrometer sections have been restructured including the addition of a section on comparing photometer and spectrometer data.

The following was changed for 2.3

• The DRG was updated in response to the status of the data reduction pipeline for HIPE 11. New section on Herschel-Planck cross calibration added.

The following was changed for 2.2

• The DRG was updated in response to the status of the data reduction pipeline for HIPE 10.

The following was changed for 2.1

• Updated Launch Pad section. Photometer, spectroscopy and calibration chapter updates. Updated source extraction section. Updates on the usage of myHSA.

The following was changed for 2.0

• Reorganisation of the SDRG. Updates in compliance with the status of the pipelines for version 8.0 of HIPE. Added SPIRE Launch Pad section. Updated source extraction section and beam sizes.
The following was changed for 1.6

- Updates in compliance with the status of the pipelines for version 8.0 of HIPE.

The following was changed for 1.5

- Updates to the Appendix sections in compliance with the status of the pipelines for version 7.0 of HIPE. New sections on merging observations and spectral mosaicing. Updates to the scan map and jiggle map sections. Added caveat on photometer error maps.

The following was changed for 1.4

- Updates to the introduction, calibration, quality control, mask info for the large and small map reprocessing sections, spectral reprocessing and Appendix sections in compliance with the status of the pipelines for version 6.0 of HIPE.

The following was changed for 1.3

- Updates to the introduction, calibration, photometer, spectral reprocessing and Appendix sections. Added sections on masks and saving processed products to pools. Updated DTE section text. Added sections on aperture photometry and source extraction. Added list of scripts to Appendix, and updated the FAQ section.

The following was changed for 1.2

- Updates to the spectral reprocessing section.

The following was changed for 1.1

- Added a Glossary/FAQ section.
- Added subsections on accessing your data, the list of SPIRE docs and on using the DTE.
- Major updates to all sections to conform to data products and data processing as of the 5.0 branch.
- Renamed the SDUM as the SPIRE Data Reduction Guide.
- Restructured the sections to give a more focused treatment for each of the SPIRE modes, combining the product and reprocessing sections for each.

The following was changed for 1.0

- Major updates to all sections to conform to data products and data processing as of the 4.0 branch.
- Added SPIRE Calibration chapter.
- Added additional section on the Spectrum Explorer for SPIRE.
- Added reprocessing section for Small Map Mode.
- Expanded reprocessing section for spectrometer pipeline.

The following was changed for 0.2

- Updates to flow charts with respect to the 4.0 branch.

The following was changed for v0.1
• First version of the SDUM manual.
Chapter 1. Introduction

1.1. Scope and Structure of this Data Reduction Guide

The purpose of this document is to provide a comprehensive reference for the standard pipeline processing of SPIRE observations, and to act as a guide for the reprocessing and analysis of the final data products.

The data structure and reprocessing guide examples contained within the SPIRE Data Reduction Guide are based upon the current HIPE release - views may differ and examples may not work on previous and subsequent releases of HIPE.

For more information on obtaining HIPE and on how to install it, please see the HIPE Quick Start Guide and the HIPE Owners Guide.

1.2. SPIRE Observing Modes

SPIRE observing modes for both the Photometer and the Spectrometer are provided as Astronomical Observation Templates (AOTs), and the way these AOTs are referred to may differ from resource to resource (HSpot, HIPE, etc). There are currently 6 available observing modes in various levels of use and release, these are:

- **Large Map Mode (Scan Mapping, POF5)**: Used for observations of large fields (>4x4 arcmins). The telescope scans line-by-line, building up a map. Scan lines can be orthogonally cross-linked to produce high quality maps.

- **Small Map Mode (1x1 Small Scan Map, POF10)**: Used for observations of small fields (>1x1 arcmins). This mode replaces the former small map 64-point Jiggle, POF3 mode. The new Small Scan Map mode consists of 2 orthogonal scan lines of fixed length. The mode operation and processing is essentially the same as the Large Map mode. For a given observation, the area covered by both scan legs defines a central square of side 5 arcmins although the length of the two orthogonal scan paths are somewhat longer than this. In practice, due to the position of the arrays on the sky at the time of a given observation, the guaranteed area for scientific use is a circle of diameter 5 arcmins.

- **Point Source Mode (7-point Jiggle, POF2)**: In this mode, the telescope stares at a target and the detector arrays are jiggled, using the Beam Steering Mirror (BSM), over the target using a 7-point pattern. The background is removed by chopping with the BSM and Nodding with the telescope.

- **Parallel Mode (Parallel)**: Used for maps observed with both SPIRE and PACS, working in parallel. These are essentially equivalent to Large Map observations.

- **Point Source Spectroscopy (SOF1)**: Used for point source spectroscopy. The Spectrometer Mechanism (SMEC) mirror is scanned to produce a spectrum over the full wavelength range

- **Small Map Spectroscopy (SOF2)**: Used for creating small spectroscopic maps. The Spectrometer Mechanism (SMEC) mirror is scanned to produce a spectrum over the full wavelength range while the BSM jiggles over either 4 or 16 positions to produce a spectral map.

1.3. SPIRE Observation ID Numbers - converting hexadecimal to decimal

The observer will notice that SPIRE observations ID numbers are interchangably referenced in terms of both hexadecimal and decimal format. This is due to legacy reasons, as originally, the SPIRE ob-
Observation IDs were denoted in hexadecimal only - however, the Herschel Science Archive is now searchable for SPIRE observations in both hex and decimal format, and HIPE also allows input of the observation ID in either format. We try where possible to adopt the decimal format for denoting the Observation ID for the SPIRE observations presented here. If you have a SPIRE observation, and wish to convert from one format to another within HIPE then this can be effected by the following example, where printing an observation ID in Hex reveals the decimal value:

```python
# For an obsid given in Hex
obsidHex=0x50001833L
print obsidHex
# shows the obsid in decimal 1342183475
#
# For an obsid given in Decimal
obsid=1342183475L
print hex(obsid)
# shows the obsid in hex 0x50001833L
```

Alternatively you can use an online tool like the one provided in this link for converting from decimal to hexadecimal and from hexadecimal to decimal.

### 1.4. List of Useful Documentation

The is a enormous amount of documentation associated with Herschel Observations. In below we overview the most important documents.

<table>
<thead>
<tr>
<th>Document Name</th>
<th>Description</th>
<th>Locality</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPIRE Data Reduction Guide</td>
<td>A guide to reprocessing your data</td>
<td>This document</td>
</tr>
<tr>
<td>Quick Start Guide</td>
<td>Gets you started using HIPE with the minimum confusion</td>
<td>Accessed within HIPE Help</td>
</tr>
<tr>
<td>Herschel Data Analysis Guide</td>
<td>Describes all the data analysis and visualization tools available in HIPE, including data access.</td>
<td>Accessed within HIPE Help</td>
</tr>
<tr>
<td>SPIRE Pipeline Specification Manual</td>
<td>A definition of the SPIRE pipeline modules</td>
<td>Accessed within HIPE Help</td>
</tr>
<tr>
<td>Herschel Scripting Guide</td>
<td>Intended for advanced users interested in developing scripts and tools within HIPE.</td>
<td>Accessed within HIPE Help</td>
</tr>
<tr>
<td>SPIRE Pipeline Description Document</td>
<td>High level overview of the pipeline structure, listing every module and the details of its purpose, inputs, outputs and algorithms</td>
<td>SPIRE instrument and calibration page: <a href="http://herschel.esac.esa.int/twiki/bin/view/Public/SpireCalibrationWeb">http://herschel.esac.esa.int/twiki/bin/view/Public/SpireCalibrationWeb</a></td>
</tr>
<tr>
<td>Bolometer Signal Chain and Photometer Pipeline Description</td>
<td>Sets out the detailed bolometer processing and Photometer pipeline algorithms.</td>
<td>SPIRE instrument and calibration page: <a href="http://herschel.esac.esa.int/twiki/bin/view/Public/SpireCalibrationWeb">http://herschel.esac.esa.int/twiki/bin/view/Public/SpireCalibrationWeb</a></td>
</tr>
<tr>
<td>Spectrometer Pipeline Description</td>
<td>This document sets out the Spectrometer pipeline algorithms in detail.</td>
<td>SPIRE instrument and calibration page: <a href="http://herschel.esac.esa.int/twiki/bin/view/Public/SpireCalibrationWeb">http://herschel.esac.esa.int/twiki/bin/view/Public/SpireCalibrationWeb</a></td>
</tr>
<tr>
<td>Herschel Products Definitions Document</td>
<td>A list of all Herschel data products and their data structure.</td>
<td>Accessed within HIPE Help</td>
</tr>
<tr>
<td>Document Name</td>
<td>Description</td>
<td>Locality</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>-------------------------------------------------------</td>
<td>---------------------------------</td>
</tr>
</tbody>
</table>
Chapter 2. SPIRE Launch Pad: Data Reduction Overview

2.1. Data Launch Pad

2.1.1. Obtaining SPIRE data from the archive and importing into HIPE

This topic is covered in detail in SDRG Section 4.1

Herschel data are stored in ESA's Herschel Science Archive (HSA):

- Identified with a unique number known as the Observation ID (ObsID)
- HIPE expects the data in the form of a Pool, so HSA data must be imported into HIPE
- A Pool is like a database, with observations organised as an Observation Context, containing links to all data, calibration and supplementary files

There are three main ways to import data from the HSA into HIPE:

1. Directly from the HIPE command line (SDRG Section 4.1.2) using
   
   ```python
   myobs=getObservation(obsID, useHsa=True),
   or to directly save the data to disk
   myobs=getObservation(obsID, useHsa=True, save=True).
   ```

2. Download a tar file by selecting "Retrieve" in the HSA GUI, unpack it, and then "index" it in HIPE using
   
   ```python
   myObs=getObservation(path="/path/to/dir")
   ```

3. Import directly into HIPE selecting "Send to HIPE" in the HSA GUI. To save this observation from HIPE to a pool on your local disk use,
   
   ```python
   saveObservation(myobs, poolName="myPool", saveCalTree=True).
   ```

Once the data are saved (and indexed) on your hard disk, the Observation Context can be read back into HIPE using,

```python
myobs=getObservation(ObsID)
```

2.1.2. Looking at your Data

This topic is covered in general in SDRG Section 4.2

For Photometer modes see SDRG Section 6.1, and for the Spectrometer see SDRG Section 7.2

Once the data are in HIPE, the Observation Context will be visible in the HIPE variables window:

- Double click to open the "Observation Viewer" (or right click > Open With>Observation Viewer)

Data are available at different levels of processing (SDRG Figure 4.11):

- **Level 0**: Raw data
- **Level 0.5**: Basic processed voltage timeline data
- **Level 1 (Photometer)**: Flux calibrated timelines in Jy/beam
- **Level 1 (Spectrometer)**: Interferograms in V
- **Level 2 (Photometer)**: Flux calibrated maps for signal, error and coverage in Jy/beam
- **Level 2 (Photometer):** Flux calibrated maps for signal, error and coverage in MJy/sr
- **Level 2 (Photometer):** For SSO observations, point source maps in the moving frame for signal, error and coverage in Jy/beam
- **Level 2 (Photometer):** Destriper Diagnostic Products
- **Level 2 (Spectrometer):** Extended calibrated spectrum in W/m²/Hz/sr and point source calibrated spectrum in Jy; or 2D list of spectra and spectral cube in W/m²/Hz/sr
- Other ancillary files are also available (see SDRG Section 4.2.5)
- Data products can be viewed by clicking on the small “+” next to the folders to expand the tree in the same way as a file system. Clicking on one level opens it in an appropriate viewer (right click to choose which). Double clicking opens the viewer in its own tab. The main viewers are:
  - **Context Viewer** – shows the entire context as a tree of products
  - **Dataset Viewer** – shows data as a table array
  - **TablePlotter** – allows one table of data to be plotted
  - **OverPlotter** – allows different tables to be opened in the same plotter
  - **Image Viewer** – displays a 2D image
  - **SDI/SDS Explorer** – shows interferograms/spectra from all Spectrometer detectors

### 2.2. Photometer Launch Pad

#### 2.2.1. Does the Photometer observation data need re-processing?

*Inspecting your maps for possible problems is covered in SDRG Section 6.5.2.*

In many cases, the photometer data is great straight out of the HSA, however, there may be occasions when reprocessing using the User pipeline scripts in HIPE may benefit your observation:

- By using newer improved calibration files: to check the Calibration Tree version see the SDRG Section 5.2.
  
  Updating to a new Calibration Tree is described in the SDRG Section 5.4

- If you see strange artefacts in the maps or bad quality maps in general (SDRG Section 6.5.2)

- By improving already usable maps (e.g. baseline removal, destriping, adding turnaround data, e.g. see SDRG Section 6.8.1)

#### 2.2.2. Re-processing with the User Pipeline Scripts

*This topic is covered in SDRG Section 6.5.1 for the Large Map and Parallel mode and SDRG Section 6.6.1 for the Small Map mode. The mapping pipeline flowchart is shown in SDRG Figure 6.34.*

Simplified User Pipeline Scripts for re-processing SPIRE data are provided in HIPE:

- Accessed directly from the Pipeline>SPIRE menu at the top of the HIPE window
- Separate pipeline script for each observation mode
• Include processing steps taking data from Level 0.5, through Level 1 to the final Level 2 maps
• Final maps are saved as FITS files (not a pool) to any specified directory
• The Observation Context can also be updated and saved to a new Pool (SDRG Section 6.5.1)

Line by line descriptions of the scripts with example plots are shown in SDRG SDRG Section 6.5.1, Section 6.6.1.

The scripts assume the following:
• The data is already on disk stored in a pool on your disk
• The latest Calibration Tree is stored as a pool on your disk (to do this, see SDRG Chapter 5)

To run the User Scripts, 3 options must be set inside the script:
• Observation ID, data pool name, output directory path (See SDRG Section 6.5.1 for other options)

Memory problems encountered during processing extremely large maps can be overcome by using a Temporal Pool (To do this, see SDRG Section 9.2)

From Level 0.5 to Level 1, the pipeline modules that users may with to experiment with are mainly:
• If the observation has strong residual drifts the Recalculate Temperature Drifts module can be tried (see SDRG Section 6.5.2)
• Deglitching algorithms (residual glitches are easily identifiable in the Error Maps)
• In some circumstances the Signal Jump Identifier (see SDRG Section 6.5.2)
• In rare circumstances Low Pass Filter Correction (see SDRG Section 6.5.2)

From Level 1 to Level 2, the pipeline carries out an iterative Destriping process as default in order to remove low frequency noise. If problems are encountered in the final maps, the SPIRE pipeline offers alternatives to the this baseline removal:
• Available from the Scripts > SPIRE Useful Scripts menu.
• Median Baseline Removal (see SDRG Section 6.8.1)
• Polynomial Baseline Removal (see SDRG Section 6.8.1)

Although the default Naïve mapper usually produces good maps, alternative mapping algorithms are also available (See SDRG Section 6.5.1.4).

For observations where producing maps of extended emission are the most important objective, additional Relative Gain Corrections are required to take into account the fact that all bolometers do not have the same uniform beam shape. This is set by an option in the SPIRE user pipeline (SDRG Section 6.5.1.4) The SPIA (SPIRE Photometer Interactive Analysis) package provides a structured GUI-based access to the more intricate parts of the scan map photometer pipeline and consists of tailored functions for I/O, Level 1, and Level 2 processing of SPIRE photometer scan map datasets (SPIA is described in SDRG Chapter 11)

2.2.3. Further Analysis

Source Extraction and Photometry is described in the SDRG Section 6.9.1

The Map Merging topic is described in the SDRG Section 6.11.2

Production of absolute calibrated maps for extended emission using the Planck zero-point correction is explained in the SDRG Section 6.10.1
SPIRE astrometry is covered in the SDRG Section 6.11.3

Treatment of Moving Object Maps is explained in the SDRG Section 6.12.1 and Section 6.12.2

Various Post Pipeline analysis tools are available for SPIRE through HIPE. Users are also encouraged to read the Calibration chapter of the SPIRE Handbook (formerly the SPIRE Observers Manual).

- **Source extraction** from SPIRE maps can be carried out using either the DAOpht or SussExtractor algorithms implemented within HIPE (SDRG Section 6.9.1.4)

- **A Timeline Source Fitter** is available for source photometry from SPIRE timelines (described in SDRG Section 6.9.1.4). The current recommendation is to extract sources with one of the above algorithms and to carry out photometry at these positions using the timeline based fitter.

- **Aperture photometry** can be performed by selecting an image in the Variables view and selecting **annularSkyAperturePhotometry** in the Tasks view. (SDRG Section 6.9.1.6)

- The SDRG describes the recipes for aperture photometry of point sources and diffuse emission.

- **A Map Merging Script** accessible from the Pipeline>SPIRE>Useful scripts menu can be used to merge the Level 1 timeline data from multiple observations (e.g. Parallel Mode, creating mosaics from many maps) to create a single map and output it as a FITS file (Section 6.11.2).

- **An Astrometry Correction script** accessible from the Scripts > SPIRE Useful Scripts menu which allows SPIRE astrometry to be improved by aligning observations with each other, ancillary images or a source list (see SDRG Section 6.11.3).

- **Two Moving Object scripts** accessible from the Scripts > SPIRE Useful Scripts menu are available to correct the maps for observations of moving objects to the moving object frame (see SDRG Section 6.12.1) and to identify the position of a faint moving object target in a map (see SDRG Section 6.12.2).

- **A Bolometer Finder script** accessible from the Scripts > SPIRE Useful Scripts menu allows a User to click on a photometer map and to bring up a plot of the timelines of any bolometer crossing that map pixel. Useful for checking which bolometer timelines are responsible for map artifacts (see SDRG Section 8.3).

- **A Super Resolution Mapper script** accessible from the Scripts > SPIRE Useful Scripts menu allows a User to create super resolution maps (see SDRG Section 6.11.4).

**2.3. Spectrometer Launch Pad**

**2.3.1. Does the observation data need re-processing?**

There are two ways to obtain the most up-to-date processing of Spectrometer data (SDRG Section 7.3.1):

- Using the "on-demand" reprocessing facility of the HSA - this is described in more detail in Chapter 1 of the Herschel Data Analysis Guide.

- Using the User pipeline scripts in HIPE (SDRG Section 7.3.4)

Some further processing of the pipeline results may be required depending on the astronomical source:

- Faint sources - see SDRG Section 7.5
- Semi-extended sources - see SDRG Section 7.6
- Bright sources - see SDRG Section 7.7
• Mapping observations - see SDRG Section 7.8

2.3.2. Re-processing with the User Pipeline Scripts

This topic is covered in SDRG Section 7.3.1

The Spectrometer pipeline flowchart is shown in SDRG Figure 7.15.

Simplified User Pipeline Scripts for re-processing SPIRE data are provided in HIPE:

• Accessed directly from the Pipeline>SPIRE menu at the top of the HIPE window
• Include processing steps taking the data from Level 0.5, through to Level 1 and 2
• Final results are saved as FITS files (not a pool) to any specified directory
• The Observation Context can also be updated and saved to a new pool

Line by line descriptions of the Spectrometer scripts with example plots are shown in SDRG Section 7.3.4

The scripts assume the following:

• The data are already stored in a Pool on your disk
• The latest Calibration Tree is stored as a pool on your disk (to do this, see SDRG Chapter 5)

To run the User Scripts, several options must be set inside the script:

• Observation ID, data pool name, output directory path (see SDRG Section 7.3.4 for other options)

Considerations for reprocessing with the Spectrometer User scripts are:

• **Memory usage** can be reduced by limiting the number of detectors processed in the script
• **Apodization** removes the ringing from the instrumental line function at the cost of reduced spectral resolution. Apodized line profiles can be fitted well by Gaussian functions
• **Second level deglitching** parameters can be modified in the script if there are outlying scans seen in the level-1 spectra (SDRG Section 7.3.1)

Inspecting the data at various stages can provide diagnostic information (see SDRG Section 7.3.1)

General considerations for **faint** and **medium strength** sources are (see SDRG Section 7.5):

• Optimisation of background subtraction
• Check spectral noise with respect to expected HSpot values
• Compare with SPIRE Photometer
• Compare point source and extended calibration

Corrections for **semi-extended** sources are described in SDRG Section 7.6

And for maps of **extended** sources (see SDRG Section 7.8):

• Understanding the SPIRE beam
• Check for clipping in individual interferograms
• Restrict the data made into the cube
• Examine the actual positions on sky observed
• Change the algorithm used for regridding
• Examine coverage and redundancy in the cube
• Maps with faint continuum

2.3.3. Further Analysis

SPIRE spectral analysis is described in SDRG Section 7.11 and cube analysis in SDRG Section 7.12.

Spectral analysis and visualisation tools are provided in HIPE (right click on product in Variables tab > Open With):

• SPIRE specific SDI/SDS Explorer (SDRG Section 8.1) for viewing spectra/interferograms via a clickable footprint of the detector arrays

• Spectrum Explorer (Herschel Data Analysis Guide Chapters 5 and 6) for viewing one, two, and three-dimensional spectral products

• Spectrum Toolbox (Herschel Data Analysis Guide Chapters 5 and 6) for a set of mathematical functions that operate on Herschel spectral data (launched from Spectrum Explorer dialogs menu)

• Spectrum Fitter GUI (Herschel Data Analysis Guide Chapter 7) for interactive line and continuum fitting (launched from Spectrum Explorer dialogs menu)

• Cube Spectrum Analysis Toolbox (Herschel Data Analysis Guide Chapter 6) for spectral cube related tasks (launched from Spectrum Explorer dialogs menu)

Spectrometer useful scripts (Scripts > SPIRE Useful scripts menu at the top of the HIPE window):

• Spectrometer Array Footprint Plot SDRG Section 7.5
• Spectrometer Background Subtraction SDRG Section 7.5
• Spectrometer Line Fitting SDRG Section 7.11
• Spectrometer Thumbnail Mosaic Plot SDRG Section 7.2.3.2
• Spectrometer Convolve Spectrum
• Spectrometer Noise Estimate SDRG Section 7.5
• Spectrometer Cube Fitting SDRG Section 7.12.7

• Combine PACS and SPIRE spectra Section 6.8 of the Herschel Data Analysis Guide

Additional notes for SPIRE Spectrometer data:

• The spectra have been corrected to the Local Standard of Rest only from HIPE v12 onwards - data processed with previous versions were not corrected (see SDRG Section 7.11)

• Lines in high resolution FTS spectra should be fitted with a Sinc profile for unapodized data, and a Gaussian for apodized data. Various tools exist within and outside of HIPE to fit lines to spectra (see SDRG Section 7.11)

• The beam size of the FTS changes across the band of each array in a non-trivial way (see SDRG Section 7.6)

• Each Spectrometer Useful Script contains an example observation ID. This observation will download from the Herschel Science Archive, if HIPE is connected to the internet, so the respective script can be tested out of the box
Chapter 3. Overview of Scripts in HIPE

The main scripts that users will first experience in HIPE are the User Pipeline Scripts accesible through the Pipeline menu in HIPE (see Figure 3.1 below). However, there are additional scripts available and a growing list summary is given below:

User Pipeline Scripts:
The most basic processing pipeline for interactive analysis. Accessible from Pipeline menu in HIPE (see Figure 3.1) or in the build structure through scripts/spire/ia/scripts/reproc/

- **Photometer Two Pass user script**: Photometer_2Pass_Pipeline.py, Two Pass user script
- **Photometer Large Map user script**: Photometer_Large_Map_Pipeline.py, Large map user script
- **Photometer Parallel user script**: Photometer_Parallel_Pipeline.py, Parallel mode user script
- **Photometer Small Map user script**: Photometer_Small_Map_Pipeline.py, Small Map mode user script
- **Photometer Point Source user script**: Photometer_Point_Source_Pipeline.py, Point Source mode user script
- **Spectrometer Point Source user script**: Spectrometer_Point_Pipeline.py, Point Source Spectroscopy user script

Note that in most circumstances, the user pipeline scripts and SPG scripts listed below should reproduce the data exactly as it is in the Herschel Science Archive if no changes are made (assuming that the input, and calibration data match the HIPE version used for the data in the archive).

User Pipeline Scripts: The most basic processing pipeline for interactive analysis. Accessible from Pipeline menu in HIPE (see Figure 3.1) or in the build structure through scripts/spire/ia/scripts/reproc/

- **Photometer Two Pass user script**: Photometer_2Pass_Pipeline.py, Two Pass user script
- **Photometer Large Map user script**: Photometer_Large_Map_Pipeline.py, Large map user script
- **Photometer Parallel user script**: Photometer_Parallel_Pipeline.py, Parallel mode user script
- **Photometer Small Map user script**: Photometer_Small_Map_Pipeline.py, Small Map mode user script
- **Photometer Point Source user script**: Photometer_Point_Source_Pipeline.py, Point Source mode user script
- **Spectrometer Point Source user script**: Spectrometer_Point_Pipeline.py, Point Source Spectroscopy user script
Overview of Scripts in HIPE

- **Spectrometer Mapping user script**: Spectrometer_Mapping_Pipeline.py, Spectroscopy Mapping user script

**SPG Scripts**: The Standard Product Generation (SPG) scripts. Accessible from Pipeline -- SPG scripts menu in HIPE (see Figure 3.1 or in the build structure through scripts/spire/ia/pipeline. These are used for the bulk processing of SPIRE data by the ICC. They are more involved than the user script but do include some advanced functionality such as the use of Temporal Pools.

- **Photometer Large Map pipeline script (POF5)**: POF5_pipeline.py, Large map SPG script
- **Photometer Parallel pipeline script (PARALLEL)**: PARALLEL_pipeline.py, Parallel mode SPG script
- **Photometer Small Map pipeline script (POF10)**: POF10_pipeline.py, Small Map mode SPG script
- **Photometer Point Source pipeline script (POF2)**: POF2_pipeline.py, Point Source mode SPG script
- **Photometer Level 2.5 pipeline script (level25)**: Level25_pipeline.py, Level 2.5 generation SPG script
- **Photometer Level 3 pipeline script (level3)**: Level3_pipeline.py, Level 3 generation SPG script
- **Photometer level25/level3 common functions**: merging_utils.py, Functions used by Level 2.5, Level 3 scripts
- **Spectrometer Point Source pipeline script (SOF1)**: SOF1_pipeline.py, Sparse Spatial Sampling Spectroscopy SPG script
- **Spectrometer Mapping pipeline script (SOF2)**: SOF2_pipeline.py, Intermediate and Full Spatial Sampling Spectroscopy SPG script
- **Engineering pipeline script (eng)**: eng_pipeline.py, Level 0 to Level 0.5 processing SPG script
- **Two Pass pipeline script (2pass)**: twopass_pipeline.py, Two Pass pipeline SPG script

**Additional Scripts**: There are a growing number of additional scripts / tools under development that in the future will be accessible from within HIPE within the Scripts -> SPIRE Useful Scripts menu as in Figure 3.1. Alternatively, they reside within the HIPE Build directory structure, accessible through scripts/spire/ia/scripts/useful/.

- **Photometer Astrometry Correction**: (Photometer_AstrometryCorrection.py). A script to align two or more SPIRE observations, a SPIRE observation with an ancillary image or a source list (described in detail in Section 6.11.3).
- **Photometer Baseline Removal and Destripper**: (Photometer_BaselineRemovalDestripper.py). An example script highlighting the use of the different baseline removal algorithms (median, polynomial) to the User along with the SPIRE Destripper (described in detail in Section 6.8.1).
- **Photometer Bolometer Finder**: (Photometer_BolometerFinder.py). A useful script launching an interactive tool to highlight problems with individual scan lines with the ability to mask scan lines to improve map quality (described in detail in Section 8.3).
- **Photometer Calculate Ephemeris SSO Position**: (Photometer_getSSOposition.py). A script to provide the start and end positions for a moving object in a map (described in detail in Section 6.12.2).
- **Photometer Calibration Bundle point/ext source**: (sourceCalPointExt_usage.py). A script to enable advanced users to produce calibration corrections for beam corrections and colour corrections for a wide variety of input spectra for point or extended sources (described in detail in Section 6.9.1.9).
• **Photometer Calibration Bundle semi-ext source**: (sourceCalSemiExt_usage.py). A script to enable advanced users to produce calibration corrections for beam corrections and colour corrections for a wide variety of input spectra for semi-extended sources (described in detail in Section 6.9.1.9).

• **Photometer Map Merging**: (Photometer_MapMerge.py). A post processing script to merge two or more photometer observations performed in SPIRE Large Map, Small Map or Parallel mode together to produce a single map (described in detail in Section 6.11.2).

• **Photometer Map Zero Point Correction**: (Photometer_ZeroPointCorrection.py). A post processing script to produce absolute calibrated maps using the Planck Maps to discern the map zero point (described in detail in Section 6.10.1).

• **Photometer point Source Photometry**: (Photometer_Photometry.py). A script with examples of making point source extraction and photometry on SPIRE maps (described in detail in Section 6.9.1).

• **Photometer Solar System Object Motion Correction**: (Photometer_SSOMotionCorrection.py). A script to fix observations of moving objects so that the map is made in the moving objects frame (described in detail in Section 6.12.1).

• **Photometer Super Resolution Maps**: (Photometer_HiResMapping.py). An example to set up beam profiles and to run the HiRes superresolution mapper on SPIRE Level 1 data (described in detail in Section 6.11.4).

• **Spectrometer Array Footprint Plot**: (Spectrometer_ArrayFootprintPlot.py). A Script to overplot the SPIRE Spectrometer array outline on top of a SPIRE Photometer map (described in detail in Section 7.5).

• **Spectrometer Background Subtraction**: (Spectrometer_BackgroundSubtraction.py). A Script to show examples of how to improve the background subtraction for faint sources (described in detail in Section 7.5).

• **Spectrometer Line Fitting**: (Spectrometer_LineFitting.py). A Script to fit lines to a SPIRE Spectrometer spectrum (described in detail in Section 7.11.9).

• **Spectrometer Cube Fitting**: (Spectrometer_CubeFitting.py). A Script to fit lines to a SPIRE Spectrometer spectral cube (described in detail in Section 7.12.7).

• **Spectrometer Thumbnail Mosaic Plot**: (Spectrometer_MosaicPlotDemo.py). A script to plot data from many detectors of the SPIRE Spectrometer arrays, with sub-plots that are orientated as the detectors in the array (described in detail in Section 7.2.3.2).

• **Spectrometer Convolve Spectrum**: (Spectrometer_ConvolveSpectrum.py). A script to convolve high spectral resolution data with the SPIRE FTS instrumental response function. For example, this script can be used with HIFI data in order to directly compare it with a SPIRE spectrum. See also Section 7.11.13.

• **Spectrometer Noise Estimate**: (Spectrometer_NoiseEstimate.py). A script to calculate and plot an estimate of spectral noise compared to the expected HSpot values (described in detail in Section 7.5.3).

• **Combine PACS and SPIRE spectra**: (Spectrometer_MergePacsAndSpire_PointSource.py). A script to combine PACS and SPIRE spectra to create a single SED (described in detail in Section 6.8 of the Herschel Data Analysis Guide).

• **Query SPIRE Observing Log**: (SPIRE_ObsLog.py). A script to interrogate the SPIRE Observing log (described in detail in Section 4.1.3).
Chapter 4. SPIRE Observation
Context Data Structure

4.1. Accessing SPIRE Data

4.1.1. Understanding how HIPE accesses data

This section gives a brief description of how to get your data into a HIPE session, from the HSA directly. This section is not an exhaustive guide and users are referred to the HIPE Help documentation on this subject contained within the Quick Start Guide and the Herschel Data Analysis Guide for a fully comprehensive explanation.

After your observation has been made, the data is archived within the Herschel Science Archive (HSA) at ESAC. In order to examine and/or reprocess your observation data, the data itself has to be retrieved from the archive. The architecture and machinery by which HIPE accesses data can seem a little bewildering at first and involves the learning of a new set of vocabulary to refer to data and its storage. Figure 4.1 schematically shows how HIPE accesses data. All data are grouped into convenient containers (like a directory on your computer) known as Pools. However, there are various types of pool depending on their location and accessibility. In Figure 4.1, four important types of Pool are described, although there are other types and a comprehensive description can be found in the Scripting Guide within the HIPE Help Documentation. Very briefly:

- **HSA Pools**: These are data pools stored on the Herschel Science Archive.
- **Local Pools**: These are data pools stored on your own local disk.
- **HTTP Pools**: These are data pools accessible via a URL.
- **MyHSA**: A special read-only pool of HSA data downloads stored on your own local disk.

In order to access any type of pool, HIPE must first initialize a gateway to the pool. This is known as a **Storage** and the code included in this gateway includes everything you need to access and fetch/write data in pools. Using such a gateway to access a pool is referred to as **registering a pool with a storage**. This registration is generally taken care of automatically by HIPE.

![Figure 4.1. How HIPE performs data access](image)

Figure 4.1. How HIPE performs data access
4.1.2. Accessing data directly from the Herschel Science Archive

HIPE provides a relatively straightforward method to access your data directly from the Herschel Science Archive. Within HIPE, select the menu item Window -- Show View -- Data Access -- Herschel Science Archive as in Figure 4.2. This will open a new window in HIPE allowing you to connect to the HSA. You may need to enter your username and password for the HSA in the boxes provided, then press the login button as shown in Figure 4.3. Once logged in, the connection to the HSA is started by pressing the Open HSA User Interface button. This will open the HSA browser in a separate window. Navigating the HSA is beyond the scope of this chapter and the user is referred to the Quick Start Guide and the Data Analysis Guide for details on querying the HSA. In summary, selecting Send to External Application from the HSA window will result in a pointer to your data appearing in the Variable window within HIPE as shown in Figure 4.4. This is the Observation Context and is named following the unique Observation ID (1342183475) for this particular observation. The Observation Context itself, is basically a catalogue of pointers describing the structure and associations within your data. The Observation Context is described in detail in Section 4.2. Note that this is still effectively a pointer to your data on the HSA and no data is stored on your hard disk yet! In order to actually save your data on your own hard disk, i.e. in a Local Pool, you can use the following line of code:

```python
saveProduct(product=obsid_1342183475, pool="MyGalaxyMap", tag="My data")
```

The first parameter is the Observation Context, in this case, as it appeared in the Variables panel in Figure 4.4. The second parameter is the name of the Pool (i.e. the name you have chosen for the directory on disk where the data will be saved). The third parameter is a tag to differentiate reprocessed versions from one another when browsing data stored on your hard disk.
If you do not want to use the GUI access to the HSA and you already know the observation ID, then the same process can optionally be performed from the command line using the command:

```python
myObsID=1342183475L
myobs=getObservation(myObsID, useHsa=True)
```

Note that for this command to work, HIPE expects you to have your HSA username and password written in your `user.props` file (usually located in your home directory under `.hcss/user.props`). If these lines are there, they should look like;

```ini
hcss.ia.pal.pool.hsa.haio.login_usr = your_username
hcss.ia.pal.pool.hsa.haio.login_pwd = your_password
```

Note that you may also have a `hipe.props` in the same location which will supercede any settings in `user.props` so care should be taken (See the "Properties and Preferences" section in the HIPE Owners Guide). If these are not in your `user.props` file then they can be initiated in your current session by;
loginUserProperty = "hcsl.ia.pal.pool.hsa.hai.login_usr"
loginPasswordProperty = "hcsl.ia.pal.pool.hsa.hai.login_pwd"
Configuration.setProperty(loginUserProperty, "your_username")
Configuration.setProperty(loginPasswordProperty, "your_password")

Finally you must of course save your observation to disk using the
saveProduct(myObsContext, pool=myPool, tag='My tag') command described
earlier in this section.

4.1.3. Querying the SPIRE Observation Log

The SPIRE Observation Log is a file containing details all SPIRE observations made during the
Herschel mission lifetime including obsid, target, mode, position and all instrument and AOR parameters.
The SPIRE Observation Log is included within the HIPE build and can be queried and interrogated via
a specialised script: Query SPIRE Observation Log, accessed from the Useful Scripts
menu in HIPE as shown in Figure 3.1. The SPIRE Observation Log is queried by creating an instance
of the log, e.g. obsLog = SpireObsLogProduct().

The log can be queried on many different parameters but perhaps the simplest is the obsid as shown
in the example taken from the script below. Using the obsid 1342201139, the log returns the entry
for this obsid in the obsidProps variable that can either be accessed from the variables
pane in the HIPE window, printed as in the script below, or interrogated for individual parameters such
as RA and Dec.

#Load Full SPIRE Observation Log (obsLog)
obsLog = SpireObsLogProduct()
# -------------------------------------------------------------
#Get Properties of a single Obsid
obsid = 1342201139
obsidProps = obsLog.getObsidInfo(obsid)
# Print source position
print "RA, Dec: ",obsidProps['ra'], ['dec']
print "Target: ", obsidProps['source']
# Print full contents of log for this obsid
print "SPIRE Observation log entry for obsid %i:" %(obsid)
for key in obsidProps.keySet():
 print "%10s: %s"%(key, obsidProps[key])
# -------------------------------------------------------------

In addition to a single obsid, the observation log can also be queried on other parameters such as a
source name as shown in the code example below. The log is queried on the source ngc1275 and the
results stored in the variable subObsLog1. Right-clicking on subObsLog1 in the variables
pane in the HIPE window shows all the observations retrieved using this search parameter as shown in
Figure 4.5. Alternatively, specific results for items, e.g. obsid or observation mode can be retrieved
by selecting individual columns in the information retrieved from the log as shown in the code snippet
below.

#Load Full SPIRE Observation Log (obsLog)
obsLog = SpireObsLogProduct()
# -------------------------------------------------------------
subObsLog1 = obsLog.filterBySourceName('ngc1275')
# List all obsids recovered from obsLog
obsidList = subObsLog1['obsLog']['obsId'].data
print "List of obsids from query:
print obsidList
print
# List all observation modes recovered from obsLog
obsModeList = subObsLog1['obsLog']['obsMode'].data
print "List of observing modes from query:"
HIPE> [1342203614,1342249054,1342249055]
HIPE> ["SpirePhotoSmallScan","SpireSpectroPoint","SpireSpectroPoint"]
# -----------------------------------------------------------
print obsModeList

Figure 4.5. Query the SPIRE Observation Log on source name.

The log can also be searched on position (RA, Dec) using a search cone as shown in the code example below. The log is searched at position RA=187.2 degrees and Dec=12.0 degrees within a search radius of 2 degrees. The log returns all observations whose centres are within a 2 degree radius of the search position.

#Load Full SPIRE Observation Log (obsLog)
obsLog = SpireObsLogProduct()
# -----------------------------------------------------------
# Filter by radius and print sources found
ra = 187.2  # position RA (deg)
dec = 12.0  # position dec (deg)
radius = 2.0  # search radius (deg)
subObsLog2 = obsLog.filterByRadius(ra,dec,radius)
sourceList = subObsLog2["obsLog"]["source"].data
print "Targets at RA=%.3f deg, dec=%.3f deg in search radius %.3f deg:*%("%(ra,dec,radius)
print sourceList
# -----------------------------------------------------------
HIPE>
HIPE>
HIPE> Targets at RA=187.200 deg, dec=12.000 deg in search radius 2.000 deg:
HIPE> ["V2","V2","M86","NGC4435/4438-1","V2","V2","NGC4459","V2","V2","NGC4388","V2","V2","NGC4325","IC 3467","NGC4459","NGC4388"]

There are also methods available to query the log on specific observation modes. For example, to select all sparse sampled spectrometer observations, the code example below can be used.

#Load Full SPIRE Observation Log (obsLog)
obsLog = SpireObsLogProduct()
# -----------------------------------------------------------
# Filter by observation mode
# Select sparse sampled spectrometer observations
subObsLog3 = obsLog.filterSpecSparse()
# Print first obsid in the list
print "First obsid in list = ",subObsLog3["obsLog"]["obsId"][0]
# -----------------------------------------------------------
HIPE>
HIPE>
HIPE>First obsid in list = 1342183470L
4.2. SPIRE Observation Context Data Structure

4.2.1. Anatomy of a SPIRE Observation: Products, Pools, Storage, and Building Blocks

For the purposes of both this chapter and the next (on reprocessing your data), we assume that you have already downloaded a data set from the Herschel Science Archive and are familiar with how to put your data into a store and how to access your data from this store within HIPE. If you haven't, please look at the HIPE Quick Start Guide and the HIPE Owner's Guide for instruction on how to do this.

Now you are the proud owner of a set of SPIRE observations. Before carrying out any processing it's most likely that you will want to have a first look at your data. SPIRE observations are supplied in a highly organized structure that may be unfamiliar compared to previous astronomical datasets you have encountered. In addition, with this new structure comes a host of new terminology which will be introduced and described in this section.

All data within the HCSS processing system are passed around in containers referred to as **Products**. There are Products for every kind of data, e.g.:

- Raw and processed Detector Data Timelines
- Calibration Data
- Auxiliary (e.g. Pointing) Data
- Images
- Image Cubes
- Data Contexts
- ............ etc

Products can contain the following (pictorially visualized in Figure 4.6):

- Meta Data
- One or more Datasets
- Processing History

**Datasets** can be:

- Array Tables
- Image arrays
- Composite (nested) Tables
- ............ etc
Figure 4.6. General structure of a SPIRE data Product

SPIRE and indeed all Herschel observations are accessed/downloaded and stored as a Pool of these products. A Pool is basically a directory that contains the original raw data, the results of the automatic pipeline processing and everything you need to process your observations again yourself (e.g. spacecraft pointing, the parameters you entered in HSport when you submitted the proposal, and the pipeline calibration tables). Looking inside a Pool, the individual FITS files associated with the data can be seen and can in fact be transported/saved individually; however, the entire observation dataset itself is structured in a very specific way with various associations between individual files, therefore until the data is in its final form it is best not to poke around too much within a pool. Data that you reprocess yourself can also be stored into the same Pool or you may alternatively wish to save the results in a new Pool. If you wish to send someone a set of processed data for example, the entire Pool directory should be “tar”ed or archived and sent. Finally, once a Pool has been created, the pool's directory name must NOT be changed or HIPE will not be able to find the data.

In general, HIPE expects all your observation pool directories to be contained in a "Local Store" or more accurately a "Local Pools location" directory which can be thought of as a Super Repository for all Observation Pools on your hard disk. By default this directory resides in ~/.hcss/lstore but can be changed and renamed by editing the HCSS user.props file. The structure of the Local Store is visualized in Figure 4.7.

Figure 4.7. General structure of the Local Store

4.2.2. Linking it altogether: Introducing the Context

The smallest “piece” of SPIRE observational data is called a Building Block. These Building Blocks correspond to basic operations within an observation and as the name suggests every SPIRE AOT is built up from a combination of these building blocks. Building Blocks are usually in the form of Timeline Data Products.

Example building blocks may be:
• A scan line in a map
• A single 7 point Jiggle
• A set of Spectrometer scans
• A segment of housekeeping scans
• A motion of the Beam Steering Mirror (BSM)

Building Blocks and other Products are grouped into a context. A context is a special kind of product linking other products in a coherent description and can be thought of as an inventory or catalogue of products. The SPIRE processed observation consists of many such contexts within one giant Observation context. Therefore, each set of building blocks have a context. Each Processing Level in the SPIRE pipeline has a context and the entire Observation has a context. Thus a complete observation may be thought of as a big SPIRE onion as depicted in Figure 4.8. Moreover, contexts are not just for building block products and higher processed data products, there are contexts for Calibration Products and contexts for Auxiliary Products (e.g. pointing) and even a context for Quality Control. The entire SPIRE Observational Context is shown in Figure 4.9 for all products from the raw building block data to the final high level processed end products from the pipeline. This is the structure and content that you should receive for your SPIRE observation from the Herschel Science Archive (HSA).
Figure 4.8. The Context structure within HCSS. The smallest “piece” of SPIRE observational data are Building Blocks. Building Blocks and other Products are grouped into a context. All the data within an entire SPIRE observation are linked by an Observation Context.
As of HIPE version 11., SPIRE observations may include a new Level 2.5 (See Section 6.1.3). The Level 2.5 products are nominally created from pairs of single parallel scan observations in the nominal and orthogonal directions with no multiple points included.

In addition to the parallel pairs, other SPIRE observations that would benefit from processing to Level 2.5 have also been included in the level 2.5 processing pipeline. Therefore the full set of Level 2.5 products may be created from:

- Nominal and orthogonal parallel pairs.
- Groups of 2 or more (largest set of 21) observations in parallel mode that are not nominally parallel pairs. An example is shown in Figure 4.10, created from 3 individual parallel maps.
- Overlapping observations from the same program made in Large Map single scan mode.

The obsid's of all observations with non-standard Level 2.5 products is listed below in (See Table 4.1):
<table>
<thead>
<tr>
<th>mode</th>
<th>obsid list</th>
</tr>
</thead>
<tbody>
<tr>
<td>parallel</td>
<td>1342184470; 1342184471; 1342184472; 1342184473</td>
</tr>
<tr>
<td>parallel</td>
<td>1342184486; 1342184487; 1342184488</td>
</tr>
<tr>
<td>parallel</td>
<td>1342188084; 1342188085; 1342188086; 1342188087</td>
</tr>
<tr>
<td>parallel</td>
<td>1342188650; 1342188651; 1342188681; 1342188682; 1342189108</td>
</tr>
<tr>
<td>parallel</td>
<td>1342189004; 1342189031; 1342190324; 1342190325</td>
</tr>
<tr>
<td>parallel</td>
<td>1342190616; 1342202090; 1342202254</td>
</tr>
<tr>
<td>parallel</td>
<td>1342195729; 1342195743; 1342196656; 1342220649; 1342220833; 1342220880</td>
</tr>
<tr>
<td>parallel</td>
<td>1342196626; 1342220878; 1342220879; 1342245911</td>
</tr>
<tr>
<td>parallel</td>
<td>1342198565; 1342198566; 1342198590; 1342205054; 1342205055; 1342205092</td>
</tr>
<tr>
<td>parallel</td>
<td>1342205558; 1342210559; 1342210917; 1342210946</td>
</tr>
<tr>
<td>parallel</td>
<td>1342210567; 1342210903; 1342222626; 1342222676</td>
</tr>
<tr>
<td>parallel</td>
<td>1342210568; 1342210918; 1342210932; 1342210947</td>
</tr>
<tr>
<td>parallel</td>
<td>1342210902; 1342210931; 1342211292; 1342222677</td>
</tr>
<tr>
<td>parallel</td>
<td>1342219629; 1342231849; 1342231850; 1342232059</td>
</tr>
<tr>
<td>parallel</td>
<td>1342219811; 1342220534; 1342220534; 1342223237</td>
</tr>
<tr>
<td>parallel</td>
<td>1342219951; 1342219978; 1342220618; 1342220648</td>
</tr>
<tr>
<td>parallel</td>
<td>1342221464; 1342221465; 1342221474; 1342221475</td>
</tr>
<tr>
<td>parallel</td>
<td>1342221904; 1342221922; 1342222113; 1342222131</td>
</tr>
<tr>
<td>parallel</td>
<td>1342221905; 1342221921; 1342222112; 1342222130</td>
</tr>
<tr>
<td>parallel</td>
<td>1342231854; 1342231856</td>
</tr>
<tr>
<td>parallel</td>
<td>1342190310; 1342190311</td>
</tr>
<tr>
<td>parallel</td>
<td>1342231855; 1342231857</td>
</tr>
<tr>
<td>parallel</td>
<td>11342224989; 1342224990</td>
</tr>
<tr>
<td>Large Single</td>
<td>1342234749; 1342236232; 1342236234; 1342236240; 1342237550; 1342237553</td>
</tr>
<tr>
<td>Large Single</td>
<td>1342237563; 1342238251; 1342246690; 1342246632; 1342247216; 1342257562</td>
</tr>
<tr>
<td>Large Single</td>
<td>1342232364; 1342232365; 1342245412; 1342245431; 1342245432; 1342245510</td>
</tr>
<tr>
<td>Large Single</td>
<td>1342245511; 1342245547</td>
</tr>
<tr>
<td>Large Single</td>
<td>1342247220; 1342247993; 1342247994; 1342247995; 1342247996; 1342247997</td>
</tr>
<tr>
<td>Large Single</td>
<td>1342247998; 1342248000; 1342248001; 1342248491; 1342248492; 1342248493</td>
</tr>
<tr>
<td>Large Single</td>
<td>1342248494; 1342248495; 1342248496; 1342248497; 1342248498; 1342248499</td>
</tr>
<tr>
<td>Large Single</td>
<td>1342248500; 1342249103; 1342249105</td>
</tr>
</tbody>
</table>
Figure 4.10. Level 2.5 map created from 3 individual observations

Maps are produced using the standard pipeline (See Section 6.5.1 and all the photometer known issues applies to these maps as well and if any artifacts such as undetected glitches, temperature drifts or detectors jumps are present the data may need to be re-reduced with the tips suggested in Section 6.5.2 using the standard pipeline in Section 6.5.1 and the map merging script described in Section 6.11.2). The pairs of observations used to build the Level 2.5 maps are included in the observations' metadata. The Level 2.5 products contain:

- Point Source calibrated maps.
- Extended emission absolute calibrated maps.
- The associated destriper diagnostic products.

4.2.4. The Level 3. Data in the Herschel Science Archive

As of HIPE version 11, SPIRE observations may also include a new Level 3.0 (See Section 6.1.3). The Level 3.0 products are mosaiced maps obtained merging all contiguous observations irrespective of the original observing program. The criteria for the Level 3 map creation are as follows:

- Mosaics of all overlapping observations, as traversed from the starting observation. There are no gaps.
- Produced from the [Planck] zero-point calibrated maps and in units of surface brightness (MJy/sr)
- Created using the Mosaic Task within HIPE to combine the existing Level-2 extended emission maps. Where Level 2.5 exists, the Level 3 products are created from the Level 2.5 maps instead.
- WCS is optimally calculated to minimize the map size
- For Level 3 maps < 10 degrees in size a TAN (Gnomonic) map projection is used.
- For Level 3 maps with a distance from the centre of > 10 degrees an ARC (Zenithal equidistant) map projection is used (i.e. the Galactic Plane).
• The Level 3 selection criteria leads to logical groupings of reasonable size except in a few cases, most notably, the Galactic Plane. These very large groups are broken up into reasonable groupings by hand. For instance the Galactic Plane is cut into chunks of approximately 15 degrees in length, with some adjustments around the Galactic Center. Note that there is no overlap between Level 3 map slices.

The list of observations used to build the Level 3 maps are included in the observations’ metadata.

4.2.5. Looking at your Observation Context in HIPE

The Observation Context can be viewed directly within HIPE. It is assumed in this example that the data for the observation with Observation ID 0x50001833 (in hexadecimal or 1342183475 in decimal) has already been downloaded from the archive and has already been stored in a pool named MyGalaxyMap in the Local Store (A pool can take any name and MyGalaxyMap is just an example). We therefore have to search for this pool and extract the Observation Context for this observation. This is possible via a slightly convoluted route using the GUI but can also be accomplished painlessly with the few lines of code shown below (or indeed a single getObservation command);

```python
obsid=0x50001833
pool='MyGalaxyMap'
myObsContext=getObservation(obsid,poolName=pool)
```

The first line of code selects the desired Observation ID while the second line defines the Data Pool within Local Store where the observation is stored on the hard disk. The getObservation command reads the Observation Context into the variable myObsContext. Note that the HSA can be queried directly using the parameter useHsa=True (default is False). Furthermore, if it is the case that the Data Pool for the observation is NOT in the Local Store, then one may point to such an exterior Pool with the option poolLocation=<directory>. After running the getObservation command, we see 2 new entries in the Variables window-pane of HIPE shown in Figure 4.11 (Note: the p is simply a place holder). Double clicking on the obsContext (or selecting Context Viewer) from the Open with menu in the variable list brings up the Observation Context in a new window as also shown in Figure 4.11. The Observation Context has Summary, Meta-Data and Data panes. The Summary pane contains information on the instrument, target position, observation ID, Operational Day and Observation Mode. The Meta-Data pane contains all relevant information on the Product necessary to describe and process the observation (including the information in the Summary pane). The Meta-Data for the observation context is summarized in Table 4.2. The Observation Context Data pane contains pointers to all other contexts and data products contained in the Data Pool. The Data pane contains many entries, listed below and in Figure 4.12 (See also Figure 4.9);

• level 0: The Level 0 context containing links to the Level 0 raw Data before any pipeline processing.

• level 0.5: The Level 0.5 context containing links to the Level 0.5 data products after the common engineering conversion has been made.

• level 1: The Level 1 context containing links to the Level 1 data products after AOT specific pipeline processing.

• level 2: The Level 2 context containing links to the final Level 2 data products from the pipeline.

• level 2.5: The Level 2.5 context containing links to the final Level 2.5 data products (if present) from the pipeline.

• level 3: The Level 3 context containing links to the final Level 3 data products (if present) from the pipeline.

• calibration: The Calibration context pointing to all calibration products required for the processing of SPIRE data.
• auxiliary: The context pointing to all auxiliary files required for processing.

• logObsContext: The context pointing to the reduction log that records the processing history of the data.

• quality: The Quality context pointing to the quality control products for this observation.

• browseImageProduct: The context pointing to thumbnail products.

• browseProduct: The context containing information from the HSA archive.

Note that the structure of the Observation Context can also be directly seen from the command line by typing `print MyObsContext;`

```python
HIPE> print myObsContext
{description="Unknown", meta=[type, creator, creationDate, description, instrument, modelName, startDate, endDate, obsState, obsid, odNumber, cusMode, instMode], datasets=[], history=None, refs=[auxiliary, browseImageProduct, browseProduct, calibration, level0, level0_5, level1, level2, logObsContext, quality]}
```

Here the Observation Context can be clearly seen to contain no data as such but rather a set of pointers or references to other different kinds of contexts. In the forthcoming chapters, the Observation Contexts for specific individual AOTs will be investigated in more detail allowing us to have a first look at our processed data!

### Table 4.2. Description of Meta Data in the SPIRE Observation Context

<table>
<thead>
<tr>
<th>Meta Data</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>odNumber</td>
<td>The Observational Day when the observation was made</td>
</tr>
<tr>
<td>obsid</td>
<td>The unique Observation ID (in decimal)</td>
</tr>
<tr>
<td>startDate</td>
<td>The start date of the observation in TAI, UTC format</td>
</tr>
<tr>
<td>endDate</td>
<td>The end date of the observation</td>
</tr>
<tr>
<td>creationDate</td>
<td>The creation date of this Product</td>
</tr>
<tr>
<td>creator</td>
<td>How the product was created (e.g. Standard Product Generation (SPG) version)</td>
</tr>
<tr>
<td>modelName</td>
<td>Whether the data is from Flight or Flight Spare, etc</td>
</tr>
<tr>
<td>obsState</td>
<td>How far has the observation been processed by the pipeline (Level 0, 0.5, 1 or 2)</td>
</tr>
<tr>
<td>type</td>
<td>The Product Type (OBS = Observation Context)</td>
</tr>
<tr>
<td>instMode</td>
<td>The instrument mode (The AOTs defined internally as POF5 for Large Map Mode)</td>
</tr>
<tr>
<td>instrument</td>
<td>The instrument name, in this case SPIRE</td>
</tr>
<tr>
<td>cusMode</td>
<td>How the AOT is referred to in the observation logs and scheduling</td>
</tr>
<tr>
<td>description</td>
<td>The Product name</td>
</tr>
</tbody>
</table>
Figure 4.11. The Observation Context within HIPE
4.2.6. What the Observation Context looks like on your hard disk

It is interesting to see what the Observation Context physically is externally of HIPE as a physical file structure. Since the HIPE processing environment is constructed using "Object Orientated Programming" (OOP) by design, every data product or context is contained within a "(Java) class". The convention for determining the organization of the directory structure is that each product or context is saved in a separate directory named after its class (see the Scripting Guide for more details on classes). This leads to the complicated directory list shown in Figure 4.13.

In reality the Observation Context is a FITS file found within the Pool folder for this observation, usually inside your designated Local Store area. In the example considered in this chapter the Observation Context is found on the following path /localstore/MyGalaxyMap/herschel.ia.obs.ObservationContext (where the pool name is "MyGalaxyMap").

In Figure 4.13 the Observation Context is a FITS file called hspirel342183475obs.fits (highlighted in yellow) which has been opened in step (1) by some external FITS viewing application (e.g. fv). The file hspirel342183475obs.fits consists of a FITS header and a table containing a list on contexts. Thus the context contains no data as such but rather a set of pointers or references to other different kinds of contexts. The list has 2 columns containing the name of the additional contexts and their Unique Reference Number (urn with a number index 0). Looking at the entry for the Level2 context we see that this is a MapContext (outlined in blue) and following step (2) this leads us to a folder herschel.ia.pal.MapContext containing another FITS file in the pool which describes the Level 2 scan map image products (hmapcontext.fits in green). Again opening this hmapcontext.fits FITS file in your favourite external viewer (shown in step (3)) shows a link to the three SPIRE map images contained within herschel.ia.dataset.image.SimpleImage.
Following the link (step (4) in orange) leads us to the folder on the disk containing the 3 image files (**mp**.fits, one each for PSW, PMW, PLW), which can be opened as normal (multi-extension) FITS files in external applications such as DS9, etc (see Section 6.1.2).

The SPIRE pipeline scripts that are run at the Herschel Science Centre to produce the results saved in the HSA are referred to as the Standard Product Generation (SPG) scripts. These are included in the HIPE installation for information (see Chapter 3). Running these scripts will reproduce exactly the data that is in the HSA (assuming that the HIPE versions match). The version that was used to process an observation is recorded in its metadata item "creator":

```
print obs.creator
```

This metadata item is set inside all products and contexts within the Observation Context. The value of "creator" is, for example,

```
SPG v9.1.0
```

and refers to the HIPE version used. It can also be seen in the HSA search results table in the "SPG version" column.

In addition to the version of the pipeline script, it is also important to know the version of the calibration tree that was used. The calibration tree version used is recorded in the final product metadata in "calVersion" and "engConvCalVersion" - see Section 5.8.

The SPG scripts are available in HIPE from the "Pipelines" button on the top menu bar. The SPG scripts are designed for automatic data processing at the HSC, but can also be run inside HIPE, starting from an Observation Context saved on disk. Open the script and run it - it will open pop-up windows asking for the data location. The scripts are complicated by the necessity of taking account of every possible observing mode. For this reason, simplified "User Pipeline Scripts" are also provided (see...
These should produce exactly the same results as the SPG scripts, but require some user input. These User Pipeline Scripts are recommended as the best way to reprocess data.

The User Pipeline Scripts for reprocessing data are described in detail for each observing mode in Chapter 6 and Chapter 7.
Chapter 5. SPIRE Calibration Data

5.1. The SPIRE Calibration Context

Calibration data is attached to the Observation Context for every observation. This section describes how to access, understand and update (if necessary) the calibration data. The calibration context, which contains the SPIRE calibration products for either the Photometer or Spectrometer can be extracted from the Observation Context as follows (where the observation context has already been read into a variable called `obs`):

```python
cal = obs.calibration
```

The view when this is visualised in the Observation or Context Viewer is shown in Figure 5.1. This viewer shows that there is either a sub-context for the Photometer (attached to Photometer observations), or for the Spectrometer (attached to Spectrometer observations), as well as some products that are common and so listed separately. The individual calibration products are contained within the "phot" and "spec" calibration contexts.

Figure 5.1. The SPIRE calibration context.
5.2. The SPIRE Calibration Tree

The calibration of the SPIRE instrument is likely to be improved throughout the mission and beyond as we gain better understanding of the instrument performance. The collection of all calibration products for SPIRE are referred to as the "Calibration Tree", and as this is updated, the calibration tree number changes. The version of the calibration tree is contained within the metadata of the calibration context, for example:

```python
print obs.calibration.version
```

Calibration trees are often (but not always) related to a particular version of HIPE.

The calibration tree contains both general calibration products that are common to all observations, and also specific products that refer only to the observation in question. The general products describe the detectors, instrument and calibration. They can have contents that vary for different conditions (as described in the next section), but these different "editions" are all attached to every observation. The specific calibration products, on the other hand, contain history information for a particular observation and so have different values for different observations. Examples of specific calibration products are the time reset history, the detector offset history, and the PCAL product.

5.3. SPIRE Calibration Product Editions

Some calibration products have several versions with different contents depending on the conditions of the observation (for example, the values may change at different times, or may depend on whether "bright" or "nominal" mode was used, etc.). These are referred to as "editions". The Calibration Context Viewer lists the dependency of the editions next to each calibration product, and gives access to all of the different editions (shown as an example in Figure 5.2).

![Figure 5.2. SPIRE calibration editions.](image)

In a script, in simple cases (such as time dependency), the Observation Context can select the correct edition automatically. In other cases, the variables upon which the product depends must be supplied to get the correct product from a List.
Some examples of accessing individual products in a script from the phot and spec contexts are:

- Spectrometer band edges product, which has no dependency,
  
  ```python
  bandEdge = obs.calibration.spec.bandEdge
  ```

- Photometer channel mask product (details which detectors are defined as dead, or noisy) - the correct time dependent edition for this observation is selected automatically,
  
  ```python
  chanMask = obs.calibration.phot.chanMask
  ```

- Flux Conversion products are selected automatically for the Photometer (dependency is on whether nominal or bright mode was used and the observing date), but for the Spectrometer, where the product also depends on jiggle position, spectral resolution and apodization function, the correct product must be selected from a List,
  
  ```python
  photFluxConv = obs.calibration.phot.fluxConv
  specTeleRsrf = obs.calibration.spec.teleRsrfList.getProduct(6, "HR", obs.startDate)
  ```

When the SPIRE calibration products are saved in FITS file format, the naming convention for the individual product edition files is derived from “SCal” (for SPIRE Calibration), plus "Phot" or "Spec" (for Photometer or Spectrometer), the name of the product, the dependencies (if there are any), and the version number of that particular edition. For example:

```plaintext
SCalPhotBolPar_v3.fits
SCalSpecBeamParam_HR_unapod_nominal_20050222_v1.fits
```

Time dependency is specified in the file name by the start date at which the edition becomes valid.

### 5.4. Updating a Calibration Tree

When an observation is processed by the HSC and placed into the Herschel Science Archive, it has the particular calibration tree of the time attached (and used in the automatic pipeline). It is possible to update this calibration tree that is attached to the observation, either to a more recent version, or to a previous version (e.g. to determine the effect of an update in calibration products).

From HIPE v12 onwards, notification of updates to the calibration tree are provided by the **Spire Calibration Automatic Updater**, which is started automatically in the background when HIPE is started. If a new calibration tree is available, the updater GUI shows the new version, and relevant information about the release is available by clicking on "Show details" (see Figure 5.3).
Figure 5.3. The SPIRE Calibration Automatic Updater GUI.

SPIRE calibration file set v11.0 incorporates the following differences from the previous community release, v10.1:

**Spectrometer:**

- **SCalSpecTeleRsrF:**
  - Significant increase in signal-to-noise
  - Remove bright mode files (one set of products used for both bright and nominal modes)
  - Remove dependency on biasMode and apodName

- **SCalSpecInstRsrF:**
  - Significant increase in signal-to-noise
  - Remove bright mode files (one set of products used for both bright and nominal modes)
  - Remove dependency on biasMode and apodName

- **SCalSpecBeamParam:**
  - Update to match new RSRFs
  - Update beam FWHM (some small differences from previous version)
If the latest calibration tree has already been downloaded but is not set as default tree in the HIPE session (defined by the `spire.cal.pool` property), a popup will appear, giving the option to set it as the default, as shown in Figure 5.4.

![Figure 5.4. The SPIRE Calibration Automatic Updater popup, asking for confirmation of whether to set a downloaded cal tree as the default.](image)

Figure 5.4. The SPIRE Calibration Automatic Updater popup, asking for confirmation of whether to set a downloaded cal tree as the default.

The updater GUI can be started manually at any time from the Tools > spire-cal menu in HIPE. The updater can be disabled from automatically running at HIPE start up by deselecting the appropriate option in the Spire Calibration preference panel (available from Edit>Preferences>Data Access>Spire Calibration - see also Section 5.5).

The list of available calibration trees in the HSA can be determined using the SPIRE Calibration View GUI (accessed from Window > Show View > Calibration > SPIRE Calibration), which requires a internet connection. Any specific calibration tree can then be requested from the HSA and installed on disk using this GUI - see Section 5.5 for more details.

Alternatively, browsing and installing the available calibration trees can be carried out on the command line (this requires a connection to the HSA) by typing,

```python
print spireCal.trees
```

which returns a list, where the highest number is the most recent version. A specific calibration tree can downloaded and installed using the spireCal task, passing the version string as input to the task (or by simply typing "spire_cal" to get the latest version). For example,

```python
calNew = spireCal(calTree="spire_cal")
```

This will pop up a dialog box asking for user login and password for the HSA if not already logged in. The command will load references to the calibration tree products, so that the actual files are only downloaded when they are accessed. An extra keyword can be added so that the calibration tree is saved directly to a local pool on the disk:

```python
cal = spireCal(calTree="spire_cal", saveTree=1, saveLocation="/directory/"
```

where the `saveLocation` keyword is optional (it defaults to the local store directory). The name of the pool is always set automatically to the calibration tree version. Note that this command may take a long time to run because it has to download all of the data from the HSA before saving to disk.
When the calibration tree is saved to the disk as a pool using the command above, it updates a HIPE property to save the calibration version. This property is called `spire.cal.pool`. Before anything is done, in a fresh HIPE installation, this property is set to the latest calibration tree matching the HIPE version.

Once the new calibration tree is installed in a pool on the local disk, the following line could be added to any reprocessing scripts to read it back into HIPE (in this example, specifying explicitly the calibration tree version),

```
calNew = spireCal(pool="spire_cal_14_1")
```

If nothing is specified to the `spireCal` task, it will use the value of the `spire.cal.pool` property to try to read the default pool from disk:

```
calNew = spireCal()
```

The calibration tree can then be updated in the Observation Context to replace the existing calibration tree using,

```
obs.calibration.update(calNew)
```

Note that this will only work if the Observation Context has an existing calibration tree. It is possible to save the Observation Context without any calibration data included at all (e.g. if saved using `saveObservation()` without `saveCalTree=True`), and in this case, the previous command will crash with an error that says 'NoneType' object has no attribute 'update'. The calibration tree must then be **added** (rather than updated) by running,

```
obs.calibration = calNew
```

There are some drawbacks of **adding** rather than **updating** the calibration tree: this means that only the general calibration products are added, and the specific calibration history products for the observation in question are not present. These specific products include the time reset history and detector offset history (needed to run the engineering conversion), and the PCAL results. If these calibration history products are required, it may be necessary to re-download the observation from the Herschel Science Archive and then **update** (rather than **add**) the new calibration tree.

To permanently save the updated Calibration Context, the Observation Context would then need to be written out to a pool (or saved back into the pool from which it was read from) on the local disk,

```
saveProduct(obs, pool="myPool", tag='Calibration updated')
```

A full description of the available versions is given on the Herschel website: [http://herschel.esac.esa.int/twiki/bin/view/Public/SpireCalibrationWeb](http://herschel.esac.esa.int/twiki/bin/view/Public/SpireCalibrationWeb).

### 5.5. The SPIRE Calibration Task GUI and Calibration Viewer

There are two GUI views in HIPE that can be used to load and inspect different calibration trees.

Calibration trees can be read into HIPE and saved to the local disk using the **SPIRE Calibration Task GUI**. This GUI can be accessed from the Tasks panel in HIPE in the list of SPIRE tasks - it is called "spireCal". The GUI is shown in Figure 5.5. By default, there is no calibration tree specified in the GUI, and if "accept" is pressed, it will try to read the calibration tree version contained in the `spire.cal.pool` property (described in the previous section) from the local disk.

There are three options in the GUI to enter the calibration tree version string: `calTree` will fetch the tree from the HSA; `pool` will read from the local disk; and `jarFile` will read from a jar file on the local disk. The calibration context can also be saved to disk by ticking the "saveTree" box.
Figure 5.5. The SPIRE calibration task GUI.

The second GUI is the **SPIRE Calibration View**, which lists all of the available SPIRE calibration trees that are in the HSA, in the ICC archive, or installed on your system. It shows the release note for each calibration tree, and this gives details of what was updated in the different tree versions. The view can be opened from the HIPE menu item **Window > Show View > Calibration > SPIRE Calibration** – see Figure 5.6.
Community Release Note for SPIRE Calibration 11.0

SPIRE calibration file set v11.0 incorporates the following differences from the previous community release, v10.1:

**Spectrometer:**
- **SCLSpecTeleRsrp:**
  - Significant increase in signal-to-noise
  - Remove bright mode files (one set of products used for both bright and nominal modes)
  - Remove dependency on biasMode and apodName
- **SCLSpecInstRsrp:**
  - Significant increase in signal-to-noise
  - Remove bright mode files (one set of products used for both bright and nominal modes)
  - Remove dependency on biasMode and apodName
- **SCLSpecBeamParam:**
  - Update to match new RSRFs
  - Update beam FWHM (very small difference from previous version)
  - Remove bright mode files (one set of products used for both bright and nominal modes)
  - Remove dependency on biasMode and apodName
- **SCLSpecTeleModel:** update to include data from recent days.
- **SCLSpecBrightGain:** update for bright mode pipeline in voltage, and now contains both PCAL and frequency dependent gains.
- **SCLSpecNonLinCorr:** removed - now included in BrightGain product as extra columns.
- **SCLSpecNonLinCorr:** updated for bright mode
- **SCLSpecChanTimeConst:** create placeholder for bright mode numbers (no actual change to numbers yet).

**Photometric:**
- **SCLPhotFluxConv:** update to use new Neptune model.
- **SCLPhotTempDriftCorr:** update to match new flux conversion product.
- **SCLPhotChanRelGain:** update to match new flux conversion product.

**Release Date**

Date after release of HCSS xx.x.

---

Figure 5.6. The SPIRE calibration view GUI.
The window is divided into three parts:

- At the top, all of the available calibration trees are listed, showing which are compatible (or incompatible) with the version of HIPE being used. The installed trees are coloured black, and the default tree (recorded in the `spire.cal.pool` property) is coloured blue. Trees that are not installed locally are coloured red. It is possible to only show "compatible" trees using the SPIRE Calibration Preferences (see below and Figure 5.7).

- In the central part, the release note for the selected tree is shown.

- At the bottom, there are three buttons to install, set as default, or remove the selected calibration tree. Note that the buttons are not active if the specific action is not allowed for the selected tree version. By default, the incompatible versions are not installable (this can be changed using the preference panel).

Configuration of the SPIRE Calibration View is possible in the HIPE preferences panel (Edit > Preferences > Data Access > Spire Calibration) - see Figure 5.7. The preference panel gives options such as selecting which server to download the calibration tree from (default is the HSA), turning on or off the automatic updater when HIPE starts and configuring the behaviour of incompatible versions.

Figure 5.7. The preference panel for the SPIRE Calibration view.

5.6. Updating Individual Calibration Products

The tasks in the pipeline take individual calibration products as input. This means that any individual calibration product can be supplied directly to the task if an updated test version is available. The easiest way to achieve this is to read the test version of the calibration product into the HIPE session from a FITS file. For example, if I made my own Channel Gain File called, `SCalSpecChanGain_v1_bright.fits`,

```python
newChanGain = fitsReader("SCalSpecChanGain_v1_bright.fits")
```

Another method would be to add the new test calibration product directly into the calibration context as a new version of that product. This would mean the pipeline could be run unchanged, just with a modified calibration context. It can be achieved, but is a little more complicated - it relies on the new product having a complete and correct set of metadata. The following command uses the metadata to add the product into the correct place inside the context,

```python
obs.calibration.addProduct(newChanGain)
```

If this doesn't work, the most likely problems are: the `fileName` metadata is missing (or not constructed using the correct conventions); the version number has not been incremented by one from
the existing version (note that the version is actually read from the `fileName` metadata item); or the "class" of the product is wrong. The following lines of code may be useful to solve these problems,

```python
# to check the fileName metadata
print newChanGain.meta['fileName']
# to set the filename (if it didn't exist before) and the version number
# (a string not an integer)
newChanGain.meta['fileName'] = StringParameter(value="SCalSpecChanGain_v2",
        description="Name of exported file")
newChanGain.version = "2"
# to check the product class
print newChanGain.class
```

### 5.7. Removing Calibration Products from the Tree

It is possible to remove some calibration products from the Calibration Context if it is taking up too much disk space. For example, the Spectrometer calibration context is quite large - if only the Photometer calibration products are needed, the Spectrometer part of the Calibration Context can be removed using:

```python
obs.calibration.spec.refs.clear()
```

The modified calibration tree could then be written back to the disk (if desired) as a new pool,

```python
saveProduct(obs.calibration, pool="spire_cal_phot")
```

### 5.8. Which Calibration Tree Version was used to Process my Data?

The calibration tree version that was used to process an observation is stored in the metadata of the final Level-1 and Level-2 products in the "calVersion" metadata item. In addition, the metadata item "engConvCalVersion" records the calibration tree used for the initial Engineering Conversion, as this might be different if the data have been reprocessed with the User Pipeline scripts in HIPE.

It is also possible to determine which versions of individual calibration products were used for the final products from the "History" dataset. The method to determine the calibration product versions from the History dataset contained within the final products is shown graphically in Figure 5.8 and described below.

1. Find the "HistoryParameters" dataset of the final product
2. Open this with Dataset Viewer in HIPE
3. Look in the last column - this gives the actual filenames of the individual calibration products used in each processing step that was applied to the data - the version is part of this name
5.9. Further Information

Further details of (expert) methods to control or manipulate the calibration tree can be found in the SPIRE Developer's Reference Manual API documentation (Javadoc) in the entry listed under:

```
herschel.spire.ia.cal.SpireCal
```
Chapter 6. SPIRE Photometer Mode Cookbook

6.1. SPIRE Scan Map and Data Structure

6.1.1. A first look at your image maps (The Level 2 Data Product)

All the information for a given SPIRE observation is contained with the Observation Context (described in Section 4.2). In this section we shall see how to examine the data for a SPIRE Large Map observation, however this description applies equally to SPIRE Parallel Mode observations.

The observation we shall be looking at is a Large Map observation of the Planetary Nebulae NGC5315 taking during the Herschel-SPIRE PV phase. NGC5315 is at RA=13h53m57.00s, Dec=−66d30'50.70'' and was covered by scanning the photometer arrays 3 times each in orthogonal direction. The entire process was then repeated (i.e. this observation has 2 repetitions) giving in total 6 scans in each orthogonal direction making 12 scan lines in total.

It is assumed that the observation has already been downloaded into a Pool within your Local Store on your computer as described in section Section 4.2. The Observation Context can be loaded into HIPE using the following 3 lines of Jython Code, where obsid is the unique Observation ID for this observation and pool is whatever name you called your Pool for this observation in your Local Store on disk:

```java
obsid=0x50001833                                     # Select the observation ID
pool  = 'OD117-ScanNGC5315-0x50001833'               # Select the pool name
myObsContext=getObservation(obsid,poolName=pool)     # Get observation context
```

For this particular observation, we chose to call our Pool OD117-ScanNGC5315-0x50001833 where OD117 means the observation was made on Operational Day 117, Scan was the AOT mode, NGC5315 was the target name and 0x50001833 is the unique Observation ID in hexadecimal (or 1342183475 in decimal). Running the above script, reads the Observation Context into memory into the variable myObsContext which appears in the Variables pane of HIPE (See Figure 6.1). Right Clicking (or CTRL-click for Apple Users) on the myObsContext variable brings up another menu. Selecting Open With -- Observation Viewer will open the Observational Context for this observation. The structure of the Observation Context was explained in Section 4.2 and here we shall look at the data inside the Observational Context. We start with the final Product of the SPIRE Large Map pipeline - the image maps. The maps are Level 2 Products and can therefore be found within the Level 2 Context. The maps can be simply accessed by clicking on the level2 folder as shown in Figure 6.2, which reveals a SPIRE Photometer Map Product (or more technically SimpleImage Products) for each of the three SPIRE arrays (PSW, PMW, PLW). Each Photometer Map Product contains 3 Table Datasets corresponding to the image (Jy/beam), error (Standard Deviation) and coverage maps (hit map for samples / pixel) for each array and these are revealed by clicking on the + sign next to the array folder.

For versions of the HCSS prior to HIPE 10.0, a single point source calibrated (Janskys/beam) map was provided in the Level 2 product for each of the PSW, PMW, PLW bands. However, for observations processed with HIPE 10.0 or later, more than one map calibration is made available within the Level 2 product. Maps are provided for the following scenarios for post HIPE 10.0 processing (see the summary in Table 6.1 below.);

- **Point Source Calibrated Maps**: The previous standard point source calibrated maps (in Jy/beam) are provided and appear in the Level 2 context in the form of psrcPxW (where PxW corresponds to PSW, PMW or PLW respectively).
• **Extended Emission Maps**: New maps for extended emission, with absolute calibration derived from the Planck all-sky maps (in MJy/Sr) are provided (solely via the Herschel Science Archive) and appear in the Level 2 context in the form of `extdPxW` (where `PxW` corresponds to PSW, PMW or PLW respectively).

• **SSO Maps**: New maps specifically for Solar System ‘moving’ objects (SSO) calibrated in Jy/beam appear in the Level 2 context in the form of `ssoPxW` (where `PxW` corresponds to PSW, PMW or PLW respectively). These maps differ from the above point source, `psrcPxW`, maps in that the SSO maps are motion corrected and centred on the moving object frame. This map can also be produced using the Photometer SSO Motion Correction useful script (see Section 6.12.1).

In addition, the Level 2 data also contains additional ‘diagnostic’ results from the pipeline destriping process (see Section 6.8.1 for details of the destriping process and an explanation of the diagnostic table produced). These products appear in the Level 2 context as:

• **Point Source Destriper Diagnostic**: Diagnostic table containing the results from the pipeline destriping process used to construct the point source calibrated maps and appear in the Level 2 context in the form of `psrcPxWdiag` (where `PxW` corresponds to PSW, PMW or PLW respectively).

• **Extended Emission Destriper Diagnostic**: Diagnostic table containing the results from the pipeline destriping process used to construct the extended emission calibrated maps and appear in the Level 2 context in the form of `extdPxWdiag` (where `PxW` corresponds to PSW, PMW or PLW respectively).

• **SSO Destriper Diagnostic**: Diagnostic table containing the results from the pipeline destriping process used to construct the SSO frame maps and appear in the Level 2 context in the form of `ssoPxWdiag` (where `PxW` corresponds to PSW, PMW or PLW respectively).

<table>
<thead>
<tr>
<th>Description</th>
<th>Name (HIPE 10+)</th>
<th>Old Name (Pre HIPE 10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point Source (standard) Maps</td>
<td>psrcPSW</td>
<td>PSW</td>
</tr>
<tr>
<td>Extended Emission Maps</td>
<td>extdPSW</td>
<td>-</td>
</tr>
<tr>
<td>Solar System Object Maps</td>
<td>ssoPSW</td>
<td>-</td>
</tr>
<tr>
<td>Point Source Destriper Diagnostic</td>
<td>psrcPSWdiag</td>
<td>pddPSW</td>
</tr>
<tr>
<td>Extended Emission Destriper Diagnostic</td>
<td>extdPSWdiag</td>
<td>-</td>
</tr>
<tr>
<td>SSO map Destriper Diagnostic</td>
<td>ssoPSWdiag</td>
<td>-</td>
</tr>
</tbody>
</table>

The image map can be viewed by clicking on the appropriate array folder (`psrcPSW`, `psrcPMW`, `psrcPLW` for point sources or the equivalent `extd` products for extended emission maps) or alternatively the image map can be displayed in a new window by right clicking on the appropriate array folder and selecting Open With - Standard Image Viewer from the drop down menu as shown in Figure 6.3. This action opens the image in the Image Viewer (a subwindow of the HIPE environment) where the image can be panned, magnified etc. Colours, cut-levels and annotation options can be accessed by right-clicking anywhere on the image. In Figure 6.3 the target can be clearly seen in the centre of the map. Individual scan lines can also be seen. Note also the 2 scan directions can be seen orthogonal to each other producing the cross-linked final image map. The image, error and coverage maps can also be displayed individually by clicking on them or by right-clicking on the appropriate dataset and selecting Open With - Image Viewer for Array Datasets from the drop down menu. Finally, right-clicking on a given image dataset and selecting Open With - Array Dataset Viewer from the drop down menu shows the image (or error or coverage) in table form (Jy/beam for every pixel in the image) as shown in Figure 6.4.

If you want to extract the SimpleImage for the PSW, PMW or PLW array as a data cube containing the image, error and coverage maps to work with, rather than view it with the Image Viewer, on the command line type the rather exhaustive:

```python
myMapProduct=myObsContext.level2.getProduct("psrcPSW")
```
# Then to view each of the map datasets
Display(myMapProduct.image)
Display(myMapProduct.error)
Display(myMapProduct.coverage)

where *myMapProduct* can be any name we choose and the following syntax means from *myObsContext* we want the Level 2 product PSW array Photometer Map Product (reference as psrcPSW, extdPSW or for pre-HIPE 10 PSW respectively). You will also notice that *myMapProduct* now appears in the Variables Panel which can correspondingly be right-clicked on to show the various viewing options available for this product. The next 3 lines in the above script allow us to display the signal, error and coverage maps respectively.

![Figure 6.1. Loading and viewing the Observation Context for the Large Map Observation.](image)

![Figure 6.2. Accessing the final Level 2 Product maps](image)
6.1.2. Saving a map as a FITS file and reading it in again

It is possible that we may also want to look at our image maps in external applications such as DS9 for example and HIPE provides the tools for exporting our maps as conventional fits files. Following on from the previous example above we can send our `myMapProduct(SimpleImage)` product to a FITS file by right-clicking on it in the variable list (which is a pointer, remember, rather than a new array) and selecting Send To - FITS file from the drop down menu. This will open the FITS writer panel as shown in Figure 6.5 where we can type in our desired filename and path. Click on Accept at the bottom of the panel to save the FITS file. This fits file will then be saved as a multi-extension fits file containing the image (hdul), error (hd2) and coverage (hdud3) maps that can then be read into DS9 as a data cube and viewed, along with the general header (hdud0). The same effect can be achieved on the command line by;
which again saves the products as a multi-extension fits file containing the image, error and coverage maps.

```python
simpleFitsWriter(myMapProduct,'mypath/myMap.fits')
```

Reading a FITS file into the HIPE session can be accomplished by either selecting Open File from the File menu in the top right hand corner of the HIPE window. Alternatively, from the command line:

```python
myMap=simpleFitsReader('mypath/myMap.fits')
```

These FITS files are imported as a simpleImage and can be manipulated in the same manner as the simpleImage products described earlier in this section.

**Tip**

The Photometer Map Products (data cubes for each array containing the image, error and coverage arrays) actually exist as fits files within the Pool for this observation in the Local Store. These can be found in the Pool for this example in the folder `/localstore/OD117-ScanNGC5315-0x50001833/herschel.ia.dataset.image.SimpleImage` (where the poolname is "OD117-ScanNGC5315-0x50001833"). The Photometer Map Products have the form `hspireplw.............pmp.fits`

### 6.1.3. Looking at observations with Level 2.5 and Level 3 products

Many individual SPIRE observations may in fact be a subset of a larger observation. For example, a pair of parallel mode observations in opposite directions or an ensemble of observations to cover a large contiguous area. For such observations, SPIRE data may contain additional Level 2.5 and Level 3 products (See also Section 4.2.3 and Section 4.2.4). The Level 2.5 products are created from pairs of single parallel scan observations in the nominal and orthogonal directions with no multiple points included. Additionally, Level 2.5 products are created for groups of 2+ Parallel Mode observations or single scan direction observations where processing to the Level 2.5 stage improves the observation. Maps are produced using the standard pipeline. The Level 3.0 products are mosaiced maps obtained by merging all contiguous observations over an area of sky, except where the contiguous coverage results in excessively large products (particularly the Galactic plane), in which case the Level 3 product will be a manageable chunk of around 15 degrees.

Such observations will have the additional Level 2.5 and Level 3 maps attached to their Observation Context as shown in the example in Figure 6.6 (which shows the Level 2 product selected).
Selecting the Level 2.5 product if available as in Figure 6.7, shows the map with the Level 2 product from this observation combined with its associated opposite scan direction parallel mode observation. The two obsid’s used to create the Level 2.5 map can be found in the Level 2.5 meta data (referred to as obsid001 and obsid002). Level 2.5 products contain the following products:

- Point Source calibrated maps (Jy/beam).
- Extended emission absolute calibrated maps (MJy/sr).
- The associated destriper diagnostic products.

For observations that cover contiguous areas of the sky including deep observations over the same area of sky, Level 3 products may also be available. Level 3 products are mosaics of Level 2 and (where available) Level 2.5 maps and can be very larger in some circumstances as shown in Figure 6.8 for the representative observation ID 1342189081. Level 3 map products are only available as maps absolute calibrated for extended emission in MJy/sr. The list of observations used to build the Level 3 maps are included in the observations’ metadata (referred to as obsid001 to obsidxxx). Note that there is no overlap between Level 3 map slices.

Any SPIRE map can be mosaiced together using the mosaic task (in conjunction with the mosaicWcsCreator task). The SPIRE mosaic task takes as input a list of images. This list can be created using the first section of the code snippet below and then fed into the mosaic task GUI as shown Figure 6.9 to create a new mosaiced map from 3 input maps. Since some mosaics may be different shapes and sizes (e.g. along the Galactic plane), the default WCS may not be the most optimal to produce the final mosaic. In this case the mosaicWcsCreator task can be used to produce a new WCS that can then be input to the original mosaic task (see Figure 6.10). The mosaicWcsCreator task takes as input a list of Observation Contexts which can be created using the second section of the code below:

```python
### SPIRE Mosaic Tool Example
# Get the observation contexts to mosaic
obs1=getObservation(1342226998, useHsa=True)
obs2=getObservation(1342229159, useHsa=True)
obs3=getObservation(1342227739, useHsa=True)

# Read the maps to mosaic
map1=obs1.level2.getProduct("extdPSW")
map2=obs2.level2.getProduct("extdPSW")
map3=obs3.level2.getProduct("extdPSW")

# Create a list of Maps
mapList=[map1,map2,map3]

# Create Mosaic Map
mosaicMap=mosaic(images=mapList)

#----------------------------------
### SPIRE WCS Creator Tool Example
# Create a list of observation context
obsList=(obs1,obs2,obs3)

# Create new WCS
outWcs = mosaicWcsCreator(input=obsList)

# Create Mosaic Map with new WCS
mosaicMap = mosaic(images=mapList, wcs=outWcs)
```
Figure 6.6. Observation Context showing Level 2, Level 2.5 and Level 3 products (the viewer shows the Level 2)

Figure 6.7. Level 2.5 product constructed from 2 parallel mode scan map observations
Figure 6.8. Level 3 product constructed from contiguous observations connected to obsid=1342189081.

Figure 6.9. Mosaic Level 2 maps to create a Level 3 map.
6.1.4. Looking at the Level 1 Timeline Data

The image maps have been created from the individual timelines of detectors as they were scanned across the target. These timelines are the Level 1 products from the Photometer Large Map Pipeline and are also available from the Observation Context. The Level 1 Large Map products are referred to as Photometer Scan Products.

The SPIRE level 1 products are the destriped timelines. Any baseline offsets having previously been removed by the destriper in the pipeline (Note that the Level 1 data processed with pre-HIPE 10.0 versions is NOT destriped.)

In Figure 6.11 we show how the Level 1 products can be accessed from the observational context. Note that within the Level 1 Context there are a total of 12 Products labelled from 0 to 12. These are all Photometer Scan Products. As noted earlier the map of NGC5315 was constructed by scanning the photometer arrays 3 times in each orthogonal direction twice making a total of 12 scan lines in total. Although the numbering system seems anonymous, the actual name of the Building Block can still be revealed by checking the Meta Data bbTypeName in the Photometer Scan Product (i.e. click on one of the folders numbered 1-12). The column names give the time, and then the signal for each detector on the arrays (not the first entry PSWR1, actually a resistor, measured in Volts and the following bolometers measured in Jy and a thermistor (PSWT1) again measured in volts, etc.).

Each Photometer Scan Product contains 5 individual Table Datasets (and a Product containing the processing history) as shown in Figure 6.11 and defined below;

- **Signal Table**: A table containing the Sample Time (in seconds) and a column for the signal from every bolometer including both detector (in Jy/beam) and non-detector (e.g. thermistor, resistor in Volts) channels

- **Mask Table**: A table containing the Sample Time (in seconds) and a column for every bolometer including both detector and non-detector (e.g. thermistor, resistor) channels with a mask value corresponding to which processing flags have been raised. The masks are defined in Section 4.3.

- **RA Table**: A table containing the Sample Time (in seconds) and a column for the RA on the sky in degrees for each detector (not including non-detector channels)

- **Dec Table**: A table containing the Sample Time (in seconds) and a column for the Dec on the sky in degrees for each detector (not including non-detector channels)

- **Temperature Table**: A table containing the Sample Time (in seconds) and a column for each Thermistor channel temperature (measured in Kelvin)

These individual Table Datasets correspond to data from a single scan line and can be viewed either as - by right-clicking - array tables (by selecting Open With Data Set Viewer) or plotted (by selecting Open With Table Plotter). Although the use of Table Plotter is beyond the scope of this document, an example is shown in Figure 6.12 where we have selected to plot the Sample Time against the Signal from the PSW D16 bolometer for this particular scan line.
Individual Table Data Sets can also be extracted from the Observational Context using the alternative command line script. Using Figure 6.12 as a guide we can see the following:

```bash
# Extract the Photometer Scan Product for the first Scan Line
scanLine1=myObsContext.refs["level1"].product.refs[0].product

# or extract the Photometer Scan Product for the second Scan Line
scanLine2=myObsContext.refs["level1"].product.refs[1].product

# Get the Signal Table from the first Scan Line
signalScanLine1=scanLine1['signal']

# Get the array of values for the Sample Time
timeScanLine1=signalScanLine1['sampleTime'].data

# Get the array of values for the PSW D16 Detector
PSWD16ScanLine1=signalScanLine1['PSWD16'].data
```
where scanLine1, etc can be any name we choose and the following syntax means from myObsContext we want the Level 1 product Photometer Scan Product for the first scan line (i.e. element [0]). You will also notice that scanLine1 now appears in the Variables Panel which can correspondingly be right-clicked on to show the various viewing options available for this product. The following lines show the procedure for extracting the second scan line (i.e. array element [1]) and go on to extract, for the first scan line the Signal Table Dataset. Finally the sampleTime and detector signal for the PSWD16 detector are extracted as normal arrays of numbers. The final list of variables in the HIPE Variable Pane is shown in Figure 6.13.

![Variables Panel](image)

Figure 6.13. Final Level 1 Photometer Scan Product Timeline Data variable list

### 6.1.5. SPIRE Serendipity Data at Level 1

When a SPIRE observation is made, the telescope must slew to the required point on the sky. During this slew, SPIRE data is being taken. This is referred to as Serendipity Scans. The Serendipity Scans are included in the raw Level 0 data and Level 0.5 Scans are also processed through the Level 0.5 - Level 1 pipeline (if the map turnarounds are included). Serendipity Scans are distinguished by a SERENDIPTY flag. All serendipity data in the archive is deglitched however, it is not destriped.

Note that the Serendipity Mode scan is always the last in the order of the Level 1 context, although it happens in terms of observation time before the actual observation mapping scans. In Figure 6.14 the Level 1 context is shown for a standard large map observation. The left panel shows the result for selecting the number 7 scan line in the observation. The building block is SpireBbScanLine (BBID=2701328392 or in hexadecimal 0xA103 as shown in Table 6.2). Viewing with the Detector Timeline Explorer, the scan line turnarounds are flagged in green with some other flags for this scan line. On the right panel, the equivalent view is shown for the number 8 final scan line in the observation. This is the Serendipity Scan and the building block is SpireBbPhotSerendipity (BBID=2701393921 or in hexadecimal 0xA104 as shown in Table 6.2). Viewing with the Detector Timeline Explorer, the entire scan is flagged in purple indicating that the scan is serendipity data.

Note that Serendipity Mode Scans were always executed after the new pointing command was given to the spacecraft and thus were included as building blocks of the next observation. At the end of the slew the instrument is configured which includes re-setting the offset voltages of the readout electronics. They are adapted to the background emission seen by the detectors at the new position. Thus, electronic offsets during the slew are still the ones from the previously executed observation and may thus be different from the observation to which they are attached.
Figure 6.14. Serendipity Scan in the Level 1 data.

In Figure 6.15 the motion of the detectors across the sky as the telescope slews in order to make a small map observation is shown. The long blue arc is the Serendipity Scan, the final small map is zoomed in for clarity. The resulting image (including the Serendipity Scan) is shown in Figure 6.16.

Note that by default the Serendipity Scans are not processed in the final map making. By default the parameter "useOnlyMasks" in the Destriper and the Naive Scan Mapper has the SERENDIPITY Flag set. Unsetting this flag will include the Serendipity Scan data into the map, often leading to extremely large maps filled with a lot of NaNs, so care should be taken with this option.

Furthermore it should be noted that including the Serendipity Scan data into the destriping process rarely leads to satisfactory results as there are too few overlapping detector scans with the map. The Serendipity Data is provided at Level 1 for completeness so that the dedicated astronomer has calibrated data available. Yet this data, still needs special treatment best done through a script, to find the right offsets to be subtracted and to remove potential residual 1/f noise.
Figure 6.15. Serendipity Scan motion across sky to make a small map.

Figure 6.16. Serendipity Scan processed as an image.
6.1.6. Looking at the Level 0.5 Timeline Data

These timeline data have been created by processing the raw Level 0 data through the Common Engineering Conversion (Level 0 - Level 0.5) Pipeline. The Level 0.5 data are the uncalibrated, uncorrected timelines measured in Volts. The level 0.5 products are also available from the Observation Context. The Level 0.5 context folder can be seen in the Observation Context and can be opened by clicking on the + next to the level0_5 folder. The Level 0.5 context contains a lot more data than the Level 1 context and includes all the data necessary to process the observation and produce science quality data. In Figure 6.17 we show all the Level 0.5 data within the observation context. We see that there are a total of 31 entries in the list informatively labelled from 0 to 30 with annotated building block names (Note that PCAL calibration flashes are no longer made at the beginning of the observation since Operational Day OD302 for Large Map mode and OD341 for Parallel mode). This can be compared to a total of 12 entries that we saw for the Level 1 products. The Level 0.5 context contains all the building blocks used in the observation and in Figure 6.17 we show how this Large-Map observation was built up from the individual building blocks. In the figure, the building blocks can be divided into roughly 4 general types, configuration blocks, calibration blocks, science blocks and movement blocks. The type of building block can be revealed by clicking on a given number from 0-30 and scrolling down the Meta data window pane to the BBtypeName entry. The individual blocks are described below in Table 6.2:

<table>
<thead>
<tr>
<th>BB number</th>
<th>BB Type</th>
<th>BB Hex prefix</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SpireBbObsConfig</td>
<td>0xAF01</td>
<td>Initial configuration</td>
</tr>
<tr>
<td>1</td>
<td>SpireBbPhotSerendipity</td>
<td>0xA104</td>
<td>Slew to target</td>
</tr>
<tr>
<td>2</td>
<td>SpireBbPOF5Config</td>
<td>0xA050</td>
<td>AOT configuration</td>
</tr>
<tr>
<td>3</td>
<td>SpireBbPOF5Init</td>
<td>0xA051</td>
<td>Initialize the AOT</td>
</tr>
<tr>
<td>4</td>
<td>SpireBbPcalFlash</td>
<td>0xA801</td>
<td>Photometer Calibration Lamp Flash</td>
</tr>
<tr>
<td>5</td>
<td>SpireBbScanLine</td>
<td>0xA103</td>
<td>A large map scan line</td>
</tr>
<tr>
<td>6</td>
<td>SpireBbMove</td>
<td>0xAF00</td>
<td>Scan Line turnaround movement</td>
</tr>
<tr>
<td>7</td>
<td>SpireBbScanLine</td>
<td>0xA103</td>
<td>A large map scan line</td>
</tr>
<tr>
<td>8</td>
<td>SpireBbMove</td>
<td>0xAF00</td>
<td>Scan Line turnaround movement</td>
</tr>
<tr>
<td>..</td>
<td>SpireBbScanLine</td>
<td>0xA103</td>
<td>A large map scan line</td>
</tr>
<tr>
<td>..</td>
<td>SpireBbMove</td>
<td>0xAF00</td>
<td>Scan Line turnaround movement</td>
</tr>
<tr>
<td>..</td>
<td>..</td>
<td>..</td>
<td>........</td>
</tr>
<tr>
<td>27</td>
<td>SpireBbScanLine</td>
<td>0xA103</td>
<td>A large map scan line</td>
</tr>
<tr>
<td>28</td>
<td>SpireBbMove</td>
<td>0xAF00</td>
<td>Scan Line turnaround movement</td>
</tr>
<tr>
<td>29</td>
<td>SpireBbPcalFlash</td>
<td>0xA801</td>
<td>Photometer Calibration Lamp Flash</td>
</tr>
<tr>
<td>29</td>
<td>SpireBbPOF5End</td>
<td>0xA052</td>
<td>End of AOT</td>
</tr>
</tbody>
</table>
Figure 6.17. Anatomy of Level 0.5 Building Block structure for a Large Map observation

Note that in the meta data, the phase parameters of the detector bias that appear in the header in the Level 1 products are given in radians while the ones shown in the Level 0.5 data products (as well as in the Nominal Housekeeping - NHK - data) are given in degrees. Converting the Level 1 parameters into degrees yields a value different from the other two. It turns out that phase(Level 0.5) = phase(NHK) = (180 - phase(level1)).

Looking at some of the individual entries in the Level 0.5 context, it can be seen that the individual Building Blocks are built up from a variety of different types of Products. clicking on the + sign for a given Building Block number reveals what Products a particular Building Block is made from. In Figure 6.18 the first handful of building blocks for our observation are opened to view the contents. The contents are a variety of Products referred to by acronyms such as CHKT, NHKT, PDT, POT, SCUT, etc, described in order of importance below:

Example building blocks may be;

- **PDT**: The Photometer Detector Timeline contains the Level 0.5 detector data.
- **NHKT**: The Nominal House Keeping Timeline contains the housekeeping data with all the settings for this observation.
- **CHKT**: The Critical House Keeping Timeline contains all the critical parameters of the instrument such as the electronics.
- **SCUT**: The Sub Control Unit Timeline contains monitoring data for the instrument operation for this observation.
• **POT**: The Photometer Offset Timeline contains all the raw DC offsets in ADU that have already been used in the raw data processing to set the dynamic range of the detectors.

Note that Building blocks such as the Slewing (Serendipity Building Block), Calibration flash and the scan line turnarounds all contain PDT data. Indeed, the scan line turnaround Building Block data IS used for scientific processing. The CHKT, NHKT, POT, SCUT Products all contain a signal table, containing data arrays and a mask table containing flag information. The Level 0.5 PDT Photometer Detector Timeline Products contain 4 Table dataset arrays;

• **Voltage Table**: A table containing the Sample Time (in seconds) and a column for the signal measured in Volts for every bolometer including both detector and non-detector (e.g. thermistor, resistor) channels.

• **Resistance Table**: A table containing the Sample Time (in seconds) and a column for the Resistance measured in Ohms for every bolometer including both detector and non-detector (e.g. thermistor, resistor) channels.

• **Mask Table**: A table containing the Sample Time (in seconds) and a column for every bolometer including both detector and non-detector (e.g. thermistor, resistor) channels with a mask value corresponding to which processing flags have been raised. The masks are defined in Section 4.3.

• **Temperature Table**: A table containing the Sample Time (in seconds) and the temperature of the 6 Thermistors (2 per array) in Kelvin.

• **Quality Table**: A table containing any Quality Flags raised for each detector.

In Figure 6.18 the PDT for the first Scan Line Building Block has been selected. Right-clicking and selecting **Open-with Dataset Viewer**, opens the voltage table in a new window. Any of the Table Data Sets can also be viewed graphically by selecting **Open-with Table Plotter** as shown in Figure 6.19. In the plot window the bolometer signal to plot can be selected from the Y-axis menu and many bolometers can be overlaid by ticking the overlay box (both circled in the plot window).

Figure 6.18. Inside the Level 0.5 Building Block structure for a Large Map observation
6.1.7. Looking at the Raw Level 0 Data

The Raw data formatted from the satellite telemetry is also available within the Observation Context. These are the Level 0 Products and will in most circumstances be of no general interest. The Level 0 Context, shown in Figure 6.20, contains 30 entries. Note that there is a significant difference in the Level 0 data structure compared to the Level 0.5 Products. In the Level 0.5 Products, each individual block in the observation has several data products (e.g. Scan line, Housekeeping data, etc - see Table 6.2).

However, in order to reduce the raw data volume at the Level 0 stage, all the data from a single building block are packed into a single Level 0 product, referred to as a Raw SPIRE Timeline (RST) for each building block, i.e. A single Level 0 product contains many separate Table datasets (one for each telemetry packet produced from the satellite). Clicking on a given number within the Level 0 context reveals the Level 0 Product for that particular building block. These products are the raw data versions of the Level 0.5 data and contain Table Datasets such as the Critical House Keeping timelines (CHK), Nominal House Keeping timelines (NHK), Raw Photometer Detector timelines (PHOTF), Raw Photometer Offset timelines (PHOTOFF) and Sub-Control Unit timelines (SCUNOMINAL).

The Raw Photometer Detector Timeline (PHOTF) Table Dataset can be viewed by right-clicking and selecting Open with - Dataset Viewer, see Figure 6.20, we find quite a different structure to the Level 0.5 PDT datasets. There are 288 columns, one for every SPIRE channel, numbered not in the familiar PSWE8, PSWE9 notation but rather as PHOTARRAY001 -- PHOTARRAY288 which corresponds to their Channel Number (from an electrical designation). The signal is still in raw ADU and there are many different time columns which correspond to various measures of the data frames, telemetry packets and packet sequence counts, etc. The only flags are contained in the PHOT-FADCFLAGS column which is set in the case of a problem with ADC process in telemetry. A full description of the data structure can be found in the Products Definition Document (HERSCHEL-HSC-DOC-0959) or the SPIRE Pipeline Description Document (SPIRE-RAL-DOC-002437).
6.2. SPIRE Point Source Mode Data Structure

6.2.1. Looking at the Point Source Mode Data

All the information for a given SPIRE observation is contained with the Observation Context (described in Section 4.2). In this section we shall see how to examine the data for a SPIRE Point Source observation. A point source observation is a staring observation, i.e. the telescope is not scanned across the sky in this mode. In order to recover the source a 7-point hexagonal jiggle pattern is made around the source position. Sky backgrounds are removed by chopping using the Beam Steering Mirror (BSM) over a distance of plus/minus 63 arcsec and any emission due to the telescope structure is removed by nodding the entire telescope and repeating the chop-jiggle cycle.

The observation we shall be looking at is a Point Source observation of the Planetary Nebulae NGC5315 taking during the Herschel-SPIRE Performance Verification (PV) phase. NGC5315 is at RA=13h53m57.00s, dec=-66d30'50.70'' and was covered by making 2 repetitions of the Point Source Mode which consisted of a pair of chopped and nod cycles at each of the 7 jiggle positions in the pattern.

It is assumed that the observation has already been downloaded into a Pool within your Local Store on your computer as described in section Section 4.2. The Observation Context can be loaded into HIPE using the following 3 lines of Jython Code, where obsid is the unique Observation ID for this observation and pool is whatever name you called your Pool for this observation in your Local Store on disk:

```python
obsid=0x50001832
pool = 'OD117-7pt.NGC5315-0x50001832'
myObsContext=getObservation(obsid, poolName=pool)
```

# Select the observation ID
# Select the pool name
# Get observation context
For this particular observation, we chose to call our Pool OD117-7ptNGC5315-0x50001832 where OD117 means the observation was made on Operational Day 117, 7pt was the AOT mode, NGC5315 was the target name and 0x50001832 is the unique Observation ID in hexadecimal (1342183474 in decimal). Running the above script, reads the Observation Context into memory into the variable myObsContext which appears in the Variables pane of HIPE (See Figure 6.21). Right Clicking (or CTRL-click for Apple Users) on the myObsContext variable brings up another menu. Selecting Open With -- Observation Viewer will open the Observational Context for this observation. The structure of the Observation Context was explained in Section 4.2 and he we shall look at the data inside the Observational Context. We start with the final Product of the SPIRE Point Source pipeline - The Jiggled Photometer Product (JPP). The JPP is a Level 2 Product and can therefore be found within the Level 2 Context. The JPP can be simply accessed by clicking on the level2 folder as shown in Figure 6.22, which reveals a SPIRE Jiggled Photometer Product. Right-clicking on the JPP and selecting Open With - Array Dataset Viewer from the drop down menu shows the data in table form as shown in Figure 6.22. The JPP contains a Table Dataset with a row for each array with the following information:

- **Array Name**: A column listing each array PSW, PMW, PLW.

- **RA**: A column listing the final fitted Right Ascension (i.e. fitted from the 7-pt jiggle pattern) for each array to the detected source within the 7-point Jiggle pattern for the target detector in decimal degrees

- **RA Error**: A column listing the errors on the Right Ascension for each array

- **Dec**: A column listing the final fitted Declination (i.e. fitted from the 7-pt jiggle pattern) for each array to the detected source within the 7-point Jiggle pattern for the target detector in decimal degrees

- **Dec Error**: A column listing the errors on the Declination for each array

- **Signal**: A column listing the Gaussian fitted signal for the target detector for each array to the detected source within the 7-point Jiggle pattern in Jy (in beam flux)

- **Error**: A column listing the error on the fitted signal for each array

Figure 6.21. Loading and viewing the Observation Context for the Photometer Point Source Observation.
6.2.2. Reading the JPP into memory and saving it as a FITS file and reading it in again

It is possible that we may also want to export our data and HIPE provides the tools for exporting data products as conventional fits files. The Level 2 JPP can be read into memory with the following admittedly long-winded command from the command line:

```python
# read entire Product
myJpp=myObsContext.level2.getProduct("JPP")
#
# read the RA data array
myRa=myJpp["outputDataset"]["ra"].data
print myRa
# read the RA for PSW array
myRaPsw=myJpp["outputDataset"]["ra"].data[0]
print myRaPsw
```

This creates a new entry `myJpp` in the Variables Pane of HIPE which can correspondingly be right-clicked on to show the various viewing options available for this product. The next 4 lines in the above script allow us to read in and print out the data for the Right Ascension for all arrays and for just the PSW array (creating entries for `myRa` and `myRaPsw` in the variable pane). The JPP Level 2 Product can be saved as a FITS file by the following command line entry;

```python
simpleFitsWriter(myJpp,'mypath/myJPP.fits')
```

where `mypath` is the desired path. Alternatively the product can be sent to a FITS file by right-clicking on it in the variable list and selecting Send To - FITS file from the drop down menu. This will open the FITS writer panel as shown in Figure 6.23 where we can type in our desired filename and path. Click on Accept at the bottom of the panel to save the FITS file.
Figure 6.23. Exporting the JPP as a FITS file

Reading a FITS file into the HIPE session can be accomplished by either selecting Open File from the File menu in the top right hand corner of the HIPE window. Alternatively, from the command line:

```python
myJPP=simpleFitsReader('mypath/myJPP.fits')
```

These FITS files are imported as an JPP Product dataset and can be manipulated in the same manner as described earlier throughout this section.

**Tip**

The JPP actually exist as a fits file within the Pool for this observation in the Local Store. These can be found in the Pool for this example in the folder /localstore/OD117-7ptNGC5315-0x50001832/herschel.spire.ia.dataset.JiggPhotProduct (where the pool-name is "OD117-7ptNGC5315-0x50001832"). The JPP will have the hspirephotometer........jpp.fits

### 6.2.3. Looking at the Level 1 Data for Point Source Observations

The final Level 2 Jiggle Photometer Product has been created from a Gaussian fit to the 7-point jiggle pattern of a target bolometer. The information on the individual jiggle positions for all bolometers is contained within the Level 1 Product and are also available from the Observation Context. The Level 1 Point Source mode product is referred to as the Averaged Pointed Photometer Product (APPP). In Figure 6.24 we show how the Level 1 product can be accessed from the observational context. The APPP holds information for each of the 7 jiggle positions for all bolometers after the signal has been demodulated (chopped) and de-noded.

Each Averaged Pointed Photometer Product contains 7 individual Table Datasets (and a Product containing the processing history) as shown in Figure 6.24 and defined below:

- **Signal Table**: A table containing a column for the Jiggle ID (1-7 position) and a column for the signal from every detector channel (in Jy/beam)
- **Error Table**: A table containing a column for the signal error from every detector channel (in Jy/beam)
- **Dec Table**: A table containing a column for the declination on the sky in degrees for every detector channel
- **Dec Error Table**: A table containing a column for the errors in declination on the sky in degrees for every detector channel
- **RA Table**: A table containing a column for the right ascension on the sky in degrees for every detector channel
• **RA Error Table:** A table containing a column for the errors in right ascension on the sky in degrees for every detector channel

• **Mask Table:** A table containing the mask value for every detector channel corresponding to which processing flags have been raised. The masks are defined in Section 4.3.

The APPP can be viewed either - by right-clicking - array tables (by selecting Open With - Data Set Viewer) or plotted (by selecting Open With - Table Plotter). Although the use of Table Plotter is beyond the scope of this document, an example is shown in Figure 6.25 where we have selected to plot the Jiggle ID against the Signal from the PSW E10 bolometer for the APPP.

6.2.4. Looking at the Level 0.5 Timeline Data for Point Source Observations

The Level 2 JPP and the Level 1 APPP products represent the output from the Point Source pipeline. These data products were created from the lower Level 0.5 data products, which were correspondingly
created from processing the raw Level 0 data through the Common Engineering Conversion (Level 0 - Level 0.5) Pipeline. The Level 0.5 data are the calibrated, timelines measured in Volts uncorrected for detector effects. These level 0.5 products are also available from the Observation Context. The Level 0.5 context folder can be seen in the Observation Context and can be opened by clicking on the + next to the level0_5 folder.

The Level 0.5 context contains a lot more data than the Level 1 context and includes all the data necessary to process the observation and produce science quality data. In Figure 6.26 we show all the Level 0.5 data within the observation context (Note that since Operational Day OD302 PCAL calibration flashes are no longer made at the beginning of the observation). We see that there are a total of 23 entries in the list informatively labelled from 0 to 22. This can be compared to the single final product that we saw for the Level 1 data.

The Level 0.5 context contains all the building blocks used in the observation and in Figure 6.26 we show how this Point Source observation was built up from the individual building blocks. In the figure, the building blocks can be divided into roughly 4 general types, configuration blocks, calibration blocks, science blocks and movement blocks. The type of building block can be revealed by clicking on a given number from 0-22 and scrolling down the Meta data window pane to the BBtypeName entry. The individual blocks are described below in Table 6.3. This observation involves two repetitions of the Point Source Mode. A single science building block consists of an operation at a given Nod position (denoted A or B) and moving to the first jiggle position on the 7-point pattern, chopping 8 times on/off source, moving to the second position, .... until all 7 positions have been visited (plus one more at the centre). This operation is then repeated at the next nod position (position B), repeated at B and then once more at nod position A. One repetition thus corresponds to a single ABBA nod cycle, therefore this observation will consist of 2 ABBA cycles.
Figure 6.26. Anatomy of Level 0.5 Building Block structure for a Point Source observation

Table 6.3. Description of the Building Blocks in a Point Source Mode Level 0.5 Context

<table>
<thead>
<tr>
<th>BB number</th>
<th>BB Type</th>
<th>BB Hex prefix</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SpireBb_StartObsAll</td>
<td>0xB6C8</td>
<td>Begin Observation</td>
</tr>
<tr>
<td>1</td>
<td>SpireBbPOF2Config</td>
<td>0xA020</td>
<td>Initial configuration of the Point Source AOT</td>
</tr>
<tr>
<td>2</td>
<td>SpireBbPOF2Init</td>
<td>0xA021</td>
<td>Initialize the Point Source AOT</td>
</tr>
<tr>
<td>3</td>
<td>SpireBbPcalFlash</td>
<td>0xA801</td>
<td>Photometer Calibration Lamp Flash</td>
</tr>
<tr>
<td>4</td>
<td>SpireBbJiggle</td>
<td>0xA321</td>
<td>Carry out chopped motion around 7-point jiggle pattern at first nod position</td>
</tr>
<tr>
<td>5</td>
<td>SpireBbMove</td>
<td>0xAF00</td>
<td>Movement of Nod position (position A to B)</td>
</tr>
<tr>
<td>6</td>
<td>SpireBbJiggle</td>
<td>0xA321</td>
<td>Carry out chopped motion around 7-point jiggle pattern at second nod position</td>
</tr>
<tr>
<td>7</td>
<td>SpireBbMove</td>
<td>0xAF00</td>
<td>Movement of Nod position (dwell at position B)</td>
</tr>
<tr>
<td>BB number</td>
<td>BB Type</td>
<td>BB Hex prefix</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>----------------------</td>
<td>---------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>8</td>
<td>SpireBbJiggle</td>
<td>0xA321</td>
<td>Carry out chopped motion around 7-point jiggle pattern at second nod position</td>
</tr>
<tr>
<td>9</td>
<td>SpireBbMove</td>
<td>0xAF00</td>
<td>Movement of Nod position (position B to A)</td>
</tr>
<tr>
<td>10</td>
<td>SpireBbJiggle</td>
<td>0xA321</td>
<td>Carry out chopped motion around 7-point jiggle pattern at first nod position</td>
</tr>
<tr>
<td>11</td>
<td>SpireBbMove</td>
<td>0xAF00</td>
<td>Movement of Nod position (dwell at position A)</td>
</tr>
<tr>
<td>12 -- 19</td>
<td>...</td>
<td>...</td>
<td>Repeat entries 4-11</td>
</tr>
<tr>
<td>20</td>
<td>SpireBbPcalFlash</td>
<td>0xA801</td>
<td>Photometer Calibration Lamp Flash</td>
</tr>
<tr>
<td>21</td>
<td>SpireBbPOF2End</td>
<td>0xA022</td>
<td>End of AOT</td>
</tr>
<tr>
<td>22</td>
<td>SpireBb_EndObsAll</td>
<td>0xB6C7</td>
<td>End Observation</td>
</tr>
</tbody>
</table>

Looking at some of the individual entries in the Level 0.5 context, it can be seen that the individual Building Blocks are built up from a variety of different types of Products. Clicking on the + sign for a given Building Block number reveals what Products a particular Building Block is made from. In Figure 6.27 the first handful of building blocks for our observation are opened to view the contents. The contents are a variety of Products referred to by acronyms such as CHKT, NHKT, PDT, BSMT, POT, SCUT, etc, described in order of importance below:

Example building blocks may be:

- **PDT**: The Photometer Detector Timeline contains the Level 0.5 detector data.

- **BSMT**: The Beam Steering Mechanism Timeline contains the information of the BSM (chop and jiggle positions as a function of time).

- **NHKT**: The Nominal House Keeping Timeline contains the housekeeping data with all the settings for this observation.

- **CHKT**: The Critical House Keeping Timeline contains all the critical parameters of the instrument such as the electronics.

- **SCUT**: The Sub Control Unit Timeline contains monitoring data for the instrument operation for this observation.

- **POT**: The Photometer Offset Timeline contains all the raw DC offsets in ADU that have already been used in the raw data processing to set the dynamic range of the detectors.
The CHKT, NHKT, BSMT, POT, SCUT Products all contain a signal table, containing data arrays and a Mask table containing flag information. The Level 0.5 PDT Photometer Detector Timeline Products contain 5 table dataset arrays:

- **Voltage Table**: A table containing the Sample Time (in seconds) and a column for the signal measured in Volts for every bolometer including both detector and non-detector (e.g. thermistor, resistor) channels.

- **Resistance Table**: A table containing the Sample Time (in seconds) and a column for the Resistance measured in Ohms for every bolometer including both detector and non-detector (e.g. thermistor, resistor) channels.

- **Mask Table**: A table containing the Sample Time (in seconds) and a column for every bolometer including both detector and non-detector (e.g. thermistor, resistor) channels with a mask value corresponding to which processing flags have been raised. The masks are defined in Section 4.3.

- **Quality Table**: A table containing any Quality Flags raised for each detector.

- **Temperature Table**: A table containing the Sample Time (in seconds) and the temperature of the 6 Thermistors (2 per array) in Kelvin.

In Figure 6.27 the PDT for the first Jiggle Building Block has been selected. Right-clicking and selecting Open-with-Dataset Viewer opens the voltage table in a new window. Any of the Table Data Sets can also be viewed graphically by selecting Open-with-Table Plotter as shown in Figure 6.28. In the plot window the bolometer signal to plot can be selected from the Y-axis menu.
(circled in the plot window) and in this example the signal versus sample time for bolometer PSW E6 has been selected. In the figure we also plot a marked line selected from the Display Style box (also circled in the plot window). In Figure 6.28 the on and off chop positions and the circuit around the 7 jiggle positions can be clearly seen.

Figure 6.28. Plotting the Level 0.5 data for a 7-point Jiggle Point Source observation

6.2.5. Looking at the Raw Level 0 Data

The raw data formatted from the satellite telemetry is also available within the Observation Context. These are the Level 0 Products and will in most circumstances be of no general interest. The Level 0 Context, shown in Figure 6.29, contains 23 entries. Note that there is a significant difference in the Level 0 data structure compared to the Level 0.5 Products. In the Level 0.5 Products, each individual block in the observation has several data products (e.g. Scan line, Housekeeping data, etc - see Table 6.3).

However, in order to reduce the raw data volume at the Level 0 stage, all the data from a single building block are packed into a single Level 0 product, referred to as a Raw SPIRE Timeline (RST) for each building block, i.e. A single Level 0 product contains many separate Table datasets (one for each telemetry packet produced from the satellite). Clicking on a given number within the Level 0 context reveals the Level 0 Product for that particular building block. These products are the raw data versions of the Level 0.5 data and contain Table Datasets such as the Critical House Keeping timelines (CHK), Nominal House Keeping timelines (NHK), Raw Photometer Detector timelines (PHOTF), Raw BSM timelines (BSNNOMINAL), Raw Photometer Offset timelines (PHOTOFF) and Sub-Control Unit timelines (SCUNOMINAL).

The Raw Photometer Detector Timeline (PHOTF) Table Dataset can be viewed by right-clicking and selecting Open-with-Dataset Viewer, see Figure 6.29, we find quite a different structure to the Level 0.5 PDT datasets. There are 288 columns, one for every SPIRE channel, numbered not in the familiar PSWE8, PSWE9 notation but rather as PHOTFARRAY001 -- PHOTFARRAY288 which corresponds to their Channel Number (from an electrical designation). The signal is still in raw ADU and there are many different time columns which correspond to various measures of the data frames, telemetry packets and packet sequence counts, etc. The only flags are contained in the PHOTFAD-CFLAGS column which is set in the case of a problem with ADC process in telemetry. A full description of the data structure can be found in the Products Definition Document (HERSCHEL-HSC-DOC-0959) or the SPIRE Pipeline Description Document (SPIRE-RAL-DOC-002437).
6.3. Memory Requirements

As can be expected for a robust data reduction and analysis software system such as HIPE, there are minimum software requirements for running the user processing scripts.

For the photometer user scripts, we expect that since the mapping pipelines never have any more than one product in memory at a time, there are no processing memory issues related to the number of repetitions. We recommend that for processing of the largest scan map/parallel mode observations with the relevant user processing scripts, the minimum memory recommendations are 2GB for large/small scan mapping and 4GB for parallel mode. See section 9.2 on a method to overcome memory problems in HIPE.

6.4. Processing SPIRE Observations with the 2-Pass Pipeline

6.4.1. The SPIRE 2-Pass User Pipeline Script

6.4.1.1. Introduction and Motivation

For versions of HIPE 12 and earlier, SPIRE observations have been processed by dedicated pipeline scripts for each observation mode, for Large Map and Parallel mode (described in Section 6.5) and
Small Map mode (described in Section 6.6). However, from HIPE version 13 a more advanced processing pipeline, referred to as the SPIRE 2-Pass Pipeline has been implemented.

The basic pipeline processing tasks and the algorithms follow the original pipeline descriptions and the User is referred to Section 6.5 for a full description of the pipeline tasks. Note that the original Large Map, Parallel and Small Map scripts described in Section 6.5 and Section 6.6 are still valid and will still produce maps of science grade quality.

The motivation for the 2-Pass pipeline is to produce exceptionally cleaned timelines for maps to be used in the final Herschel Science Archive. The 2-Pass Pipeline addresses residual faint glitches and tails that remain in the timeline data that can also produce ringing effects. The main aim of the 2-Pass Pipeline is to improve outlier detection (glitch, glitch tail, signal jump) and any Fourier ringing that may result from failed outlier detection, in order to produce higher quality final maps (see Figure 6.30 where the faint glitch tail has been successfully removed by the 2-Pass Pipeline). The 2-Pass Pipeline achieves this by running a First-Pass stripped down pipeline of the bare minimum of pipeline tasks (especially without any Fourier analysis tasks) and running the Second Level Deglitching to mask glitches. These 2ndLevelGlitch masks are then transferred back to the Level 0.5 products. Then a Second-Pass of the pipeline is made the same as the standard Level 0.5 - Level 1 processing, as performed in HIPE12 and previous versions.

A schematic overview of the SPIRE 2-Pass Pipeline is shown in Figure 6.31. For full details of the Level 0.5 to Level 1 and Level 1 to Level 2 pipeline processing the User is referred to Figure 6.34, Figure 6.35 and Section 6.5.

![Figure 6.30. Improvements with the SPIRE 2-Pass Pipeline](image-url)
Figure 6.31. The SPIRE Photometer 2-Pass Pipeline.

The SPIRE 2-Pass Pipeline can be accessed by clicking on Pipeline on the top bar within HIPE, selecting 'SPIRE' and then clicking on Photometer Two Pass user pipeline - the script will open up in the Editor window within HIPE shown in Figure 6.32.

Figure 6.32. Selecting the Photometer 2-Pass User pipeline script within HIPE

6.4.1.2. Running the SPIRE 2-Pass pipeline

The 2-Pass User Pipeline allows us to reprocess our data from Level 0.5 to Level 1. The 2-Pass pipeline script requires some inputs before it can be executed. The pipeline requires an Observation ID (in either hexadecimal or decimal), optional Data Pool and an output directory to write the final maps as FITS files from the pipeline similar to the examples provided in the script below:

```python
myObsid    =  1342189512  # 0x50002FC8 in decimal
myDataPool = "OD250-ScanNGC5315-1342189512"
outDir     = "/Users/cpearson/jython/localstore/plots/"
```

An observation can either be read from a Pool in your Local Store;

```python
obs=getObservation(myObsid,poolName=myDataPool,instrument="SPIRE")
```
or from the Herschel Science Archive;
In addition, there are up to 5 global processing environment options as Boolean parameters that may be set. These are described in the pipeline script (see code excerpt below) and are the choice of whether to save the final maps to FITS files (saveMaps), to use virtual storage if memory is an issue for large observations (tempStorage) and whether to include turnaround data at the ends of the scan lines for increased map coverage (includeTurnaround). The User may also specify whether to create extended calibrated maps in MJy/sr in addition to the nominal point source calibrated Jy/beam maps (createExtdMaps). Note that in order to do this, the two Planck HFI-545 and HFI-857 full-sky maps need to be downloaded from the HSC website (See Section 6.10.1). Finally there is an option for observations of moving objects to create maps in the moving object rest frame (createSSOMaps).

The 2-Pass Pipeline allows some limited options to be set for the different processing tasks. All tasks and associated options are explained in detail for the original SPIRE pipelines in Section 6.5. The 2-Pass Pipeline allows optional changes to be made to the 2nd Level Deglitching, Destriping, Map Making and Zero Point Absolute Calibration steps of the pipeline as shown below:

```python
# Deglitching Parameters
l2DeglitchRepeat = 100
kappa = 5.0
kappa2 = 5.0

# Destriping Parameters
offsetFunction = "perScan"
polyDegree = 0
withMedianCorrected = True
nThreads = 2
jumpThresh = -1.0
jumpIter = 100
brightSourceThresh = 1.5
roi = 0

# Mapmaking Parameters
pswSize = 6.0  # Recommended map pixel size for PSW
pmwSize = 10.0  # Recommended map pixel size for PMW
plwSize = 14.0  # Recommended map pixel size for PLW
minVel = 5.0  # Recommended min scan velocity to be included in map

# Extended Emission map creation Planck ZeroPointCorrection Parameters
# The Planck Maps are required in order to produce the maps calibrated for extended emission
# change Paths to point to Planck maps on YOUR local disk
hfi545Gain = zeroPointCorrection["hfi545Gain"]  # Recommended gain for Planck HFI 545GHz channel
hfi857Gain = zeroPointCorrection["hfi857Gain"]  # Recommended gain for Planck HFI 857Hz channel
hfiFwhm = zeroPointCorrection["hfiFwhm"]  # Recommended Planck HFI FWHM
```

The 2-Pass Pipeline is then executed in a single line with a call to the pipeline task containing all optional parameters;
#*************************************************************************
# RUN THE 2 Pass PIPELINE
spirePhotPipeline(obs, cal=cal, execEngConv=execEngConv, \
    tempStorage=tempStorage, includeTurnaround=includeTurnaround, \
    createExtMaps=createExtMaps, createSSOMaps=createSSOMaps, \
    l2DeglitchRepeat=l2DeglitchRepeat, kappa=kappa, kappa2=kappa2,\
    offsetFunction=offsetFunction, polyDegree=polyDegree, \
    withMedianCorrected=withMedianCorrected, nThreads=nThreads, \
    jumpThresh=jumpThresh, jumpIter=jumpIter, \
    brightSourceThresh=brightSourceThresh, \
    pswSize=pswSize, pmwSize=pmwSize, plwSize=plwSize, \
    minVel=minVel, createHiResMaps=createHiResMaps, \
    hfi545Gain=hfi545Gain, hfi857Gain=hfi857Gain, hfiFwhm=hfiFwhm)
#*************************************************************************

Note that the default output from the User Pipeline are maps as FITS files saved in the user defined outDir path. It is also possible to write the entire observation back to the original or a new Pool. To do so, the following line at the end of the script must be uncommented and an appropriate Pool name inserted.

#saveProduct(product=obs, pool='enter-new-pool-name')

---

6.5. Recipe for Photometer Large Map and Parallel Mode

---

6.5.1. Large Map and Parallel Mode User Pipeline Script

6.5.1.1. Prerequisites

The Large Map mode is essentially the same as the SPIRE component of the Parallel Mode - for both modes, this processing guide will allow you to reprocess your data. For this data reprocessing example, we assume that you wish to reprocess your data starting from Level 0.5 products. For this data reprocessing example, we will be using the Large Map observation (obsID: 1342183475) of NGC 5315. We will in this example assume that you have received the engineering pipeline processed Level 0.5 data products from the HSC, and have stored them in a storage pool "OD117-ScanNGC5315-0x50001833", either by a direct download or through HIPE.

You can access the Photometer Large Map User Pipeline processing script by clicking on 'Pipeline' on the top bar within HIPE, selecting 'SPIRE' and then clicking on 'Photometer Large Map User Pipeline' - the script will open up in the Editor window within HIPE.
Figure 6.33. Selecting the Photometer Large Map User Pipeline pipeline script

6.5.1.2. User Pipeline Inputs

The User Pipeline Script allows us to process our data from Level 0.5 to Level 1. The pipeline script requires some editing before it can be executed. The pipeline requires an Observation ID (in either Hexadecimal or decimal), Data Pool and an output directory to write plots and FITS files from the pipeline similar to the examples provided in the script below:

```
myObsid    = 0x50001833   # 1342183475 in decimal
myDataPool = "OD117-ScanNGC5315-0x50001833"
outDir     = "/Users/cpearson/jython/localstore/plots/"
```

In addition, there are up to 6 optional parameters that may be set. These are described in the pipeline script (see code excerpt below) and are the choice of whether to include turnaround data at the ends of the scan lines for increased map coverage (includeTurnaround Boolean parameter), whether to create maps absolute calibrated for extended emission (MJy/sr) (makeExtendMaps Boolean parameter), the choice of baseline subtraction (the default is the destriper or alternatively a median baseline subtraction) and flags to turn on the optional "Bolometer Jump" and "Cooler Burp" correction (See Section 6.5.2.12).

```#
# Additional Options
# (D) includeTurnaround: Include the scan line turnarounds in the processing and mapmaking
# (E) makeExtendMaps: Create absolute calibrated maps in MJy/sr
# (F) baselineSubtraction: Subtract a baseline from each scan to avoid stripes
# (G) destriper: Determine and remove baselines to achieve an optimum fit between all timelines
# (H) jumpDetection: Detect bolometer jump with destriper and flagged scan are not used in map making process
# ** At least one of the options F or G must be True.
# (I) coolerBurpCorrection: Search and correct cooler burp.
includeTurnaround          = False
makeExtendMaps             = False
useBaselineSubtraction     = False
useDestriper               = True
bolometerJumpDetection     = False
coolerBurpCorrection       = False
```

Note that the default output from the User Pipeline are maps as FITS files saved in the user defined `outDir` path. It is also possible to write the entire observation back to the original or a new Pool. To do so, the following line at the end of the script must be uncommented and an appropriate Pool name inserted.

```
#saveObservation(obs,poolName="enter-a-poolname",saveCalTree=True)
```
6.5.1.3. Level 0.5 to Level 1 Processing

The Large (Parallel Mode) Map pipeline is shown in the flowchart in Figure 6.34. The flowchart covers the processing steps from Level 0.5 to the creation of the Level 2 maps. Note that as explained previously in Table 6.1, the pipeline produces Point Source Calibrated maps in Janskys/beam. In addition by setting the makeExtendMaps=True, maps absolute calibrated for extended emission (using the Planck data to calibrate) can also be produced. The standard (SPG) archive processing produces by default up to three additional maps and diagnostic products. Maps for extended emission, with absolute calibration derived from the Planck all-sky maps (in MJy/Sr) appear in the Level 2 context in the form of extdPxW. Maps specifically for Solar System 'moving' objects (SSO) calibrated in Jy/beam appear in the Level 2 context in the form of ssoPxW (where PxW corresponds to PSW, PMW or PLW respectively). These SSO maps differ from the normal point source psrcPxW maps in that the SSO maps are motion corrected and centred on the moving object frame (see Section 6.12.1). In certain cases super resolution maps are also created for appropriate observations (see Section 6.11.4).

Note that Destriping using 2nd Level Deglitching should not be performed on observations of SSOs, before the SSO motion correction has been applied, as this will result in misidentifying source samples as glitches. This is checked by default in the SPG pipeline processing at ESAC.

The actual number of explicit pipeline steps required to process a Large Map (& Parallel) mode observation are relatively few, as outlined in the schematic overview of the pipeline script shown in Figure 6.35 (and the essence of these steps is exactly the same for the Small Map pipeline described in Section 6.6.1). The pipeline from Level 0.5 to Level 1 processes data by looping over the scan lines to start building up the map. Each step in the pipeline applies some correction to the data, to produce the final Level 1 pipeline product, as flux calibrated timelines with associated positions. The pipeline works on a Photometer Detector Timeline (PDT) but also requires additional data such as the Nominal Housekeeping Timeline (NHKT) and other auxiliary products for the telescope pointing information.

![Figure 6.34. The SPIRE Photometer Large Map pipeline.](image)

Figure 6.34. The SPIRE Photometer Large Map pipeline.
Figure 6.35. The Essence of the SPIRE Photometer mapping Pipeline (applicable to Large Map, Parallel and Small Map) Script.

In the following section the processing steps are broken up into their constituent parts. The first step in the pipeline is to join all the scan legs and turnarounds together. The timelines are joined together to avoid ringing effects caused by pipeline modules involving corrections using Fourier domain filtering.

```plaintext
# (1) join all scan legs and turnarounds together
bbCount = bbid & 0xFFFF
pdtLead = None
nhktLead = None
pdtTrail = None
nhktTrail = None
if bbCount > 1:
    blockLead = obs.level0_5.get(0xaf000000L+bbCount-1)
    pdtLead = blockLead.pdt
    nhktLead = blockLead.nhkt
    if pdtLead != None and pdtLead.sampleTime[-1] < pdt.sampleTime[0]-3.0:
        pdtLead = None
        nhktLead = None
if bbid < MAX(Long1d(bbids)):
    blockTrail = obs.level0_5.get(0xaf000000L+bbid)
    pdtTrail = blockTrail.pdt
    nhktTrail = blockTrail.nhkt
    if pdtTrail != None and pdtTrail.sampleTime[0] > pdt.sampleTime[-1]+3.0:
        pdtTrail = None
        nhktTrail = None
pdt = joinPhotDetTimelines(pdt, pdtLead, pdtTrail)
nhkt = joinNhkTimelines(nhkt, nhktLead, nhktTrail)
```

The next 2 steps in the pipeline produce the pointing information for the observation. The SPIRE Beam Steering Mechanism is a movable mirror used for jiggling observations and although not used for mapping observations a constant offset must still be taken into account. In order to reconstruct the SPIRE Pointing Product, the BSM Angles Timeline (BAT) has to be added to the Detector Angular Offsets on the array, and to the SIAM which relates the SPIRE instrument to the Herschel Pointing Product (HPP) itself.
The next step is the **Electrical Crosstalk Correction**, which corrects for crosstalk between the Thermistor-bolometer channels only (bolometer-bolometer crosstalk is negligible).

The next step is the **Signal Jump Detector**, which detects jumps in the Thermistor timelines that occasionally occur, leading to map artifacts introduced in the temperature drift correction stage later in the pipeline.

This module subtracts baselines and smoothed medians from the Thermistor timelines to identify any jumps. Any jumps are flagged using the `maskJumpThermistorsDarksSignal` mask bit. The current recommended values for all parameters are included in the call to the module (specific details can be found in the SPIRE Pipeline Specification Manual (PSM)). The module requires the Temperature Drift Correction calibration file. Note that the appropriate version of this calibration file is only available with HIPE version 8 onwards. If an earlier version of HIPE is used then an older version of the Signal Jump Detector is automatically used. For observations of bright sources, the use of this module is not encouraged because of the very low probability of Signal Jumps in the timelines of Dark Pixel channels used by the Temperature Drift Correction module for bright mode observations, therefore at present this step of the pipeline is skipped for observations in bright source mode.

Cosmic ray rejection (deglitching) is one of the most challenging data analysis problems for SPIRE, as artifacts, caused by undetected glitches, limit the calibration accuracy and sensitivity, and directly influence the quality of the final data products. The SPIRE pipeline incorporates a two stage deglitching process described below. The first step is to run the **Concurrent Deglitcher** on the timeline data - this is to detect and remove glitches that occur simultaneously in groups of connected bolometer detectors due to a cosmic ray hitting the substrate of a photometer array. An extreme example of the performance of the Concurrent Deglitcher is shown in Figure 6.36, where the right panel shows the effect of not running the Concurrent Deglitcher on the timeline data. The imprint of the array is clearly seen on the final map.

For each detector, a running median of the signal is computed. The `size` parameter is the half-size of the window on which the median is computed and the `Kappa` parameter is a measurement of...
the glitch detection threshold. The number of samples flagged and replaced depends on the strength of the glitch and the corresponding GLITCH_FIRST_LEVEL flag is raised. Note that the default is for the detected glitches to be reconstructed, however if just detection and flagging is required the reconstruction can be switched off by setting `correctGlitches = False`. For more information on the concurrent deglitcher, see the respective section of the Pipeline Specification Manual.

![Figure 6.36. Performance of the Concurrent Deglitcher.](image)

The second step in the deglitching (referred to as First Level Deglitching) is to run the Wavelet Deglitcher on the timeline data. This module employs a complex algorithm using wavelet transformation in Fourier space and for an in-depth explanation of the parameters that one can pass to the Wavelet Deglitcher, please refer to the relevant entry in the SPIRE Pipeline Specification Manual:

```python
# (7) Run the wavelet deglitcher on the timeline data
pdt=waveletDeglitcher(pdt, scaleMin=1.0, scaleMax=8.0, \
    scaleInterval=7, holderMin=-3.0, holderMax=-0.3, \
    correlationThreshold=0.3, optionReconstruction='linearAdaptive20',\n    reconPointsBefore=1, reconPointsAfter=3)
```

Note that the call is slightly different for Parallel Mode observations due to fine tuning to obtain the optimal set of parameters:

```python
# (7) Run the wavelet deglitcher on the timeline data
pdt=waveletDeglitcher(pdt, scaleMin=1.0, scaleMax=8.0, \
    scaleInterval=5, holderMin=-1.9, holderMax=-0.3, \
    correlationThreshold=0.69, optionReconstruction='linearAdaptive20',\n    reconPointsBefore=1, reconPointsAfter=3)
```

There is one further optional deglitching module that can be included in the pipeline processing replacing the Wavelet Deglitcher. The alternative Sigma-Kappa Deglitcher can be used if problems are found using the standard pipeline deglitching. The deglitcher is present but by default comment-ed out in the pipeline. Details of this module are found in the SPIRE Pipeline Specification Manual.

Next, the **Low Pass Filter Response Correction** is applied to the detector timelines. The electronics impose a delay on the data with respect to the telescope position along the scan and this effect must be taken into account to ensure that the astrometric pointing timeline is properly matched to the detector data timeline. The call to the module is:

```python
# (8) Apply the Low Pass Filter response correction
pdt=lpfResponseCorrection(pdt,lpfPar=lpfPar)
```
Note that to reduce ringing effects in the timeline data (caused by the pipeline modules using Fourier domain filtering), all the individual timelines are concatenated together with the turnarounds between scan lines, at the beginning of the pipeline. However, the last scanline of an observation is processed without turnaround data. Therefore this last scan line is prone to show edge ringing effects from the two Fourier pipeline modules (Low Pass Filter Response and Bolometer Time Response correction) run in the Level 0.5 to Level 1 processing. This effect is most prominent in timelines with a large slope. Although the overall effect of this feature on the map is almost negligible, it will effect pixels in the maps hit by the last samples (although these pixels are located at the border, which is well outside the user-requested area). The edge ringing effects can be overcome by a method that mirrors the signal in the timelines effectively removing any slope from the data. In Figure 6.37 the effect of mirroring the timelines to remove edge ringing is shown. Large maps processed with and without the mirroring option are shown along with the corresponding difference map. Several artefacts in the difference map can be seen. The speckled noise pattern at the edges of the scans (single pixel, white and black points) is due to one of the maps having high frequency ringing at the edges. The other effect is the array imprints that are visible as smudges, which are undetected common glitches. These imprints are more common at the edges of the scan where the lack of ringing in the second map, affects the final (post temp-drift correction) slope of the timelines. Also shown in Figure 6.37 are example plots of the final scan line timeline data showing the ringing in the first panel, a comparison of the scan line with and without the mirroring and the difference between the unmirrored and mirrored timelines respectively.

![Figure 6.37. Edge ringing in the final scan line.](image)

Timeline mirroring can be turned on in the pipeline by supplying an additional option to the `lpfResponseCorrection` task;

```java
pdt = lpfResponseCorrection(pdt, lpfPar=lpfPar, doMirrorSignal=Boolean.TRUE)
```

The ringing is more severe, for larger differences between the points at edges of the timeline. Such large differences are more likely with longer scan lengths (i.e. more likely to be seen in Parallel Mode observations) and also are more likely to be observed at times were the bath temperature changes rapidly (for example during a cooler burp). For most Scan Map observations the default method should suffice. The cases where the `mirror` method would be advantageous include: (1) Slopes in the timeline due to real structure in the field. For example a Small Scan Map with cirrus emission on one side of the map. The timelines of such a field would have a slope and correcting them with the default method would create ringing. (2) Slopes in timeline due to bath temperature drift. In this case, the longer the scan-length, the larger the difference between the start and the end of the timeline will be. Such bath temperature drifts are common during the first few hours after a cooler recycle. Note that in general, using the `mirror` option will always produce slightly better results, however the corresponding time to run this module will be doubled.
The next step is to apply the **Flux Conversion** to the detector timelines to convert the voltage timelines into timelines in the units of Jy/beam. A calibration file containing the appropriate Astronomical Unit Conversion Table for either normal or bright source mode is used:

```plaintext
# (9) Apply the flux conversion
fluxConv=fluxConvList.getProduct(pdt.meta["biasMode"].value,pdt.startDate)
pdt=photFluxConversion(pdt,fluxConv=fluxConv)
```

The next step is optional and set by the `coolerBurpCorrection = True` flag at the beginning of the User Pipeline Script to deal with artefacts in maps caused when a "cooler burp" occurs. During the cooler burp the drifts of bolometers are unusual and for this reason the temperature drift correction leaves residuals. Cooler burps are detected by checking for large variations in the subK Temperature parameter in the SPIRE Housekeeping data. If this variation is above some threshold then a flag is set in the meta data. This flag is checked at the Temperature Drift Correction stage of the pipeline (see Section 6.5.2.12 for details):

```plaintext
# (10) If coolerBurpCorrection flag = True then update timeline meta data
pdt.meta["coolBurpDetect"]=BooleanParameter(coolerBurpCorrection, \
    "Indicates a cooler burp")
```

The next step, the **Temperature Drift Correction**, removes low frequency noise, caused by variations of the detector array bath temperature, from the timelines. A correction timeline is generated for each detector using data and calibration information for that detector. This is then subtracted from that detector’s signal timeline. The module requires the Temperature Drift Correction calibration file used earlier in the Signal Jump Detector module. Note that if the `coolerBurpFound` flag is set then the Temperature Drift Correction task will apply additional multiplicative factors to its coefficients to correct for the cooler burp artefacts in the map:

```plaintext
# (11) Make the temperature drift correction
pdt=temperatureDriftCorrection(pdt,tempDriftCorr=tempDriftCorr)
```

The next step is the **Bolometer Time Response Correction** which corrects for any additional low-level, slow response in the SPIRE bolometers. This is accomplished by multiplying the signal in Fourier space by an appropriate transfer function, requiring a calibration file containing the corresponding detector time constants:

```plaintext
# (12) Apply the bolometer time response correction
pdt=bolometerResponseCorrection(pdt,chanTimeConst=chanTimeConst)
```

The next step in the Level 0.5 to Level 1 processing is to **Cut The Timelines** back into individual scan lines. However, if you want to include turnaround data in map making, the `includeTurnaround` parameter should be set to `TRUE` at the beginning of the pipeline script:

```plaintext
# (13) Cut the timeline back into individual scan lines.
pst=cutPhotDetTimelines(pst,extend=includeTurnaround)
```

The penultimate step in the pipeline is to attach positional information (RA, Dec) to the data timelines, to create the Photometer Scan Product (PSP). The **Associate Sky Position** module attaches the sky position timeline onto the detector timeline by querying the SPIRE Pointing Product that contains within it the necessary products to determine the positions of the SPIRE detectors on the sky, the Herschel Pointing Product, SIAM Product, the BSM Angles Timeline and the Detector Angular Offset table (all described above at the beginning of this pipeline section).
Finally, we apply the **Time Correlation** to the PSP which corrects for any drifting of the clock that is on board the spacecraft:

```python
# (15) Apply the time correlation
psp=timeCorrelation(psp,timeCorr)
```

and we shall store our Photometer Scan Product in the Level 1 context, which brings to a close the processing of the individual scan lines (but not yet the end of the Level 0.5 to Level 1 processing):

```python
# Add Photometer Scan Product to Level 1 context
level1.addProduct(psp)
```

At this stage, the data is at the intermediate Level 0.7 stage. To complete the processing to Level 1, baseline removal (destriping) must be carried out on all the processed calibrated positional timelines.

Due to the large telescope background, fluxes measured by the SPIRE bolometers are effectively very small differences on top of a dominating offset that is usually several orders of magnitude larger. Due to variations in the thermal and electronic stability of the system, residual offsets in the flux calibration from one detector to another are observed, resulting in striping in the final maps. The default algorithm used by the User Pipeline to remove such striping is an **Iterative Destriping Algorithm** that iteratively updates offsets in the timelines until an optimal solution is found. An alternative option is the Median Baseline Removal algorithm, which subtracts the median from each detector in the Level 1 timelines and then creates a map. Users are encouraged to read Section 6.8.1 for details on the Destriper and other various baseline removal methods. The two methods in the pipeline can be inter-changed by use of the Boolean flags at the beginning of the pipeline:

```python
useBaselineSubtraction = False
useDestriper = True
```

The Destriper module takes as input a SPIRE Level 1 context (scans) and outputs a list of destriped scan lines as input to the mapping algorithm. Note that the Destriper also produces maps itself using the naive mapper, and these can be accessed via the variable `maps` below. Please see Section 6.8.1 for details of the Destriper parameters.

The Destriper also supports 2nd Level deglitching via outlier rejection, however, this feature is currently switched off by default by the parameter `l2DeglitchRepeat=0`. To perform the level 2 deglitching, the destriper creates an additional temporary map of Medians, i.e. each pixel contains the median value of all unmasked (valid) readouts that fall within the borders of that skybin, and a map of Median Absolute Deviations (MAD), i.e. each pixel contains the MAD of all unmasked (valid) readouts that fall within the borders of that skybin. Glitches are identified as outliers between the median map and the MAD map times some threshold ($\kappa$). Glitches are then masked and the new map created by the naive mapping process. In the case of small number statistics, i.e. when few readouts are in a skybin, the threshold provided by the MAD map may become unrealistically low. To alleviate this problem, a second parameter ($\kappa2$) is allowed, as a lower threshold for 2nd level deglitching. To activate the 2nd Level Deglitching the following parameters in the destriper call should be set (to the recommended values): `l2DeglitchRepeat = 100, l2IterMax = 1, kappa=7, kappa2=7` (see Section 6.8.1 for more details).

Note that for very large maps the destriper requires more than 7Gb of memory which may cause memory problems on smaller machines. Memory problems can be overcome by setting the `useSink=True` parameter in the Destriper call below (Default is `useSink=False`). Note also that the destriper removes the relative offsets between individual scan lines which means that negative fluxes can still
exist in the final maps. For absolute flux calibrated maps, the User should use the Extended Emission Maps processed using the Planck All-Sky Maps to produce an absolute zero-point (see Section 6.1).

### Destriping ###

if useDestriper:
    print
    print "Destriper Run for OBSID= %i, (0x%x)"%(myObsid,myObsid)
arrays = ["PSW","PMW","PLW"]
pixelSize = [6,10,14]  #Map pixel size in arcsec for PSW, PMW, PLW respectively
maps = []
#
if bolometerJumpDetection:
    jumpThresh=8
else:
    jumpThresh=-1
#
# Using Level 1 context. Run destriper as an input to the map making
for iArray in range(len(arrays)):
    scans,map,diag,p4,p5 = destriper(level1=scans,
    pixelSize=pixelSize[iArray], offsetFunction='perScan',
    array=arrays[iArray], polyDegree=0, kappa=20.0, iterThresh=1.0E-10,
    l2DeGlitchRepeat=0, iterMax=100, l2IterMax=5, 
    nThreads=2, jumpThresh=jumpThresh, jumpIter=15, 
    withMedianCorrected=True, brightSource=True, useSink=False, storeTod=False)
#
# Save diagnostic product
if obs.level2.refs['pdd'+arrays[iArray]]!=None:
    obs.level2.refs.remove('pdd'+arrays[iArray])
obs.level2.setProduct('psrc'+arrays[iArray]+'diag', diag)
#
# Keep destriper maps for inspection
maps.append(map)
print "Finished the Destriper Run for OBSID= %i, (0x%x)"%(myObsid,myObsid)
print
### Finished Destriping ###

6.5.1.4. Level 1 to Level 2 Processing (Mapmaking)

The standard mapping algorithm in the User Pipeline is the Naive Mapper. The Naive Mapper data simply projects the full power seen by a detector onto the nearest sky map pixel. For each bolometer timeline at each time step, the signal measurement is added to the total signal map, the square of the signal is added to the total signal squared map, and 1 is added into the coverage map. After all bolometer signals have been mapped, the total signal map is divided by the coverage map to produce a flux density map, and the standard deviations are calculated using the total signal, total signal squared, and coverage map.

print 'Starting Naive Map maker'
mapPlw=naiveScanMapper(scans, array="PLW", method=UnweightedVariance)
mapPmw=naiveScanMapper(scans, array="PMW", method=UnweightedVariance)
mapPsw=naiveScanMapper(scans, array="PSW", method=UnweightedVariance)

Although the current recommendation is the Naive Mapper. The alternative is the Madmap algorithm which is a maximum-likelihood based method of estimating a final sky map from the input data. The Madmap algorithm can be run with the following code but requires the Channel Noise Table Calibration Product

print 'Starting Mad Mapper'
mapPlw=madScanMapper(scans, array="PLW", chanNoise=chanNoise)
mapPmw=madScanMapper(scans, array="PMW", chanNoise=chanNoise)
mapPsw=madScanMapper(scans, array="PSW", chanNoise=chanNoise)

Error Maps

Several options for the error map associated with the naive mapmaker are available. The mathematical formulae for all the error maps below are explained in the SPIRE Pipeline Specification Manual (PSM). The default error map UnweightedVariance treats all unflagged detectors equally in averages (no weighting) and the error map gives the standard deviation of the mean of samples falling in a map pixel. Therefore, the unweighted averages fall short of achieving maximum possible S/N. An additional error map is provided by the following alternative call and requires the Channel Noise Table Calibration Product since the errors are weighted by the individual bolometer noise measurements.

map = naiveScanMapper(scans, array=array, method=WeightedVariance, chanNoise=chanNoise)

For areas on maps with very low coverage the errors may not be accurate using the weighted method. Therefore a third Hybrid error map is available which uses the above weighted error map, except at very low sample counts. For these low-sample pixels, the observed signal values are not used in estimating the error flux - instead it is calculated from the white-noise only:

n = 5  # Hybrid error map low sample threshold
map = naiveScanMapper(scans, array=array, method=WeightedVariance, chanNoise=chanNoise, hybridMap = True, hybridThreshold = n)

Median Maps

There is also the option to produce a Median Map with the Naive Mapmaker using the MedianMap keyword;

mapPswMedian = naiveScanMapper(scans, array=array, method=MedianMap)

With this keyword set, the Naive Mapmaker will produce the following:

• A map of medians, i.e. each pixel contains the median value of all unmasked (valid) readouts that fall within the borders of that skybin.

• A map of median absolute deviations (MAD), i.e. each pixel contains the MAD of all unmasked (valid) readouts that fall within the borders of that skybin.

• A map of coverage, i.e. each pixel contains the number of all unmasked (valid) readouts that fall within the borders of that skybin.

The output is equivalent to the default Naive Mapper output, except that the signal map is replaced by the median map and the error map is replaced by the median absolute deviation map (such median maps are required (and produced internally) during the 2nd level deglitching process for outlier rejection).

Turnaround Data

If the includeTurnaround = True flag has been set at the beginning of the pipeline then the final map will include the turnarounds at the end of the scan line legs. In addition, it is possible to further control the amount of turnaround data included by setting optional keywords in the naive mapmaker. The User may specify a maximum and minimum velocity (in arcsec/s) to be included as shown below. The recommended minimum velocity is 5 arcsec/s and this is the default.

map = naiveScanMapper(scans, array="PSW", minVel=10, maxVel=100)
Masks

The mapper excludes samples from entering the map according to their mask. Depending on the mask there are two different approaches implemented.

- The DEAD, SLOW and NOISY mask bit is checked only once at the start of the scan line for each bolometer. If set, the mapper will ignore all the samples in the scanline for this bolometer, i.e. they will not end up in the map.

- For the MASTER, GLITCH_FIRST_LEVEL_UNCORR, GLITCH_SECOND_LEVEL_UNCORR, SERENDIPITY and TRUNCATED_UNCORR masks the mapper checks for each sample individually. If the mask is set the sample will be ignored. For more details on SPIRE mask handling, see Section 8.4 of this manual.

Saving the Final Maps

The Final maps are written out as FITS files to the directory specified at the beginning of the pipeline script;

```python
# Save Maps to output directory
simpleFitsWriter(mapPsw, "%smapPSW_%i.fits"%(outDir, myObsid))
simpleFitsWriter(mapPmw, "%smapPMW_%i.fits"%(outDir, myObsid))
simpleFitsWriter(mapPlw, "%smapPLW_%i.fits"%(outDir, myObsid))
```

Finally, the Level 2 Context for the observation is updated with the maps (For post HIPE 10 processing the Point Source Calibrated maps are referred to using the `psrc` prefix);

```python
obs.level2.setProduct("psrcPSW", mapPsw)
obs.level2.setProduct("psrcPMW", mapPmw)
obs.level2.setProduct("psrcPLW", mapPlw)
```

Note that it is also possible to save the entire Observation Context to either the same Pool or a new Pool by uncommenting the `saveObservation` command in the User Script and supplying a Pool name.

```python
#saveObservation(obs,poolName="enter-a-poolname",saveCalTree=True)
```

Three point source calibrated maps are each produced for PSW, PMW and PLW, and are visible through the Product Viewer by right-clicking on the required variable in the Variable pane and selecting 'Open With':

![Figure 6.38. Selecting the Product Viewer](image-url)
the actual map with fluxes (denoted as 'image', see Figure 6.39);

Figure 6.39. The NGC 5315 PSW Level 2 image map

The statistical flux error map, calculated as the **Standard Deviation** using the total signal, total signal squared, and coverage map (denoted as 'error' see Figure 6.40):

Figure 6.40. The NGC 5315 PSW Level 2 error map

and an image which shows the coverage map for our scans (denoted as 'coverage' see Figure 6.41):

Figure 6.41. The NGC 5315 PSW Level 2 coverage map

We can export our images to FITS files by right clicking on the respective variable in the Variable pane (e.g. mapPsw), and selecting ‘Send To -> FITS file’.

Congratulations! You have now re-processed your Large Map data all the way to the final Level 2 maps!
6.5.1.5. Creating Absolute Calibrated Maps for Extended Emission

For observations where producing maps of extended emission is the most important objective, additional processing can be carried out to produce maps absolutely calibrated for extended emission in MJy/sr. The absolute calibration is made using the zero-point correction obtained by the Planck maps. The creation of absolute calibrated maps using the Planck zero-point is described in detail in Section 6.10.1. In order to carry out the processing for extended emission maps, it is assumed that the Planck maps have already been downloaded and the correct properties have been set for the gains for the Planck HFI 545GHz and 857GHz bands respectively (see Section 6.10.1.3 for details). Before the maps for extended emission are created the pipeline sets up the following environment for the Planck map zero-point correction:

```python
if makeExtendMaps:
    # Planck Map Setup
    hfi545Map = Configuration.getProperty("spire.spg.hfi.545map")
    hfi857Map = Configuration.getProperty("spire.spg.hfi.857map")
    hfi545Gain = zeroPointCorrection["hfi545Gain"]  # Planck HFI 545GHz channel gain
    hfi857Gain = zeroPointCorrection["hfi857Gain"]  # Planck HFI 857Hz channel gain
    hfiFwhm    = zeroPointCorrection["hfiFwhm"]     # Planck HFI FWHM

    # Prepare zero-point correction task
    level1 = obs.level1
    level2 = obs.level2
    level2ZeroPoint = MapContext()
    for key in level2.meta.keySet():
        level2ZeroPoint.meta[key] = level2.meta[key].copy()
    # Create new Level1Context
    scansZeroPoint = Level1Context()
    scansZeroPoint.meta = level1.meta
```

In order to produce maps calibrated for extended emission, an additional processing step; Extended emission gain correction is required for individual bolometers since the SPIRE calibration assumes uniform beams across the array, not taking into account variations among bolometers. For the creation of maps with extended emission, inclusion of both the scan turnaround data and the application of the relative gains is strongly recommended. These are implemented in the pipeline by setting the following parameters at the beginning of the pipeline:

```python
includeTurnaround = True
makeExtendMaps    = True
```

Then the relative gains are applied (using the chanRelGains calibration product) as;

```python
# Optionally apply Relative Bolometer Gains for extended emission#
if applyExtendedEmissionGains:
    print
    print "Apply relative gains for bolometers for better extended maps"
    for i in range(level1.getCount()):
        psp = level1.getProduct(i)
        psp = applyRelativeGains(psp, chanRelGains)
        scansZeroPoint.addProduct(psp)
```

Note that for observations processed earlier than HIPE version 8, an additional correction must be made to the data before the relative gains can be applied, due to a bug in earlier versions of the data products.
if psp.type=='PPT':  psp.setType('PSP')

The simpler and stronger recommendation is simply to re-process the observation with a version of HIPE of 8 or later with the User Pipeline with the correction for relative gains switched on.

Note that maps with the Extended emission gain correction are optimized for extended emission and should not be used for point source photometry as the relative extended gain correction causes increased scatter (error) in the photometry if the amplitude is used for measurement. Normal gains are best for photometry of point sources in which it is the amplitude (peak height) of the source which is used for the photometry measurement. However, since the gain correction is better for mapping extended emission, aperture photometry (capturing most of the point source emission) should be improved with it.

After the relative gains have been applied, the timelines must again be destriped. However, the pipeline tries to make this process more efficient by using the previous destriper diagnostic products as a starting point.

```python
# Try to load the de-striper diagnostic products to speed-up re-processing
arrays = ['PSW','PMW','PLW']
for array in arrays:
diagref = level2.refs['psrc'+array.upper()+'diag']
if diagref != None:
diag = diagref.product
else:
diag = None
#
# (Re-)run destriper on new Level1Context
newscans,mapZero,diagZero, p4,p5 = destriper(level1=scansZeroPoint, array=array,
nThreads=2, 
    withMedianCorrected=True, useSink=True, startParameters=diag)
#
# Save diagnostic product, this time with prefix extd, into the "level2" variable
level2ZeroPoint.refs.put(array,ProductRef(mapZero))
level2.refs.put('extd'+array.upper()+'diag', ProductRef(diagZero))
#
```

The maps output from the Destriper are then passed to the zeroPointCorrection task in the pipeline (see Section 6.10.1) to produce the final absolute calibrated maps in MJy/sr.

```python
# Run the zeroPointCorrection tasks on extdPxW maps
print "Running the zero point correction task"
zeroPointMaps, zeroPointParam=zeroPointCorrection(level2=level2ZeroPoint, 
    hfiFwhm=hfiFwhm, hfi545Gain=hfi545Gain, hfi857Gain=hfi857Gain, 
    colorCorrHfi=colorCorrHfi, fluxConv=fluxConv, 
    hfi545Map=hfi545Map, hfi857Map=hfi857Map)
#
```

Finally the pipeline script writes the extended maps to FITS files a described earlier in Section 6.5.1.4 and updates the Level 2 context to produce the final fully processed observation.

### 6.5.1.6. Saving Products from the User Scripts to Local Pools

The SPIRE User Reprocessing Scripts are written to save the final products of the pipeline to FITS files. However, it may be preferable to save the products to local pools instead. This section describes how to do this for both the photometer and spectrometer scripts. The options for saving the reprocessed products from the scripts are:

- Save to FITS file (as described above).
• Save final products only into a new local pool.

• Save back into existing Observation Context.

• Change the existing Observation Context and save to a new local pool.

When saving to an existing pool, there is always the choice whether to overwrite the contents, or to create new versions of them without removing the originals.

**Saving final products only**

The final Level-2 products can easily be saved to a new pool instead of a FITS file by replacing the `simpleFitsWriter` task with (e.g. for Photometer scan map for PSW band)

```python
saveProduct(mapPsw, pool="myPoolName", tag="Map made with user script")
```

**Saving products back into the Observation Context**

The SPIRE User pipelines **automatically update** the Observation Context and the User does not need to do anything.

We include a discussion on updating the Observation Context for **information purposes only**. In order to save the products back into the Observation Context, its structure needs to be taken into account. A reminder of the Photometer Observation Context layout for the Level 1 and Level 2, and how this relates to variables in the User Scripts is given below in Table 6.4.

**Table 6.4. Summary of Photometer Observation Context**

<table>
<thead>
<tr>
<th>Level</th>
<th>Sub-context name</th>
<th>Product Type</th>
<th>variable in User Script</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 1</td>
<td>..</td>
<td>PSP</td>
<td>scans</td>
</tr>
<tr>
<td>Level 2</td>
<td>&quot;psrcPSW&quot;</td>
<td>PMP</td>
<td>mapPsw</td>
</tr>
<tr>
<td>Level 2</td>
<td>&quot;psrcPMW&quot;</td>
<td>PMP</td>
<td>mapPmw</td>
</tr>
<tr>
<td>Level 2</td>
<td>&quot;psrcPLW&quot;</td>
<td>PMP</td>
<td>mapPlw</td>
</tr>
</tbody>
</table>

When saving back into the observation context, it is important to get the sub-context names correct so that the products end up in the right place.

**Level 1 Products:** Saving the Level-1 products into the Observation Context is easy for the Photometer scripts – for example, the following statement can be added at the end of the script:

```python
obs.level1 = level1
```

**Level 2 Products:** For the Level-2 products, the following code lines are an example of what should be added at the end of the script where the final FITS files are written:

```python
obs.level2.setProduct("psrcPSW", mapPsw)
```

**Saving the Observation Context to disk**

By far the simplest method, and the recommended method to save the entire Observation Context to either the same Pool or a new Pool is using the `saveObservation` command in the User Script (supplying a Pool name);
In the rare case where the User may want to have more control over the saving process a more detailed method is explained below. Instead of using the simple `saveObservation` command, the following line should can added at the end of the script:

```python
localStoreWriter(obs, "myPoolName")
```

If the Observation Context is saved back into the same pool as it was read from, the default behaviour is to create new versions of the products that have changed (rather than overwriting). Only products that have changed will actually be saved to disk as new versions. It is possible to overwrite rather than creating new versions – this is possible by modifying the HIPE property; `hcss.ia.pal.pool.lstore.version`. The possible values are and the effect they have on the files that are saved to disk are:

- **new**: create a new version by appending a timestamp to the name (default)
- **overwrite**: overwrite the existing file with the same name; no timestamp is added
- **error**: produce an error if a file with the same name already exists; no timestamp is added

The property can be set initially in the `hipe.props` file (located in your .hcss directory). Once it is set, it can be modified in the properties panel inside HIPE (go to Edit > Preferences > Advanced..) It should also be possible to modify the behaviour at the pool level of granularity. This can be specified with property names of the form, `hcss.ia.pal.pool.lstore.<pool-name>.version`.

Note that it can be dangerous to change this property away from the default value because it affects all operations that write data to pools. For example, the photometer SPG pipeline writes out temporary files and will go wrong if the property is set to **overwrite**. Changing to overwrite files should be done with great care!

### 6.5.2. Large / Parallel Mode Map Troubleshooting and Tips

#### 6.5.2.1. Introduction

Most of the SPIRE maps will look very good as they come out of the pipeline. However, problems can occur and this Chapter gives guidelines in interpreting SPIRE photometer maps, possible issues and how users could overcome them with interactive analysis. Future versions of HCSS will most likely automatically solve most of the issues presented here. In most of the following examples, we consider Large Map mode (POF5), however, the discussion equally also applies to Parallel (POF9) and Small Map (POF10) mode.

If maps look strange or not what was expected, obvious initial checks would be to check that the Level 2 processing was reached for all bands (PSW/PMW/PLW), that the astrometry of maps matches that in the meta data / original HSPOT input, etc. The quality flags can also be examined. In Table 6.5 a thumbnail summary of common map artefacts is shown with links to detailed explanations.
6.5.2.2. Overview of example map problems

Figure 6.42. Thumbnail Summary of Map Artefacts.

Table 6.5. Summary of Map Artefacts

<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Single Scan Observation (Described in Section 6.5.2.5)</td>
</tr>
<tr>
<td>2</td>
<td>The Effect of a Bright Source (Described in Section 6.5.2.6)</td>
</tr>
<tr>
<td>3</td>
<td>Stray Light in the Maps (Described in Section 6.5.2.7)</td>
</tr>
<tr>
<td>4</td>
<td>Failure of the Baseline Removal (Described in Section 6.5.2.8)</td>
</tr>
<tr>
<td>5</td>
<td>The Effect of Thermistor Jumps (Described in Section 6.5.2.9)</td>
</tr>
<tr>
<td>6</td>
<td>The Effect of a Saturated Thermistor (Described in Section 6.5.2.10)</td>
</tr>
<tr>
<td>7</td>
<td>The Effect of Jumps in a Bolometer Signal (Described in Section 6.5.2.11)</td>
</tr>
<tr>
<td>8</td>
<td>The Effect on Maps due to Cooler Temperature Variations (Described in Section 6.5.2.12)</td>
</tr>
<tr>
<td>9</td>
<td>The Effect on Maps due to Residual Glitches (Described in Section 6.5.2.13)</td>
</tr>
<tr>
<td>10</td>
<td>Holes (NaN's) in the final maps (Described in Section 6.5.2.14)</td>
</tr>
<tr>
<td>11</td>
<td>Edge Effects in the Maps (Described in Section 6.5.2.15)</td>
</tr>
<tr>
<td>12</td>
<td>Large values in SPIRE error maps (Described in Section 6.5.2.16)</td>
</tr>
<tr>
<td>13</td>
<td>Problems with OD1304-1305 Maps (Described in Section 6.5.2.17)</td>
</tr>
<tr>
<td>14</td>
<td>Ringing due to large undetected glitches. (Described in Section 6.5.2.18)</td>
</tr>
</tbody>
</table>
6.5.2.3. Default pixel sizes

For scan map observations, the default pixel sizes for the 250, 350 and 500 micron maps are 6, 10, 14 arcsec respectively. These pixel sizes were adopted using the following criteria, in descending order of priority: 1) the pixel size would not lead to too many NaNs in the final map; 2) the median coverage map above 10 pixels in number and 3) the median error is below a certain factor times the median error achieved at FWHM/2.

The coverage uniformity was assessed after convolving the hit map by an idealized (gaussian) PSF. For each pixel size and array, the median coverage is computed, as well as the minimum and maximum coverage after excluding the 5% lowest values and the 5% largest values, and the minimum coverage after excluding the 0.5% lowest values.

The same treatment was applied to the error map to quantify sensitivity fluctuations within the field of view, but with the target pixels masked out.

6.5.2.4. Missing Meta Data for Scan Lines

For Parallel Mode observations in the archive the meta data parameters numScanLinesNom and numScanLinesOrth may be both set to zero. These parameters appear in the header of the observation context. This is a problem with parallel mode observations only, large and small maps are unaffected. The number of scan lines can be estimated by counting the number of individual scan line building blocks as shown in Figure 6.11.

6.5.2.5. Single Scan Observation

If a very small patch of sky is observed in Large Map mode, the result will be a single scan observation. Although, this may appear somewhat strange, it is actually what is expected for this kind of observing strategy. It is a questionable approach (see Figure 6.43 for the example obsid 0x50002805) because the presence of bad bolometers creates exposed stripes on the map, especially close to the border. In the PLW case, a stripe is also visible to the right of the central scan.

![Figure 6.43. PLW, PMW and PSW maps for a single scan observation.](image-url)
6.5.2.6. The Effect of a Bright Source

Sources brighter than 200 Jy at 250 microns should be observed in bright source mode. Sources below this limit can also appear very bright on SPIRE maps and can create artifacts due to diffraction effects from the telescope support structure with the Airy rings also visible as shown in Figure 6.44 left and right respectively for the example obsid 0x50002A88.

![Figure 6.44. Artifacts in the map due to bright source effects.](image)

6.5.2.7. Stray Light in the Maps

Stray light effects can be seen in the photometer maps if, for example, a planet crosses one of the critical zones of the telescope. In such cases the planet’s light can be refracted by the structure supporting the secondary mirror into the instrument focal plane (e.g. Figure 6.45).

![Figure 6.45. The effect of stray light from a planet on a map.](image)

6.5.2.8. Failure of the Baseline Removal

For all SPIRE map data it is necessary to perform signal baseline removal before the map making stage of the SPIRE photometer pipeline in order to avoid striping in the final images. The standard way of doing this is using the median of the signal in each scan line as the baseline. Although this works in most cases there are examples where this correction fails. This is the most frequent suspect for any tartan looking Level-2 SPIRE products (another cause for striping effects can also be a bad temperature drift correction see Section 6.5.2.9) In Figure 6.46, a strong extended source introduces some bias into the baseline signal and the median removal is unsuitable. More robust estimations for the true baseline signal may be included in later versions of the pipeline.
6.5.2.9. The Effect of Thermistor Jumps

Sudden jumps (from higher to lower voltage) in the signal timelines have been observed in SPIRE data. Currently the cause is unknown but the phenomenon often (but not always) happens on the occurrence of a large positive or negative spike in the timelines. When such a signal jump is found in thermistor timelines it can causes problems with temperature drift correction (see the Thermistor timeline data in Figure 6.47). When using those thermistor data to correct the bolometers baseline, the same jump is then applied to all the bolometers (originally not affected by the jump). This is apparent in the maps as dark or light patches (see Figure 6.47). In future versions of HIPE, the SPIRE pipeline may address this issue but at present users may try using one thermistor (the one not affected by the jump) for the correction (note that this is not possible for PMW because only one thermistor is available) or to manually correct the thermistor signal, removing the jump. The simplest option is to just reject the affected area. Note that there are also jumps in the bolometers themselves (see Section 6.5.2.11), but in this case only single lines are affected.
6.5.2.10. The Effect of a Saturated Thermistor

If a thermistor signal hits the saturation limit, the temperature drift correction process for all subsequent scan lines will fail. This in turns leads to a signal baseline such that the median removal increases the problem. In Figure 6.48, all scan lines after scan number 50 are affected. The problem is identified by plotting the median signal as a function of scan line as seen in the inset in Figure 6.48. Note that this is a very rare effect and in most of the cases, striping on the map is due to jumps in the thermistor (see Section 6.5.2.9) or to variations in the cooler temperature (see Section 6.5.2.12). Whereas the stripes due to thermistor jumps present an almost instantaneous change between dark and bright regions, a saturated thermistor causes a smoother gradual variation since the temperature is varying slower than a sudden jump, but having a stronger intensity than stripes due to uncorrected temperature variations. The SPIRE team is working on improving the robustness of the temperature drift correction module within the pipeline, which should improve the final maps. The current solution at present for the user is to only use the not saturated thermistor (not possible for PMW since there is only a single thermistor).
6.5.2.11. The Effect of Jumps in a Bolometer Signal

A voltage jump in a single bolometer timeline results in a single stripe on the final map, due to a consequent bad baseline subtraction as shown by the white arrows in the left panel of Figure 6.49. The destriper (see Section 6.8.1) contains a method based on wavelet analysis to detect these jumps. In case of a jump detection, the bolometer's timeline affected by the problem is not used to produce the final level2 map (see Figure 6.50). In very few cases, multiple jumps on many detectors may be present causing the more serious map artifact shown in the right panel of Figure 6.49. The cause of such jumps is unknown and still under investigation. At present, possible solutions the user may want to try are to either correct the detector signal manually thus removing the jump or alternatively to reject the affected timeline by masking the bolometer.
6.5.2.12. The Effect on Maps due to Cooler Temperature Variations

The SPIRE cooler must be recycled periodically to maintain a constant low temperature. After the end of the SPIRE cooler recycle, the cooler temperature is 2 mK lower than the nominal plateau (lasting approximately 40h, see middle panel of Figure 6.51) and the cooler requires 8h to reach the stable temperature of the plateau. Observations taken during this period can be affected by a temperature drift too strong to be corrected by the pipeline. As an example, in Figure 6.51, observations taken during one Observation Day are compared. The map in the left panel of Figure 6.51, was processed from an observation immediately after the cooler recycle (i.e. during the strong gradient seen in the middle panel of the figure), while the map in the right panel of the figure was processed some 8 hours after the cooler recycle ended. Large scale striping in the left panel are clearly obvious whilst the map in the right panel appears fine. The difference to effects caused by thermistor jumps is that there is a long but constant drift along the scan direction, present during the entire observation. The sudden discrete jump in the cooler temperature are referred to as Cooler Burps and happens between 6 to 7h after the cooler recycle ends every first SPIRE day. The resolution in this case is to attempt to detect and correct for the variations by checking for large variations in the subK Temperature parameter in the SPIRE Housekeeping data. If this variation is above some threshold then a flag is set in the meta data. If the coolerBurpFound flag is set then the Temperature Drift Correction task will apply additional multiplicative factors to its coefficients to correct for the cooler burp artefacts in the map as shown in Figure 6.52. The correction for cooler burps is optional and is enabled by setting the coolerBurpDetection = True flag at the beginning of the User Pipeline Scripts.

There are some cases of extremely long duration maps made in a single scan direction which may show severe effects of temperature variation across the scans in all bands to varying severity as shown in Figure 6.53. In these cases it is improbable that all artefacts can be removed from the map although experimenting with the Coler-Burp detection on and off may help by removing some of the effects. Such large single scan direction maps may be included as Level 2.5 products if accompanying orthogonal scan observations are available and users are recommended to check these Level 2.5 products if available.
6.5.2.13. The Effect on Maps Due Residual Glitches

Although deglitching is carried out during the standard processing of SPIRE observations (glitch detection, removal and reconstruction). Occasionally, glitches get through and consequently show up on the final maps as bright pixels. Glitches are caused by cosmic rays hitting the detector (affecting single detectors only) or the detector base plate (affecting multiple detectors at once). In the example in Figure 6.54, a cosmic ray hits the PLW base plate generating a glitch on all detectors, which escape
detection by the deglitcher module. The result is a regular pattern of bright pixels on the final PLW map which is simply the footprint of the PLW array on the sky for a given instance of time (the duration of glitch).

An attempt to remove residual glitches can be made by turning on the 2nd Level Deglitching in the SPIRE Destriper Pipeline Task. This procedure is detailed in Section 6.8.1. Alternatively, for data reduced with versions earlier than HIPE 13, Users may want to reprocess their observation using the new SPIRE 2-Pass Pipeline (Section 6.4.1) for better residual glitch removal.

![Figure 6.54. Bright pixels in the final map caused by multiples glitches.](image)

### 6.5.2.14. Holes (NaN's) in the final maps

For maps without cross-linking (orthogonal scans) or no repetitions, holes (represented as NaN's) may be visible in the final maps. NaN's on the signal map, correspond to 0's on the coverage map. This is not an error with the processing but rather an obvious side-effect for maps with low coverage (see Figure 6.55). The most frequent source of NaN's in the map is when the pipeline detects a glitch but it is not able to either remove it, or to reconstruct the detector timeline. In such a case, a mask flag is raised and the affected data is excluded from the final level 2 product. For areas of low coverage the outcome of this, is a NaN on the signal map. This is particularly true for parallel mode (or for single scans), where no repetition or cross-scan is allowed in a single AOR. This is a cosmetic problem and could optionally be overcome by changing the pixel size, or using alternative, external map-making algorithms (e.g. those that use a Bayesian approach).
6.5.2.15. Edge Effects in the Maps

Edge effects are often seen on SPIRE photometer maps (e.g. Figure 6.55). Since these are outside of the guaranteed coverage map area, these can therefore be ignored.

6.5.2.16. Large values in SPIRE error maps

There are cases where the ratio of the error map pixels to signal map pixels are anomalously high and donut shaped artefacts appear at the positions of sources in the error maps (e.g. Figure 6.57). This is due to the fact that error maps produced with the naive mapmaker contain increased errors associated with binning data from Gaussian sources. This is an artifact of the mapmaking process and is also apparent in other map making algorithms. The problem arises since the mapmaker currently calculates the value in a pixel in the error map using the standard deviation of all data falling into a map pixel. If the data lie along the sloped side of a PSF, the dispersion looks high when the data are rebinned. The data at the peak of the PSF look flatter than the sides, and so the error map value at the peak of a PSF looks lower than on the sides of the PSF (see Figure 6.58). This effect has been recreated by injecting an artificial source into a photometer map containing an additional real single point source (the upper artificial source in Figure 6.57). The torus (donut shape) in the error map for both sources is clearly seen confirming the effect is due to the binning rather than any pointing jitter or shot noise. This effect is an artefact of the map making algorithm and cannot be easily overcome, however, the problem with
the error maps can be circumvented by using alternative non-map based methods of source extraction such as the timeline fitter method.

![Signal Map](image1) ![Error Map](image2)

Figure 6.57. Donut shaped artefacts in the error maps at the positions of sources.

![Plot of original signal, line of best fit, mean values of fitted data](image3) ![Standard deviation of fitted data](image4)

Figure 6.58. Anomalous errors from binning data from Gaussian sources.

### 6.5.2.17. Problems with OD1304-1305 Maps

For some reason as yet not understood, a small number of bolometers (PSW B5 F8 E9) are unusually noisy during ODs 1304, 1305. As a result, noisy artefacts can be seen in the final photometer maps (Figure 6.59). This effect was present HIPE versions 11 or earlier but has since been fixed in later versions by masking the offending bolometers. For reprocessing of older maps the following script may be used to mask the bolometers in question:

```python
##### EDIT MASK SCRIPT BEGINS #####
# List of detectors to be masked
bolos = ['PSWB5', 'PSWE9', 'PSWF8']

# Level 1 of your observation, assuming the observation context variable is named 'obs'
levell = obs.levell

# Create new level 1
new_ll = Level1Context()

for scan in range(0, levell.getCount()):
    # Load level 1 product, scan by scan
    data = levell.refs[scan].product
    # Change mask for selected detectors in all scans, setting it to MASTER
    for bolo in bolos:
        data['mask'][bolo].data[:] = 1
    new_ll.addProduct(data)

pass
##### EDIT MASK SCRIPT ENDS #####
```
6.5.2.18. Ringing due to large undetected glitches.

There are some cases of large undetected glitches going undetected by the deglitching algorithms resulting in ringing in the final maps. These often occur in maps with strong residual temperature drift (e.g. obsid= 1342191158). In most cases, the glitches are at the very edge of the turnaround data with a typical hit rate of 3.4 samples per pixel (See Figure 6.60). In such cases, it appears that the only samples in the pixel are the glitch samples. The ringing is expected to be confined in the turn-around data region of the map.

Figure 6.60. Large undetected glitch in the turnaround data.

6.5.2.19. Problems with very early SPIRE Observations.

Although most observations made with SPIRE generally produce very good maps there remain some examples, especially from the early stages of the mission where artefacts remain uncorrected in the final maps. Observations carried out in the very early phases (commissioning phase in the first 70 days of the Herschel mission) or during Performance Verification (PV) phase from operational day (OD) 70-168 may be subject to various caveats such as different bias amplitudes and hence flux calibration or failed temperature drift corrections since the pipeline is not optimized to deal with the early commissioning and verification observations. An example is shown below in Figure 6.61 for an early observation on OD42 of Uranus (1342179049) where the scratch on the map is the result of failed temperature-drift correction due to a cooler burp. Users are advised to use such early observations at their own risk and discretion.

![Figure 6.61](image-url)
6.5.2.20. Flux discrepancies in low coverage in saturated map regions

Careful consideration is required for observations where there exists a sharp gradient between faint and bright regions, especially where external mappers to HIPE are being used. In such scenarios, individual bolometers go outside their electronic limits (saturate) as they cross into a high surface brightness region. However, the bolometers do not all saturate at the same surface brightness level. The naive mapper in HIPE identifies all of the samples falling within an individual map pixel. The signal for the pixel is then calculated using the mean of the sample signals in the pixel. Pixels near bright compact sources cover regions where the surface brightness varies greatly. If some detectors travel up or down the gradient, they will produce usable data only at the faint end of the pixel. In this case, the pixel signal value is weighted towards the faint end of the surface brightness in the region, so the signals are no longer reliable. In most of the central area of the map, the coverage values are around 30, but near sources that saturate, the coverage drops below 20, and a few of the pixels have coverage values of 1-3. This indicates that most of the detectors saturated when inside the regions covered by these map pixels. In summary, for bright sources and strong gradients, saturation does not occur sharply but gradually across the arrays. This can be considered equivalent to looking at the edge of image in terms of coverage and signal to noise.

The use of some external mappers will exacerbate this effect since they do not assign the flux density of a sample to an individual map pixel but instead spread it out over several map pixels using, for example, a drizzle technique. At the edges of bright regions where all detectors are saturated, the signal may be distributed from nearby low surface brightness regions into pixels where no detectors produced usable signal as they crossed through the pixels. This gives the illusion that signal is measured in these regions.

The recommendation to Users is that when using data around saturated regions, they should check their coverage map. If the values in the coverage map are abnormally low compared to the surrounding area, the values in the signal map should not be used.

6.5.2.21. Flux Discrepancies in Fast Parallel Mode Mosaiced Maps

Some examples of the Level 3 products for Fast Parallel mode have been found to have photometry discrepancies where the measured fluxes are around 10 percent less due to a lack of astrometry astrometry correction between the Level 2 and Level 3 creation. All maps in this comparison are in fast...
parallel mode. The more maps you combine the more the beam broadens. This may be fixed in later versions of HIPE by automatic astrometry correction.

6.5.2.22. Uncorrected SSO Maps

Note that moving object SSO maps are automatically corrected by the standard processing pipelines at the HSC (See Section 6.1.1). However, if the SSO observations are re-processed using the User Pipelines then care must be taken to ensure that 2nd Level Deglitching is turned OFF in the Destriper, or else deglitching will be attempted on the smeared frame map and result in flagging of most pixels as glitches. To turn the 2nd Level deglitching off in the Destriper, set the parameter l2DeglitchRepeat=0.

Note that additionally, for the purposes of Photometry, except for fast moving sources (sources visibly smeared on the maps), fluxes measured from the non-motioned corrected map are more consistent than fluxes measured on the motion corrected maps.

6.6. Recipe for Photometer Small Map Mode

6.6.1. Small Map Mode User Pipeline Script

6.6.1.1. Prerequisites

For this data reprocessing example, we will be using the Small Map observation (obsID: 1342195871) of the star Gamma Draconis. We will in this example assume that you have received the engineering pipeline processed Level 0.5 data products from the HSC, and have stored them in a storage pool "OD358-SmallScanMapGammDra0x5000489F", either by a direct download or through HIPE.

The Photometer Small Map User pipeline processing script can be accessed by clicking on 'Pipeline' on the top bar within HIPE, selecting 'SPIRE' and then clicking on 'Photometer Small Map User Pipeline' - the script will open up in the Editor window within HIPE.

6.6.1.2. User Pipeline Inputs

The User Pipeline Script allows us to process our data from Level 0.5 to Level 1. The pipeline script requires some editing before it can be executed. The pipeline requires an Observation ID, Data Pool and an output directory to write plots and FITS files from the pipeline similar to the examples provided in the script below.:

```python
myObsid = 0x5000489F  # 1342195871 in decimal
myDataPool = "OD358-SmallScanMapGammDra0x5000489F"
outDir = "/Users/cpearson/jython/localstore/plots/"
```

Other options are described in the Large Map Pipeline in Section 6.5.1.2

6.6.1.3. Level 0.5 to Level 1 Processing

Currently the entire Small Map Pipeline is identical to that of the Large Map Pipeline, including the values of parameters in the calls to the individual pipeline modules. Therefore the User is referred to the equivalent description in Section 6.5.1.3
6.6.1.4. Level 1 to Level 2 Processing

Currently the entire Small Map Level 1 to Level 2 processing Pipeline is identical to that of the Large Map Pipeline, including the values of parameters in the calls to the individual pipeline modules. Therefore the User is referred to the equivalent description in Section 6.5.1.4.

Three maps are each produced for PSW, PMW and PLW, and are visible through the Product Viewer by right-clicking on the required variable in the Variable pane and selecting 'Open With':

![Product Viewer](image)

Figure 6.62. Selecting the Product Viewer

The Level 2 Products contain 3 images. The actual map with fluxes (denoted as 'image' see Figure 6.63), the statistical flux error map, calculated as the Standard Deviation (denoted as 'error' see Figure 6.64) and an image which shows the coverage map for our scans (denoted as 'coverage' see Figure 6.65):

![Gamma Draconis PSW Level 2 image map](image)

Figure 6.63. The Gamma Draconis PSW Level 2 image map

![Gamma Draconis PSW Level 2 error map](image)

Figure 6.64. The Gamma Draconis PSW Level 2 error map
6.6.1.5. Saving Products from the User Scripts to Local Pools

The SPIRE User Reprocessing Scripts are written to save the final products of the pipeline to FITS files. However, it may be preferable to save the products to local pools instead. The methodology is equivalent to that described for the Large Map processing in Section 6.5.1.6.

6.6.2. Small Map Troubleshooting and Tips

6.6.2.1. Introduction

Since the operation of the Small Map mode is identical to that of Large Map mode, the output image maps are of the same quality and have the same caveats as the maps produced from Large Map mode. Therefore, we refer the user to Section 6.5.2 for a description of possible problems with the Small Map mode.

6.7. Recipe for Photometer Point Source Mode

6.7.1. Point Source Mode User Pipeline Script

6.7.1.1. Prerequisites

For this data reprocessing example, we will be using the Point Source observation (obsID: 1342183474) of NGC 5315. We will in this example assume that you have received the engineering...
pipeline processed Level 0.5 data products from the HSC, and have stored them in a storage pool "1342183474_POF2_NGC5315", either by a direct download or through HIPE. Figure 6.66 outlines the steps required to process the Jiggle pipeline. Note that the actual number of pipeline steps required to process a Point Source mode observation are relatively few, as outlined in the schematic overview of the pipeline script shown in Figure 6.67.

Figure 6.66. The SPIRE POF2 Photometer Point Source pipeline.
6.7.1.2. Level 0.5 to Level 1 Processing

Now, we can process our data from Level 0.5 to Level 1. Looping over each BBID, we first convert the BSM telemetry into a Y, Y and Z angle timeline and then into a chopper id/jiggle id timeline. We can use these to create the SPIRE pointing product. We then perform a number of corrections to the data, after which we will have produced the Level 1 pipeline data product. In order to execute these steps in the most efficient manner possible, we execute a number of pipeline tasks within a single loop. A simplified version of this loop, adapted from the POF2 pipeline script, is given below:
Breaking this loop down into its constituent parts, first we get the basic engineering data products (pdt: Photometer Detector Timeline; bsmt: BSM timeline):

```python
pdt = obs.level0_5.get(bbid).pdt
bsmt = obs.level0_5.get(bbid).bsmt
```

Next, we convert the BSM telemetry in a spacecraft axis Y angle and Z angle timeline - the bat product:

```python
bat=calcBsmAngles(bsmt,bsmPos=bsmPos)
```

and we also converting the BSM telemetry into a chopper id & jiggle id timeline (cjt):

```python
cjt = calcBsmFlags(bsmt, bsmOps=bsmOps)
```

Using the bat product along with the SIAM, HPP and detAngOff products we loaded in earlier, we can construct the Spire Pointing Product for this observation:

```python
spp=createSpirePointing(detAngOff=detAngOff,bat=bat,hpp=hpp, siam=siam)
```

We can now run the wavelet deglitcher on the timeline data. Cosmic ray rejection (degitching) is one of the challenging data analysis problems for SPIRE, as artifacts, caused by undetected glitches, limit the calibration accuracy and sensitivity, and directly influence the quality of the final data products.

```python
pdt=waveletDeglitcher(pdt, scaleMin=1.0, scaleMax=8.0, scaleInterval=5, holderMin=-1.6, holderMax=-0.1, correlationThreshold=0.6, correctGlitches=False)
```

Extracting the fluxConv product from the calibration context, we can perform flux conversion on our PDT product:

```python
fluxConv=obs.calibration.phot.fluxConvList.getProduct(pdt.meta["biasMode"].value,pdt.startDate)
pdt=photFluxConversion(pdt,fluxConv)
```

Add pointing timelines to the data

```python
ppt=associateSkyPosition(pdt,spp=spp)
```

Next, we demodulate the data timelines suing the Chop-Jiggle Marker timeline to separate the different chop and jiggle positions. We calculate the signal for each chop cycle creating a Demodulated Photometer Product (DPP)

```python
dpp = demodulate(ppt, cjt=cjt)
```

We next perform second level deglitching on our data:
Create an average signal at each jiggle position by combining all chop cycles:

\[
dpp = \text{secondDeglitching}(dpp)
\]

Append a list containing pairs of A and B Nod position dpp's. The bbCount contains the BB number

\[
cyc = \left(\frac{\text{dpp.bbCount} - 1}{4}\right) + 1
\]

if \( nyc \geq nrep + 1 \):

\[
\text{for } k \text{ in range}(\text{nyc} - \text{nrep}):
\]

\[
\text{dpparr.append(DenodInput())}
\]

\[
\text{dpparr[ncyc - 1].addProduct(dpp)}
\]

\[
nrep = \text{nyc}
\]

else:

\[
\text{dpparr[ncyc - 1].addProduct(dpp)}
\]

\[
\text{count} = \text{count} + 1
\]

Now take the results at the different Nod positions and de-nod the data, whilst creating a Pointed Photometer Product (ppp) to hold the denoded data

\[
\text{ppps} = []
\]

for \( i \) in range(nrep):

\[
\text{denin} = \text{dpparr}[i]
\]

\[
\text{ppp} = \text{deNodding(denin)}
\]

\[
\text{ppps.append(ppp)}
\]

Finally, we average over all Nod cycles, creating a Level 1 Product: the Averaged Pointed Photometer Product (APPP) \( \text{appp} = \text{nodAverage(ppps)} \)

Looping through the Level 0.5 to Level 1 processing, HIPE produces the following as command-line output:

```
Processing observation 1342183474 (0x50001832) from data pool 1342183474_POF2_NGC5315.
Total number of building blocks for this observation: 8
Starting BBID=0xa3210001: 1/8 Building Blocks
Starting BBID=0xa3210002: 2/8 Building Blocks
Starting BBID=0xa3210003: 3/8 Building Blocks
Starting BBID=0xa3210004: 4/8 Building Blocks
Starting BBID=0xa3210005: 5/8 Building Blocks
Starting BBID=0xa3210006: 6/8 Building Blocks
Starting BBID=0xa3210007: 7/8 Building Blocks
Starting BBID=0xa3210008: 8/8 Building Blocks
Finished the Level 0.5 to Level 1 processing for OBSID= 1342183474, (0x50001832)
Starting the Level 1 to Level 2 processing for OBSID= 1342183474, (0x50001832)
Finished the Level 1 to Level 2 processing for OBSID= 1342183474, (0x50001832)
```

### 6.7.1.3. Level 1 to Level 2 Processing

In the next step of the POF2 Point Source pipeline processing, we can obtain the final Level 2 products for Point Source Observations, by passing the APP to the "pointSourceFlux" module, and by inspecting the output JPP product. This is the final product containing the best fit flux and # position using all the 7-pt Jiggle information:

```
# user products
jpsfp = pointSourceFit (appp)
jpp = sourceFlux (jpsfp)
simpleFitsWriter(jpp, "%sJPP_%i.fits"%(outDir, myObsid))
```
Finally, we can create sparse maps for each detector array (see Fig 5.55 for the PSW sparse map), whilst writing the FITS output to our defined output directory:

```python
print "Creating sparse maps for OBSID= %i, (0x%x)"%(myObsid,myObsid)
mapPsw=naiveApppMapper(appp, array="PSW")
mapPmw=naiveApppMapper(appp, array="PMW")
mapPlw=naiveApppMapper(appp, array="PLW")
simpleFitsWriter(mapPsw, "%smapPSW_%i.fits"%(outDir, myObsid))
simpleFitsWriter(mapPmw, "%smapPMW_%i.fits"%(outDir, myObsid))
simpleFitsWriter(mapPlw, "%smapPLW_%i.fits"%(outDir, myObsid))
print "Finished sparse maps for OBSID= %i, (0x%x)"%(myObsid,myObsid)
```

Figure 6.68. PSW Sparse Map

with the following output on the command line from HIPE.

```
# Creating sparse maps for OBSID= 1342183474, (0x50001832)
Finished sparse maps for OBSID= 1342183474, (0x50001832)

# Creating sparse maps for OBSID= 1342183474, (0x50001832) Finished sparse maps for OBSID= 1342183474, (0x50001832)

Congratulations! You have now successfully reprocessed your point source data from Level 0.5 to the final Level 2 user products! Additional and more detailed information regarding the data processing modules and the data at the various levels of processing can be found in the SPIRE Users Manual.

**Saving Products from the User Scripts to Local Pools**

The SPIRE User Reprocessing Scripts are written to save the final products of the pipeline to FITS files. However, it may be preferable to save the products to local pools instead. The methodology is equivalent to that described for the Large Map processing in Section 6.5.1.6.
6.7.2. Point Source Mode Troubleshooting and Tips

The usage of the SPIRE Point Source mode is recommended only for specialized observations of bright isolated sources in the range 0.2-4Jy where the astrometry is accurately known and accurate flux measurement is required. For sources fainter than 200mJy (where the background produces a significant contribution) or at fluxes higher than 4Jy (where pointing jitter can introduce large errors) the Small Scan Map mode is preferable. Taking these caveats into account, if when looking at the final Level 2 Products an unexpected result is apparent, the following points may help. Note obvious initial checks would be to check that the Level 2 processing was reached for all bands (PSW/PLW/PMW), that the astrometry of maps matches that in the meta data / original HSpot input, etc. The quality flags can also be examined.

6.7.2.1. No maps output from the Point Source Mode

Note that nominally, the output from the Point Source mode pipeline is a Jiggled Photometer Product (JPP) containing the numbers for the fitted positions and flux plus errors for the source in the PSW, PMW, PLW. No map is output in this mode although it is possible to obtain a sparse unfilled map with the following lines of Jython included after the Level 1 product production in the pipeline script, producing the maps shown in Figure 6.69. Note that the target and the two negative chop fluxes can be seen in the map.

```python
mapPsw=naiveApppMapper(appp, array="PSW")
mapPmw=naiveApppMapper(appp, array="PMW")
mapPlw=naiveApppMapper(appp, array="PLW")
```

![Figure 6.69. The sparse maps for the Point Source mode.](image)

6.7.2.2. Different Flux Measurements depending on Observation Date

Observations of sources taken at different times can produce varying signal strengths for the source in the final product, since for Point Source mode, the effective sky confusion level is increased due to chopping and nodding (see left panel of Figure 6.70). The result of the measurement is therefore affected by the specific characteristics of the sky background in the vicinity of the source and will depend on the chop/nod position angle in the event of an asymmetric background. The example in the right panel of Figure 6.70 shows a scan map observation of a ~220mJy source. The circle drawn around the source corresponds to the chop and/or nod throw used in the Point Source mode (126
arcs). Moving around the circumference of the circle it is found the background can vary by 30mJy around the mean depending on the chop/nod position angle used for the observation. Note that as set out in the SPIRE Handbook (formerly the SPIRE Observers Manual), the point source AOT is not recommended for sources fainter than ~200mJy, for which a small scan map will produce a better measurement including an accurate characterisation of the background.

Figure 6.70. The contribution of the background to chopped observations.

6.7.2.3. Large Errors on Bright Source Fluxes

Observations of bright sources may have uncertainties dominated by pointing jitter and nod-position differences, resulting in a S/N on the order of 100 at most (the uncertainties in the data will also be limited by the accuracy of the flux calibration, which will be at least 5%). This can be seen in Figure 6.71 where on the left an observation of a 5.1 Jy source is shown as the fitted flux on a prime jiggle detector as a function of the number of individual Nod cycles. The signal can be seen to vary by as much as 0.2Jy. In the right panel of Figure 6.71 this effect is shown pictorially as a Gaussian with three points on curve assuming arbitrarily a slight offset from 5.6° away from the maximum. Assuming small Nod uncertainties 0.5° away from either side of the central point we see a difference between flux of central point and the other two points is 0.2 Jy. Note that in the observers manual this mode is not recommended for sources brighter than 4Jy.

Figure 6.71. The effect of pointing jitter and nod uncertainties on the measured signal.

6.8. Destriping and Baseline Removal

6.8.1. Improving Maps with Baseline Removal and Destriping Tools

Due to the large telescope background, fluxes measured by the SPIRE bolometers are effectively very small differences on top of a dominating offset that is usually several orders of magnitude larger. Due to variations in the thermal and electronic stability of the system, residual offsets in the flux calibration from one detector to another are observed, resulting in striping in the reconstructed maps.
The current version of the SPIRE pipeline uses an iterative destriper to remove stripes in the maps and is recommended to produce the highest quality maps available from the pipeline. In earlier versions of the standard SPIRE pipelines the algorithm used to remove such striping was a simple Median Baseline Removal which determined the median from each detector in the Level 1 timelines and then creates a map. The Median Baseline Removal algorithm is significantly faster than the destriper and in many cases, works as well as most of the readouts are seeing an overall flat background. In structured regions with cirrus clouds and gradients the Median Baseline Removal method will produce less perfect results.

For this reason, an example script (Photometer_BaselineRemovalDestriper.py, accessible from the Scripts menu in HIPE as shown in Figure 6.72) has been provided within HIPE to allow the User to experiment with different forms of stripe removal from maps. The available algorithms are briefly outlined below and explained in detail in the following sections. Finally, a rough guide to recommendations is given below in Section 6.8.1.4

1. **SPIRE Destriper:** The current pipeline default. A generalized algorithm that is independent of the map contents. It iteratively updates offsets until an optimal solution is found (see Section 6.8.1.1).

2. **Median Baseline Subtraction:** This algorithm subtracts the median value from each scan or alternatively from the entire timeline (see Section 6.8.1.2). In addition a Region of Interest can be defined to mask out a region of the map.

3. **Polynomial Baseline Subtraction:** An alternative algorithm that fits a polynomial of given order (set by the User) to each scan or alternatively to the entire timeline (see Section 6.8.1.3). In addition a Region of Interest can be defined to mask out a region of the map.

![Figure 6.72. Selecting the Photometer Baseline Removal and Destriper Script](https://nhscsci.ipac.caltech.edu/sc/index.php/Spire/PhotScanMapDestriper)

### 6.8.1.1. The SPIRE Destriper

**Overview**

The SPIRE destriper was developed to provide a generalized solution independent of the map contents and is the current default method for baseline/stripe removal in the SPIRE pipelines. The destriper iteratively updates offsets in the timelines until an optimal solution is found. The disadvantage is that it takes substantially longer to run than a simple median subtraction, so for large scale point source surveys it will not do better but take much longer. For structured regions however, like the galactic plane, such a tool is indispensable to create stripe free maps. There are also provisions for residual signal drifts and additional deglitching. A comparison between the the current pipeline default Median Baseline Removal and the Destriper on a map with extended emission is shown in Figure 6.73.

The Destriper is described in great detail on a dedicated [Destriper Web Page](https://nhscsci.ipac.caltech.edu/sc/index.php/Spire/PhotScanMapDestriper) found at https://nhscsci.ipac.caltech.edu/sc/index.php/Spire/PhotScanMapDestriper.
Destriper Call and Parameters

The basic call to the destriper is shown in Example (4) in the Photometer Baseline Removal and Destriper Script and is described in below in Table 6.6 and Table 6.7. Note that in addition to the detripped output, the Destriper als provides a Diagnostic Table containing useful information about the fit, described in Table 6.8 and Figure 6.75 below. Note that the diagnostic table can be used to feed information back into the destriper for a new set of iterations (See startParamters option in Table 6.7) Note that for very large maps the detriper requires more than 7Gb of memory which may cause memory problems on smaller machines. Memory problems can be overcome by setting the useSink=True parameter in the Destriper call below (Default is useSink=False)

```python
# Get the Level 1 scan lines from the Observation Context
level1 = obs.level1
# run destriper + mapmaking
level1Corrected, mapPsw_destriper, diagnosticProduct, PSWtod, signalMinusMapSignal = \
    destriper(level1=level1, array='PSW', \
              pixelSize=6.0, offsetFunction='perScan',
              polyDegree=0, kappa=20.0, iterThresh=1.0E-10,
              l2DeglitchRepeat=100, iterMax=100, l2IterMax=5, \n
              nThreads=2, jumpThresh=8.0, jumpiter=10, \
              withMedianCorrected=True, brightSource=True, useSink=False, 
              storeTod=False)
```

The Destriper can also be accessed via a GUI selected from the Tasks window within HIPE. The GUI is shown in Figure 6.74 and the parameters listed in Table 6.7.

![Destriper GUI interface](image)

Figure 6.74. Destriper GUI interface
Table 6.6. Description of Destriper Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>level1Corrected</td>
<td>List context containing the processed scans with optimized offset functions subtracted from the timelines of the selected detector array.</td>
</tr>
<tr>
<td>mapPsw_destriper</td>
<td>Simple reconstructed map for selected detector.</td>
</tr>
<tr>
<td>DiagnosticProduct</td>
<td>Diagnostic product with detailed information about the fitting results.</td>
</tr>
<tr>
<td>PSWtod</td>
<td>Time ordered data product that can be used by a subsequent map maker (if the skybin size unchanged).</td>
</tr>
<tr>
<td>signalMinusMapSignal</td>
<td>List context with signal minus the re-sampled map signal after the last iteration which provides a check of the data the offset function is being fitted against.</td>
</tr>
</tbody>
</table>

Table 6.7. Description of Destriper Call Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>level1</td>
<td>A list context or a Level 1 context containing all scans of one or more observations.</td>
</tr>
<tr>
<td>tod</td>
<td>Provide a Time Ordered Data (TOD) set or TOD buffer instead of Level 1 product.</td>
</tr>
<tr>
<td>array</td>
<td>The array that is to be processed, either “PSW”, “PMW”, “PLW”.</td>
</tr>
<tr>
<td>pixelSize</td>
<td>Size of the skybins in arcsec (default is set to 6, 10, 14 for PSW, PMW, PLW respectively).</td>
</tr>
<tr>
<td>polyDegree</td>
<td>Degree of polynomial that forms the offset functions. Default is 0.</td>
</tr>
<tr>
<td>offsetFunction</td>
<td>Selects between determining separate offset functions for each detector and building block (“perScan”) or one offset function per detector for the entire observation “fullTimeline”. Note that if the input contains scans of more than one observation, separate offset functions are calculated for each observation.</td>
</tr>
<tr>
<td>kappa</td>
<td>Upper threshold for number of standard deviations of the residual between signal, offset function and re-sampled map timeline, above which a Level 2 glitch is detected. Recommended value is 7.</td>
</tr>
<tr>
<td>kappa2</td>
<td>Lower threshold for number of standard deviations of the residual between signal, offset function and re-sampled map timeline, above which a Level 2 glitch is detected. Recommended value is also 7.</td>
</tr>
<tr>
<td>iterThresh</td>
<td>Threshold for Chi squared to stop iterations.</td>
</tr>
<tr>
<td>l2DeglitchRepeat</td>
<td>Number of iterations after which the Level 2 deglitcher is executed. This is a repetitive process. If set to zero the Level 2 deglitcher is never executed. If set to a larger value than the number of iterations necessary for convergence, the Level 2 deglitcher is executed once at the end.</td>
</tr>
<tr>
<td>iterMax</td>
<td>Maximum number of iterations that are allowed.</td>
</tr>
<tr>
<td>l2IterMax</td>
<td>Maximum number of iterations for 2nd level deglitches.</td>
</tr>
<tr>
<td>nThreads</td>
<td>Number of threads for multi-threading processing.</td>
</tr>
<tr>
<td>jumpThresh</td>
<td>Threshold to detect the bolometer jump.</td>
</tr>
<tr>
<td>jumpIter</td>
<td>Number of the iteration which the jump detection is performed.</td>
</tr>
<tr>
<td>withMedianCorrected</td>
<td>Include an initial median subtraction (recommended)</td>
</tr>
<tr>
<td>brightSource</td>
<td>Switch to decide whether bright sources should be excluded. This does not affect blank fields and can be left set to True.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>brightSourceThresh</td>
<td>Threshold for bright sources used when brightSource flag is set. Default is 1.5, for really bright sources the threshold can be lowered to 1.0.</td>
</tr>
<tr>
<td>startParameters</td>
<td>Optional table dataset (e.g. Diagnostic table) that can be fed back into another Destriper run.</td>
</tr>
<tr>
<td>useSink</td>
<td>Boolean switch to use a Temporary Storage to save processing memory. The default is off (False)</td>
</tr>
<tr>
<td>storeTod</td>
<td>Boolean switch to store TOD in a temporary directory. The default is off (False)</td>
</tr>
<tr>
<td>roi</td>
<td>Optional mask to exclude an area from the destriper (see Section)</td>
</tr>
<tr>
<td>useOnlyMasks</td>
<td>Overides default mask set. Masks can be selected from a drop down list in the GUI window. Masked data will be excluded from the processing.</td>
</tr>
<tr>
<td>ignoreDefaultMasks</td>
<td>Masks to explicitly include in the processing. Masks can be selected from a drop down list in the GUI window. Caution: this overides the default set of masks.</td>
</tr>
<tr>
<td>withMedianCorrected</td>
<td>Set to True is median is to be subtracted first. Default = True</td>
</tr>
<tr>
<td>minVel</td>
<td>(used by Naive Mapper) Minimum velocity limit of turnaround data to be included in the map making in arcsec/s. Default = 5 arcsec/s</td>
</tr>
<tr>
<td>maxVel</td>
<td>(used by Naive Mapper) Maximum velocity limit of turnaround data to be included in the map making in arcsec/s. Default is no limit</td>
</tr>
<tr>
<td>method</td>
<td>(used by Naive Mapper) Method to calculate the error map. Default is method=UnweightedVariance corresponding to the standard deviation with no weighting. Alternative is method=WeightedVariance which produces a weighted error map using individual bolometer noise characteristics (requires the chanNoiseList calibration product). Alternatively a map of medians can be produced using the method=MedianMap option</td>
</tr>
<tr>
<td>chanNoise</td>
<td>(used by Naive Mapper, see WeightedVariance method for mapper above) Channel noise calibration product container detector noise power spectrum.</td>
</tr>
</tbody>
</table>

**Table Dataset**

<table>
<thead>
<tr>
<th>Table Dataset</th>
<th>Mean</th>
<th>Median</th>
<th>Min</th>
<th>Max</th>
<th>StdDev</th>
<th>obsError/1000</th>
<th>obsError/10000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1.0</td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>2.0</td>
<td>20.0</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>3.0</td>
<td>30.0</td>
</tr>
</tbody>
</table>

**Figure 6.75. Destriper Diagnostic Product**
### Table 6.8. Description of Destriper Diagnostic Table

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>Index number of detector channel. This can be the standard order of channels as they appear in the data. This will appear more than once if “perScan” was selected and more than one scan was commanded.</td>
</tr>
<tr>
<td>channelName</td>
<td>Strings like “PSWE6#” or “PLWD2#”. Name of detector channel. This will appear more than once if “perScan” was selected and more than one scan was commanded.</td>
</tr>
<tr>
<td>detIndex</td>
<td>Detector bolometer number on the PSW, PMW, PLW arrays.</td>
</tr>
<tr>
<td>scanNumber</td>
<td>Scan number or sequential building block number in the AOR starting with 0. There will only be more than one scan number if “perScan” was selected and more than one scan was commanded.</td>
</tr>
<tr>
<td>Iter</td>
<td>Number of iterations for this channel until converged. If the channel didn’t converge set to parameter iterMax.</td>
</tr>
<tr>
<td>chiSquare</td>
<td>Last chi square achieved for this channel.</td>
</tr>
<tr>
<td>convergence</td>
<td>The convergence status achieved for this channel.</td>
</tr>
<tr>
<td>numberL2Flags</td>
<td>Number of Level 2 flags set in the timeline for this channel.</td>
</tr>
<tr>
<td>a0</td>
<td>First parameter of polynomial fit function</td>
</tr>
<tr>
<td>a1</td>
<td>Second parameter of polynomial fit function</td>
</tr>
<tr>
<td>a2</td>
<td>Third parameter of polynomial fit function</td>
</tr>
<tr>
<td>a3</td>
<td>Fourth parameter of polynomial fit function</td>
</tr>
<tr>
<td>deselected</td>
<td>Flag for Do not include this channel in Destriper run</td>
</tr>
</tbody>
</table>

### Improving Destriped Maps

The basic destriper call described above can produce greatly improved results over the standard baseline removal techniques, especially for diffuse maps. Further improvements can be made by considering the following:

#### Treatment of Bright Sources:

Setting the bright source flag (brightSource=True) in the case of a bright source in the centre of the map as shown in Figure 6.76

#### Map Turnarounds:

Including turnaround data in the final maps by re-running the User Pipelines with the parameter includeTurnaround=True set (See Figure 6.77)

#### Removal of Residual Glitches (2nd Level Deglitching):

Even after pipeline processing, some residual glitches may remain in the final maps. The destriper supports outlier rejection (2nd Level Deglitching) as an option. To turn the 2nd Level Deglitching on in the Destriper, the parameters l2DeglitchRepeat, l2IterMax, kappa and kappa2 should be set (See Table 6.7). Recommend values are l2DeglitchRepeat = 100, l2IterMax = 1 (or 2, 3), kappa = 7, kappa2 = 7. Note that lower kappa values of, kappa = 4, kappa2 = 4 are possible but may unexpectedly be flagging the top of your sources at a level of 1-3 percent, resulting in lower flux densities. The recommended kappa values are 7, verified to not affect point-source fluxes. The example Destriper call is shown below:

```python
# run destriper + mapmaking
level1Corrected, mapPsw_destriper, diagnosticProduct, \
PSW tod, signalMinusMapSignal = \
destriper(level1=level1, array='PSW', \
pixelSize=6.0, offsetFunction='perScan',\
```
Masking out a Region of Interest (ROI): 

In some cases it may be desirable to mask a region in a map to exclude it from the destriping process. For this reason, a Region Of Interest (ROI) can be defined to effectively mask out an area. Once defined, the roi parameter can be included in the destriper call as a final parameter as shown below:

```python
# run destriper + mapmaking
level1Corrected, mapPsw_destriper, diagnosticProduct,
PSWtod,signalMinusMapSignal = 
destriper(level1=level1, array='PSW', 
pixelSize=6.0, offsetFunction='perScan',
polyDegree=0, kappa=20.0, iterThresh=1.0E-10,
12DeglitchRepeat=100, iterMax=100, 12IterMax=5, 
nThreads=2, jumpThresh=8.0, jumpIter=10, 
withMedianCorrected=True, brightSource=True, 
useSink=False, storeTod=False, roi=roi)
```

The ROI is a versatile parameter (using a SkyMask as input). The sky mask can be used to define a number of different masks. For example:

```python
# Define a simple region of interest as everything NOT in
# a circle of radius 3 arcmin around (ra, dec)
roi = SkyMaskCircle(ra, dec, 3).not()
#
# Define region of interest to avoid two bright sources:
# (ra1, dec1) and (ra2, dec2)
circle1 = SkyMaskCircle(ra1, dec1, 3) # region INSIDE circle1
circle2 = SkyMaskCircle(ra2, dec2, 3) # region INSIDE circle2
roi = circle1.or(circle2).not() # region NOT in (circle1 OR circle2)
#
# Define region of interest NOT in rectangle
roi = SkyMaskRectangle(raMin, raMax, decMin, decMax).not()
```

The SkyMask can be combined with other logical operators (as well as being inverted with .not()) such as .and(), .or() and .xor().

Destriping using the full timeline data:

In some cases particularly observations affected adversely by cooler burps the destriper may still not provide a clean final map and residual striping may remain. In this case, a possible remedy may be to run the Destriper in Full Timeline mode. In this mode, the destriper attempts to fit the entire timeline data from the observation in one go (as opposed to the default of fitting per scan). In full timeline mode, the best results are obtained if turnaround data is included in the processing. To run the Destriper in full timeline mode, the offsetFunction option should be set to 'fullTimeline' and the degree of the fitting polynomial should be selected, e.g. polyDegree = 1. The degree of polynomial depends on the length of observation, where usually a longer observation requires a higher polynomial. An example of a Destriper call in full timeline mode is given below (see also the Example (5) in the Photometer Baseline Removal and Destriper Script).

```python
# get the level 1 data from an observation
level1 = obs.level1
#
# Run destriper in full timeline mode
level1Corrected, mapPsw_destriper_full, diagnosticProduct,
PSWtod,signalMinusMapSignal = 
destriper(level1=level1, array='PSW', 
```
Optimum Destriped Maps:

The best Destriper maps are those correctly calibrated for Extended Emission (\texttt{extdPxW} in the Level 2 Products). These maps include the application of the relative gain factors for extended emission (setting the \texttt{applyExtendedEmissionGains=True}) (See Example (6) in the Photometer Baseline Removal and Destriper Script and Figure 6.77).

The best stripe free maps for SPIRE are therefore produced by the destriper including the extended gain factors and the map turnarounds as shown in the progressive maps in Figure 6.77.

6.8.1.2. Median Baseline Subtraction

The Median Baseline Subtraction task is provided as an option in the standard pipeline and in the majority of cases provides good science quality image maps. In its default configuration, the task will subtract a constant from each detector equal to the median value in each detector’s signal timeline for each individual scan line. The default call is given by Example (1) in the Photometer Baseline Removal and Destriper Script, where the input is a Level 1 Context and the output is a list that can be directly fed into the mapmaker;

```
level1 = obs.level1
scansBaseline=baselineRemovalMedian(input=level1)
```
Additionally, the Median Baseline Subtraction task may also be run on the entire observation (subtracting a fixed value for all scan lines of a detector) which may improve some observations. In this case the call to the task would be:

```python
lev1 = obs.lev1
scansBaseline=baselineRemovalMedian(input=lev1, wholeTimeline=True)
```

Observations with a bright source at the centre of the map may cause the Median Baseline Subtraction task to produce dark patches in the map (See Figure 6.78). For this reason, a Region Of Interest (ROI) can be defined to effectively mask out an area from the baseline removal procedure. Figure 6.78 shows the differences in the final map using the Median Baseline Removal task in different modes, including using a ROI. The simplest way to incorporate a ROI is to define a circle of some radius in arc minutes around the centre of the map as outlined below;

```python
# Get the Level 1 scan lines from the Observation Context
lev1 = obs.lev1
# Defining Region Of Interest (ROI)
# Define a circle of radius 4 arc minutes around centre of map as our mask
wcs = obs.lev2.getProduct("PSW").wcs
roi = SkyMaskCircle(wcs.crval1, wcs.crval2, 4).not()
Display(roi.masks(obs.lev2.getProduct("PSW")), title="(1a) ROI Mask")
#******** (Example 1a):  Scan-by-scan, ROI ****************
scans_ROI1 = baselineRemovalMedian(input=lev1, roi=roi)
```

Figure 6.78. Median Baseline Removal Task

The ROI is a versatile parameer (using a SkyMask as input). The sky mask can be used to define a number of different masks. For example:

```python
# Define region of interest as everything NOT in a circle of radius 3 arcmin around (ra, dec)
roi = SkyMaskCircle(ra, dec, 3).not()
scans = baselineRemovalMedian(input=obs.lev1, roi=roi)

# Define region of interest to avoid two bright sources: (ra1, decl1) and (ra2, dec2)
circle1 = SkyMaskCircle(ra1, decl1, 3) # region INSIDE circle1
circle2 = SkyMaskCircle(ra2, decl2, 3) # region INSIDE circle2
roi = circle1.or(circle2).not() # region NOT in (circle1 OR circle2)
scans = baselineRemovalMedian(input=obs.lev1, roi=roi)

# Define region of interest NOT in rectangle
roi = SkyMaskRectangle(raMin, raMax, decMin, decMax).not()
scans = baselineRemovalMedian(input=obs.lev1, roi=roi)
```

The SkyMask can be combined with other logical operators (as well as being inverted with .not()) such as .and(), .or() and .xor().
6.8.1.3. Polynomial Baseline Subtraction

The Polynomial Baseline Subtraction task provides an alternative baseline subtraction algorithm when the standard Median Baseline Subtraction method has problems (See the examples in Figure 6.79). The Polynomial Baseline Subtraction task fits a polynomial of degree specified by the User to each detector’s signal timeline for each individual scan line. The default call is given by Example (2) in the Photometer Baseline Removal and Destriper Script, where the input is a Level 1 Context and the output is a list that can be directly fed into the mapmaker:

```python
levell = obs.levell
scansBaseline = baselineRemovalPolynomial(input=levell, polyDegree=1)
```

Additionally, the Polynomial Baseline Subtraction task may also be run on the entire observation (fitting a polynomial to the entire observation for each detector), shown in Example (3) in the Photometer Baseline Removal and Destriper Script. The fit to the entire timeline is useful in removing any "low frequency" drifts in a long observation. However, the task will fail if the data contains "discontinuities" e.g. jumps. In this case the call to the task would be:

```python
levell = obs.levell
scansBaselineMerged = baselineRemovalPolynomial(input=levell, polyDegree=1, wholeTimeline=True)
```

6.8.1.4. Recommendations and Suggestions

In Figure 6.79 examples are shown for the results obtained by a selection of baseline removal configurations and the SPIRE Destriper for a Large Map with a bright source, Small Map, Large Map with extended emission and a single scan. The figure shows that careful consideration and some experimentation must be made in order to get the best looking maps. Note that all configurations of the baseline removal and destriping algorithms are shown in the Figure and the User is advised to read the above sections and experiment with settings to get the best results.

In Figure 6.80 the recommendations for baseline/Destriper algorithm usage is summarized as a function of observation mode, filed size, source type, etc. Again, these are intended as rudimentary guidelines only and the User is advised to read the above sections and experiment with settings to get the best results. In many cases the Destriper gives the best results, especially for extended emission, where the optimum results are usually obtained by using the corrections for extended gains in combination with the scan line turnaround data. In this case, the code for the Destriper should by cut-and-pasted into the appropriate User Pipeline Script and the following options should be set to TRUE in the User Script before it is run:

```python
includeTurnaround = True
applyExtendedEmissionGains = True
```

For bright sources, then the Region Of Interest (ROI) should be set for the Median or Polynomial tasks. For the Detriper, the Bright Source Flag needs to be set.
Figure 6.79. Examples of Baseline Removal and Destriping results
6.9. Recipe for SPIRE Photometry

6.9.1. SPIRE Source Extraction & Photometry in HIPE

HIPE has many methods available for source extraction and photometry. In this section, we will show how to perform both source extraction and photometry on the Level 2 product maps for a variety of source types (see Section 6.9.1.3 for an overview of the point source photometry methods available in HIPE). In Section 6.9.1.4 the source extraction and photometry using the HIPE algorithms SUSSEXtractor and DAOphot is explained. In Section 6.9.1.5 photometry via timeline fitting of the Level 1 products is explained. In Section 6.9.1.6 aperture photometry on SPIRE images is outlined.

The current recommendation for point source photometry is to use the Timeline Fitting algorithm for photometry of all but the faintest SPIRE maps, however all photometry methods return broadly similar results for SPIRE data. Source coordinates can either be fed directly to the Timeline Fitter or alternatively can be provided via the output of the source extraction from SUSSEXtractor or DAOphot. These recommendations are shown pictorially in Section 6.9.1.6.
6.9.1.1. SPIRE Flux Calibration

Full details of the SPIRE calibration can be found in the SPIRE Observers Manual and in dedicated publications: the calibration scheme is described in Griffin et al. (2013) and the implementation using Neptune as the primary calibration standard, is describe in Bendo et al. (2013). The treatment of the calibration for extended emission can be found in North et al. (2013).

The SPIRE photometer flux calibration has two sources of uncertainties which should be included in addition to the statistical errors of any measurement for point source calibration. One is a systematic uncertainty in the flux calibration related to the uncertainty in the models used for Neptune, the primary calibrator; these uncertainties, which are correlated across all three SPIRE bands, are currently quoted as 4%. The other source of uncertainty is a random uncertainty related to the ability to repeat flux density measurements of Neptune. This random uncertainty is <1.5% for all three bands.

For extended emission calibration, in addition to the above uncertainties, there is an additional 1% uncertainty due to the current uncertainty in the measured beam area.

6.9.1.2. Converting Jy/beam to surface brightness or flux densities in the SPIRE pipeline

Pipeline data and Level-2 maps are calibrated in Jy/beam. It is important to note that, since the SPIRE photometer flux calibration is performed on the timeline data, the beam areas equivalent to the beams of the timeline data must be used when calibrating extended emission in terms of surface brightness (Jy/pixel or /sr). To convert maps from from Jy/beam to Jy/pixel, the point source calibrated maps need to be divided by the beam. The beam areas corresponding to the 1 arcsec pixel scale for a spectral index ($\alpha$) of -1 as used in the pipeline should be used (see Table 6.10). However, point source fluxes measured on surface brightness maps (e.g. Jy/pixel) need to be corrected by a multiplicative factor corresponding to the ratio of the pipeline beam ($\alpha = -1$) and the effective beam for the assumed spectral index of the source, which take into account the RSRF, the aperture efficiency and the variation of beam profile with frequency. These ratios are given in Table 6.11.

The standard SPIRE pipeline flux calibration assumes a point source calibration for the standard spectral index of $\alpha = -1$ adopted for Herschel. Conversion factors to transform RSRF-weighted fluxes densities to monochromatic flux densities for point and extended sources are applied automatically in the the pipeline. All the flux conversion parameters are explained in detail in the (SPIRE Handbook). The parameters are listed in Table 6.9. K4P is the pipeline point source flux conversion parameter. KMonE($\alpha = -1$) is the conversion to monochromatic surface brightness for an extended source with...
\( \alpha = -1 \). KPtoE is the conversion from point source flux density to extended source surface brightness for a source spectrum \( \alpha = -1 \). \( \Omega_{\text{pip}} \) is the beam solid angle for a source with \( \alpha = -1 \), i.e. The effective beam area (\( \Omega_{\text{eff}} \)) for \( \alpha = -1 \). K4E is the flux conversion parameter used in the pipeline, defined as KMonE(\( \alpha = -1 \)\( \Omega_{\text{pip}} \), which converts to the flux density of an extended source.

The conversion between extended and point source calibration is given by the ratio \( K_{4E}/K_{4P} \). This ratio is referred to as \( K_{4E/4P} \) and converts a point source monochromatic flux density to a monochromatic extended source surface brightness (see Table 6.9, not to be confused with KPtoE which includes the beam in the parameter and cannot be derived directly from the SPIRE Calibration Tree - see the code examples below).

All these factors are automatically applied to the standard pipeline products point source (psrcPsW in Jy/beam) and extended emission (extdPsW in MJy/sr) products. The extended emission (extdPsW, see Table 6.1) products have also been processed with the relative gains applied and have been absolute zero-point corrected using the Planck maps.

The K4E and K4P are the only K parameters explicitly included within the SPIRE Calibration Tree. For the purposes of Point source photometry, dividing by the K4E/K4P (K4EdivK4P) parameter converts from extended to point source calibration.

The relationship between the various K parameters is given by:

\[
KPtoE = \frac{K_{\text{MonE}}}{K_{4P}} = \frac{(K_{4E}/\Omega_{\text{eff}})}{K_{4P}} = \frac{(K_{4E/4P})}{\Omega_{\text{eff}}}
\]

Note that the pipeline assumes a value of \( \alpha = -1 \), therefore for other spectral indicies, the appropriate colour corrections should also be applied (see Section 6.9.1.8).

For the purposes of point source extraction, where the absolute scale of the beam model is unimportant, the appropriate FWHM corresponding to the 1 arcsec pixel scale should be used from Table 6.12 below.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>PSW</th>
<th>PMW</th>
<th>PLW</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wavelength (microns)</td>
<td>250</td>
<td>350</td>
<td>500</td>
</tr>
<tr>
<td>K4P</td>
<td>1.0102</td>
<td>1.0095</td>
<td>1.0056</td>
</tr>
<tr>
<td>KMonE (MJy/sr per Jy/beam)</td>
<td>91.5670</td>
<td>51.6654</td>
<td>23.7113</td>
</tr>
<tr>
<td>KPtoE (MJy/sr per Jy/beam)</td>
<td>90.6462</td>
<td>51.1806</td>
<td>23.5798</td>
</tr>
<tr>
<td>Beam Area (arcsec(^2))</td>
<td>469.3542</td>
<td>831.275</td>
<td>1804.3058</td>
</tr>
<tr>
<td>K4E</td>
<td>1.0102</td>
<td>1.0095</td>
<td>1.0056</td>
</tr>
<tr>
<td>K4E/K4P (K4EdivK4P)</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 6.10. Beam Areas assumed by the pipeline (\( \alpha = -1 \))

<table>
<thead>
<tr>
<th>Spectral Index</th>
<th>Beam Area (arcsec(^2))</th>
<th>Beam Area (/10(^8)sr)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( F_{\nu}=\text{nu}^{-4} )</td>
<td>PSW</td>
<td>PMW</td>
</tr>
<tr>
<td>-1.0</td>
<td>469.3542</td>
<td>831.27497</td>
</tr>
</tbody>
</table>

Table 6.11. Effective Beam Area ratios (beam correction) as function of spectral index (\( \alpha \))

<table>
<thead>
<tr>
<th>Spectral Index</th>
<th>Effective Beam Area Ratio (( \Omega_{-1}/\Omega(\alpha) ))</th>
</tr>
</thead>
<tbody>
<tr>
<td>( F_{\nu}=\text{nu}^{-4} )</td>
<td>PSW</td>
</tr>
<tr>
<td>-4</td>
<td>0.9593</td>
</tr>
</tbody>
</table>
Table 6.12. SPIRE FWHM Parameters for 1 arcsec pixels.

<table>
<thead>
<tr>
<th>Band (micron)</th>
<th>FWHM (arcsec)</th>
<th>Mean FWHM (arcsec)</th>
<th>Ellipticity (Flattening) (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>250</td>
<td>18.4x17.4</td>
<td>17.9</td>
<td>5.1</td>
</tr>
<tr>
<td>350</td>
<td>24.9x23.6</td>
<td>24.2</td>
<td>5.4</td>
</tr>
<tr>
<td>500</td>
<td>37.0x33.8</td>
<td>35.4</td>
<td>8.7</td>
</tr>
</tbody>
</table>

6.9.1.3. Recipes for SPIRE Point Source Photometry

The SPIRE pipeline and HIPE have several methods available for point source photometry, e.g. map based methods (SUSSEXtractor and DAOphot Section 6.9.1.4), timeline fitting (Section 6.9.1.5) and aperture photometry Section 6.9.1.6. Each method requires different input parameters (e.g. effective beam areas, FWHM, aperture sizes, colour and aperture corrections, etc). The rationale behind the various steps required for photometry can be found in the SPIRE Handbook (formerly the SPIRE Observers Manual), where as, in the following sections, photometry methodology specifically within the HIPE environment is explained. In Figure 6.82 and the associated table, the algorithmic steps required within HIPE to perform point source photometry are summarized for the main photometry tasks available.
Figure 6.82. Summary of Point Source Photometry methods in HIPE. The necessary inputs are summarised in the table below.

<table>
<thead>
<tr>
<th>Input</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>K4EdivK4P conversion factor</td>
<td>Table 6.9</td>
</tr>
<tr>
<td>Pipeline Beam Areas ($\alpha = -1$)</td>
<td>Table 6.10</td>
</tr>
<tr>
<td>Effective Beam ratios (beam correction)</td>
<td>Table 6.11</td>
</tr>
<tr>
<td>FWHM</td>
<td>Table 6.12</td>
</tr>
<tr>
<td>rpeak parameters</td>
<td>Table 6.13</td>
</tr>
<tr>
<td>Apertures</td>
<td>Table 6.14</td>
</tr>
<tr>
<td>Aperture Corrections</td>
<td>Table 6.15</td>
</tr>
<tr>
<td>Colour Corrections</td>
<td>Table 6.16</td>
</tr>
</tbody>
</table>

6.9.1.4. Source Extraction with SUSSEXtractor or DAOPhot

HIPE provides two tasks to carry out source extraction on the SPIRE Level 2, Level 2.5, Level 3 maps, `sourceExtractorDaophot` and `sourceExtractorSussextractor` tasks are both included within HIPE, with `sourceExtractorSussextractor` optimized for use with SPIRE maps.

These algorithms are explained in detail in the _Herschel Data Analysis Guide Section 4.18_

This section explains how to use the source extractors via the graphical interface; advanced usage is described in the _User Reference Manual_:

- DAOPHOT task
- Sussextractor task

The two tasks are listed in the Applicable folder of the Tasks view whenever an image is selected in the Variables view. Figure 6.83 shows the lists of parameters for the two tasks (hover the mouse pointer on a parameter to reveal the tooltip):
Figure 6.83. List of parameters for the two source extraction tasks.

Example

Both SUSSEXtractor and DAOphot work on the Level 2 maps. For the purposes of source extraction the point source calibrated maps \((\text{psrcPxW})\) can be used for both algorithms. For photometry, SUSSEXtractor should use the point source calibrated maps \((\text{psrcPxW})\). However, since DAOphot actually carries out aperture photometry, for optimal results, for DAOphot the maps calibrated for extended emission \((\text{extdPxW})\) should be used for photometry.

The required inputs to be specified are the input map, the FWHM, as given in Table 6.12 for map pixel sizes of 1 arcsec for the PSW, PMW, PLW arrays respectively and the detection threshold. DAOphot also requires the pipeline beam area in Table 6.10 to convert the map to surface brightness units since it is carrying out aperture photometry.
The examples below are adapted from the official SPIRE Photometry script available from the Useful Scripts menu within HIPE and shows how to extract the necessary calibration parameters from the SPIRE calibration Tree and how to perform source extraction and photometry with SUSSEXtractor and DAOphot for a single SPIRE band.

SPIRE photometry is based on the assumption of a spectrum of the form \( \nu \cdot F(\nu) = \text{constant} \). In the case of a source having a different spectral shape multiplicative corrections must be applied to any point source photometry. All the necessary corrections for photometry are contained within the SPIRE Calibration Products and can be interrogated as a function of spectral index (\( \alpha \)).

### Changing the SUSSEXtractor PRF

Note that for HIPE12, SUSSEXtractor flux densities could as much as 5 percent lower due to a change in the default PRF from 5x5 to 13x13 pixels. The original default value of 5x5 pixels for the PRF has been restored for HIPE version 13 onwards. Note a change in PRF will affect the flux density measured by SUSSEXtractor. A 5x5 PRF (the default) provides consistent flux density measurements with the SPIRE Timeline Fitter. However, the PRF can be changed using the example below;

```python
# Loading an observation (Gamma Draconis) from the HSA
obsid = 1342198160

# For a source with spectrum \( S(\nu) \) proportional to \( S^\alpha \)
# SPIRE default pipeline assumes \( \alpha = -1 \)
alpha = 2.0

# Run for an individual SPIRE band
array = "PSW"  # SPIRE Array Bands: "PSW", "PMW", "PLW"

# Loading an observation of Gamma Dra from the HSA
obs = getObservation(obsid, useHsa=True, instrument='SPIRE')

# Extract the Point Source (Jy/beam) and Extended (MJy/sr) calibrated maps
# from the Observation Context
mapPsrc = obs.level2.refs['psrc'+array].product
mapExtd = obs.level2.refs['extd'+array].product

# Values are obtained from the SPIRE calibration tree assuming
# a point source with the spectral index alpha specified above
# N.B.: spire_cal_12_2 or later version is needed
# Beam Area for pipeline (alpha=-1)
# Beam Corrections: Beam(alpha=-1)/Beam(alpha)
# Aperture Corrections derived from carrying out aperture photometry on beam map
# See the HIPE Useful Script -Photometer Calibration Bundle Point/ext source-
# for more details on the aperture corrections.
# Colour Corrections for point sources (pipeline assumes alpha=-1)
cal = spireCal()

beamCorrTable = cal.phot.refs['ColorCorrBeam'].product
aperCorrTable = cal.phot.colorCorrApertureList.refs[0].product
kCorrPsrcTable = cal.phot.colorCorrKList.refs[0].product

beamArea = beamCorrTable.meta['beamPipeline']*.title()+"Arc").double
beamCorr = beamCorrTable.getAlphaCorrection(alpha, array)
kCorrPsrc = kCorrPsrcTable.getAlphaCorrection(alpha, array)
aperCorr = aperCorrTable.getApertColorCorrection(alpha, array)

# K4p is the conversion factor (colour correction) applied in the standard
# pipeline to give flux for nu S_\nu= constant point source
# (converts from alpha=0 to alpha=-1)
# Since this factor is automatically applied in the standard photometric calibration,
```
# we need it for removal and replacement by that for extended sources.
k4P = cal.phot.fluxConvList.refs[2].product[array].meta['k4P_'+array].double

# K4e is similar to K4p but includes the frequency-dependent beam,
# making it applicable to fully extended sources with nu F nu=constant spectrum
k4E = cal.phot.fluxConvList.refs[2].product[array].meta['k4E_'+array].double

# Therefore the conversion between extended and point source calibration is
K4EdivK4P = k4E / k4P

# FWHM of point source beam profile taken from Calibration Tree
fwhm = cal.phot.getBeamProf(array).meta['FWHM_gMean%s'%array.title()].value

# radius (aperture radius needed by timeline fitter and aperture photometry)
radiusValues = {'PSW':22, 'PMW':30, 'PLW':42}
pixelSizeValues = {'PSW':6.0, 'PMW':10.0, 'PLW':14.0}

# Radii of inner and outer annulus for background estimation
innerArcsec=60.0
outerArcsec=90.0

# Select parameters for the given array
radius = radiusValues[array]
annulus = AnnulusValues[array]
pixelSize = pixelSizeValues[array]

format = '%30s = %5.1f mJy'            #Format string for printout

SUSSEXtractor can either perform source extraction on the entire map but also supports a Region of Interest (ROI), as shown below, to constrain the search area to a smaller map region.

```python
# Define region of interest (ROI) around requested position from FITS header
wcs = mapPsrc.wcs
roi = SkyMaskCircle(obs.meta['raNominal'].value, obs.meta['decNominal'].value, fwhm*3./60.)
srcSussex = sourceExtractorSussextractor(image=mapPsrc, 
               fwhm=fwhm, useSignalToNoise=True, roi=roi)

# The default PRF for SUSSEXtractor from HIPE is 5x5 pixels
# To change the PRF to a different size please use the following
# e.g. for 13x13 pixels (NOTE: the flux will change!).
# prfNew = PrfGaussian([13,13],fwhm/pixelSize)
# srcSussex = sourceExtractorSussextractor(image=mapPsrc, detThreshold=5.0, 
#                                        fwhm=fwhm, useSignalToNoise=True, prf=prfNew, roi=roi)

if len(srcSussex['sources']['ra'].data) > 1:
    print "Warning: More than one source found around requested position!"

# Get position of first source in list.
ra  = srcSussex['sources']['ra'].data[0]
dec = srcSussex['sources']['dec'].data[0]

# Apply colour correction for given alpha
fluxSussex = srcSussex['sources']['flux'].data[0] * kCorrPsrc

print format%'Flux from SUSSEXtractor', fluxSussex
```

---

130
Both DAOphot and SUSSEXtractor can detect and extract sources independently but they can also take a source list or RA and Dec as input. In the example below, DAOphot takes as input the RA and Dec of the first source found by SUSSEXtractor from the previous example above.

As emphasized earlier, DAOphot photometry should be made using the maps calibrated for extended emission (extdPxW). In order to use these maps for point source photometry the maps must be converted to a point source calibration and the units should be in Jy/pix. These two steps are shown in the example code below.

Fluxes obtained via SUSSEXtractor do not require any aperture correction. However, the fluxes obtained from DAOphot do require a multiplicative aperture correction. By default, DAOphot estimates the aperture correction automatically itself by setting the parameter doApertureCorrection = True. The automatic aperture correction is calculated from photometry on the point response function (PRF) to determine the correction. Alternatively, the user can estimate their own aperture correction using different aperture sizes within DAOphot to estimate a curve of growth or use the numbers given in Table 6.15. The aperture corrections in Table 6.15 are given for the typical case where there is a background around the source but also for the scenario where the background has been subtracted. Note that if DAOphot is run with the parameter doApertureCorrection = False then the resulting flux must also be multiplied by the effective beam ratio (beam correction), for the appropriate spectral index, $\alpha$, given in Table 6.11, in order to take into account the RSRF, the aperture efficiency and the variation of beam profile with frequency.

Note that there are also aperture corrections for arbitrary apertures (from 0 to 700 arcsec) in the SPIRE Calibration Tree in the RadialCorrBeam product under normArea Table. The corrections in this table (by means of the Encircled Energy fraction = normalized beam area) are only for an $\alpha$=−1 source, however, the colour corrections for the normalized beam are very small and in most cases can be ignored without too much concern.

```python
# For optimal results begin with MJy/sr calibrated maps
# ExtendPxW maps include the relative gain correction which flatfield the
# detectors of the arrays for their integral beam profiles rather than their peaks
mapExt = imageDivide(image1=mapExt, scalar=K4EdivK4P)
# Convert map units to [Jy/pixel] for Aperture Photometry algorithm.
mapExt = convertImageUnit(mapExt, newUnit="Jy/pixel", beamArea=beamArea)

# By default DAOphot applies an internally derived automatic aperture correction
# set with doApertureCorrection=True
srcDaoAuto = sourceExtractorDaophot(image=mapExt, detThreshold=5.0, \
    doApertureCorrection=True, fwhm=fwhm, \
    beamArea=beamArea, inputSourceList=[ra,dec], \
    innerArcsec=innerArcsec, outerArcsec=outerArcsec)

# Colour correcting results for a source with defined alpha (default being -1)
# The automatic aperture correction also corrects for the beam variation therefore
# no correction for beam variation as function(alpha) is required
fluxDaoAuto = srcDaoAuto["sources"]*[flux].data[0] * kCorrPsrc
print format%('Flux from DAOphot (auto)', fluxDaoAuto)
```

```python
# no automatic aperture correction set with doApertureCorrection=False
srcDaoTable = sourceExtractorDaophot(image=mapExt, detThreshold=5.0, \
    doApertureCorrection=False, fwhm=fwhm, \
    beamArea=beamArea, inputSourceList=[ra,dec],\
    radiusArcsec=radius, \
    innerArcsec=innerArcsec, outerArcsec=outerArcsec)
```
# Apply beam correction, colour correction, aperture correction for given alpha
fluxDaoTable = srcDaoTable["sources"]["flux"].data[0] * beamCorr * kCorrPsr * aperCorr
print format%('Flux from DAOphot (table)', fluxDaoTable)

For both SUSSEXtractor and DAOphot, the default, the point response function (PRF) is assumed to be Gaussian, with full-width-half-maximum (in arcsec) provided by the fwhm parameter. Alternatively, you can specify a custom PRF via the prf parameter. This should be a variable of type SimpleImage. The image should be of odd dimension, with the peak at the centre, normalised such that it gives the (central pixels) of a point source of flux 1 Jy, in the units of the input map. The PRF image is assumed to have the same pixel scale of the main image, and does not need to have an associated WCS.

A PRF can be extracted from the beam profile in the calibration tree. Assuming that cal is your SPIRE calibration context and myImage is your image (in Jy/beam units) for the PSW band, issue the following command:

```
myPRF = cal.phot.beamProfList.getProduct('PSW')
```

Be sure to verify that the PRF image has the peak value at the image centre. Similar commands may be used to construct PRF images for the PMW and PLW bands.

The output from the source extraction is a SourceListProduct and is called sourceList by default. You can inspect it in the Product Viewer like any other product, as shown in Figure 6.84.

To access the measured fluxes and positions directly, the SourceListProduct must be addressed in the following way flux = srcSussex["sources"]["flux"].data[0] (see also the example code snippet)

![Figure 6.84](image)

Figure 6.84. The list of sources shown in the Product Viewer, with the internal dataset highlighted.

To display the extracted sources on the image, drag and drop the sourceList variable on the image in the Editor view. A circle with fixed width is overlaid at the location of each source, as shown by the following figure. Note that dragging and dropping will not work if you select the returnPixelCoordinates checkbox in the task graphical interface. When this option is selected, the task returns source coordinates in pixels rather than astronomical coordinates.

The manipulation of the sourceList is covered in detail in the Herschel Data Analysis Guide Chapter 4, which describes how to import a sourceList from a text or FITS file, overlay a sourceList on an image and change the size and colours of the circles.
Note that both DAOphot and SUSSEXtractor can take a source list (e.g. from some ancillary data catalogue) as input. When a source extractor is given a sourceList as input, it measures fluxes and flux error for all of the positions in that list. The flux returned in the estimate of the flux of a source at that position, regardless of whether there really is a source at that position, or whether a formal detection is possible. In this way, the source extractors will effectively also return an upper limit for an undetected source. See the Herschel Data Analysis Guide Section 4.18 for more details.

**Common problems**

- **No error extension (sourceExtractorSussextractor only)**

  sourceExtractorSussextractor requires the input image to have an error extension, and if this is not present the task will fail. The error (uncertainty) in the pixel values should be determined as part of the map-making algorithm. However, an easy way to add an error extension to a SimpleImage, image, assuming the uncertainty in each pixel is 0.001, is to use:

  ```python
  image.setError(image.getImage() * 0 + 0.001)
  ```

- **Invalid units, or units not specified**

  Both source extraction tasks require the input image to specify its units in a valid format. If the task cannot recognise the units of the image as units of surface brightness then it will fail. To set the units of the SimpleImage, image, to be "Jy/beam" (for example), use `image.setUnit("Jy/beam")`. Other units based on Jy, mJy, MJy, beam, pixel, sr, etc. are recognised.

- **Reducing the number of NaNs in the DAOphot background radii**

  The default values for the DAOphot background annulus of 60-90 arcsec can produce a substantial number of NaN results in some cases. Although the current values for the background annulus produce the most consistent photometry, in severe cases, the number of NaNs can be reduced by changing the default background annulus radii as follows;

  - PSW: inner radius = 22.0 arcsec, outer radius = 33.0 arcsec
  - PSW: inner radius = 30.0 arcsec, outer radius = 45.0 arcsec
  - PSW: inner radius = 42.0 arcsec, outer radius = 63.0 arcsec

**6.9.1.5. Source Fitting of Point Sources in Timeline Data**

The sourceExtractorDaophot and sourceExtractorSussextractor source extraction algorithms work on the final image maps, however SPIRE also provides an alternative method for photometry working on the timeline data itself before the map making process. The **Timeline Source Fitter** performs photometry at a given position (or set of positions) by fitting a Gaussian to the timeline samples on the sky. The **Timeline Fitter is currently the recommended algorithm for SPIRE point source photometry**. The Timeline Source Fitter does not work on maps and does not perform source extraction. The Timeline Source Fitter requires Level 1 destriped timelines as input and the locations of sources at which to fit. The Timeline Source Fitter can be accessed via a GUI in the **Tasks window within HIPE under sourceExtractorTimeline** as shown in Figure 6.85.
Figure 6.85. GUI for the Timeline Fitter

From the GUI box in Figure 6.85, it can be seen that there are various parameters for the Timeline Fitter. The input (destripedLevel1) is a destriped (or baseline subtracted) level 1 context or Scan Context (i.e. the output from the destripper or baseline removal algorithms). The other input required is a set of RA, Dec coordinates in decimal degrees for a single source ([RA, DEC]) or a list of coordinates ([[RA1,RA2,...], [DEC1,DEC2,...]]). Alternatively a source list can be appended (see below). Additional parameters are explained below in Table 6.13.

The Timeline Source Fitter can also be run from the command line as:

```python
sourceList = sourceExtractorTimeline(input=destripedLevel1, 
array='PSW', inputSourceList=[RA,DEC], rPeak=22,
allowVaryBackground=True, useBackInFit=True,
fitMaxIterations=10000)
```

Table 6.13. Parameters for the Timeline Fitter

<table>
<thead>
<tr>
<th>Parameter Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>array</td>
<td>String parameter for SPIRE array &quot;PSW&quot;, &quot;PMW&quot;, &quot;PLW&quot;.</td>
</tr>
<tr>
<td>inputSourceList</td>
<td>Either a Double1d array with 2 entries containing the estimated RA and Dec of source in degrees. Or a Source List Product from, for example, the output of SUSSEXtractor</td>
</tr>
<tr>
<td>rpeak</td>
<td>Optional parameter for radius of the region that will include the peak of the source. Appropriate values are 22, 30, 42 for the PSW, PMW, PLW respectively and exclude the airy rings.</td>
</tr>
<tr>
<td>rbackground</td>
<td>Optional parameter (double1D with 2 entries) for the inner and outer radius of the annulus for use as background subtraction. Setting either to a negative value will result in no background being subtracted. Default value is array dependent and is PSW [70.,74.], PMW [98.,103.], PLW [140.,147.].</td>
</tr>
<tr>
<td>useBackInFit</td>
<td>Optional Boolean parameter. If True then all data samples from background annulus will be used in the fit. Otherwise the median value will be removed from the data to fit. Default value is True. Note: using useBackInFit = True will improve the fit.</td>
</tr>
<tr>
<td>allowVaryBackground</td>
<td>Optional Boolean parameter. If True then background is treated as a free parameter in the fit. This parameter is ignored if the background is ignored in the fit. Default value is True. Note: using allowVaryBackground = True will improve the fit.</td>
</tr>
<tr>
<td>allowTiltBack</td>
<td>Optional Boolean parameter. If True then a tilted plane is used for the background. Default value is False. This is ignored if allowVaryBackground is set to False</td>
</tr>
</tbody>
</table>
### Parameter Options

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fitEllipticalGauss2d</td>
<td>Optional Boolean parameter. If True then an elliptical Gaussian is fit to the data. Otherwise a circular Gaussian is used. Default value is False. Note that for relatively bright sources elliptical Gaussians are fine, however for sources fainter than approximately 30mJy, circular Gaussians should be used in preference.</td>
</tr>
<tr>
<td>modelGauss2dSigma</td>
<td>Optional Double set in degrees will perform a circular Gaussian with the sigma parameter of the Gaussian fixed (ignored if fitEllipticalGauss2d parameter set). Default value is -1.0.</td>
</tr>
<tr>
<td>fitMaxIterations</td>
<td>Optional integer parameter to set a limit on the number of iterations performed by the Levenberg Marquardt Fitter. Default value is 10000.</td>
</tr>
<tr>
<td>fitTolerance</td>
<td>Optional (Double) parameter that sets the tolerance the Levenberg Marquardt Fitter attempts to achieve when performing the fit. Default value is 1e-4.</td>
</tr>
<tr>
<td>roi</td>
<td>Region of Interest. A skymask where only the sources lying inside the ROI will be included.</td>
</tr>
<tr>
<td>slow</td>
<td>By default, if processing more than one source, the task will attempt to store all of the data in memory at once. If it runs out of memory, it will use a slower method, which involves iterating over all of the data for each source. Set this parameter to True to use this slower method without first attempting to store all of the data in memory.</td>
</tr>
</tbody>
</table>

The output from the Timeline Fitter is a standard Source List Product (default name is `sourceList`) containing results from the fitting, including position and positional errors (RA, Dec, x, y), Flux with errors (in mJy), fitted background with errors (Jy/beam), quality of the fit. The Source List Product can be written to an output ASCII file by the following command:

```python
#Save Source List as ascii file
AsciiFileResults = AsciiTableTool()
AsciiFileResults.save("SourceListTLF_PSW.dat",sourceList["sources"])
```

### Example

It is also possible to provide a source list from another source extraction algorithm, for example using SUSSEXTractor to extract the sources and passing the output source list as input for the timeline fitter as shown below (note that in this example the sources are not colour corrected):

```python
# Example of using a source list as input to the Timeline Fitter
# Get an observation from Herschel Science Archive
obs = getObservation(1342195871, useHsa=True, instrument='SPIRE')

# Get the level 2 maps
# Note: delete psrc prefix for pre-HIPE10
mapPsw=obs.level2.getProduct("psrcPSW")

# Extract sources with SUSSEXTractor
sourcesSxtPSW = sourceExtractorSussextractor(image=mapPsw, fwhm=17.6, detThreshold=5)

# Timeline Source Fitter
sourceList = sourceExtractorTimeline(input=obs.level1, array='PSW',
                                    inputSourceList=sourcesSxtPSW, rPeak=22, 
                                    allowVaryBackground=True, useBackInFit=True,
                                   )
```
Recommended Settings for the Timeline Fitter

To fit point sources in fields with flat backgrounds (such as extragalactic fields where the background is dominated by confused extragalactic sources), the following settings are recommended:

- The parameter \( r_{\text{Peak}} \) should be set to 22, 30, and 42 for PSW, PMW, and PLW, respectively. This aperture was selected because it includes the Gaussian part of the PSF (the region from the centre up to the gap between the peak and the first Airy ring).

- The parameter \( r_{\text{background}} \) has different defaults for each array. The values will potentially work for most sources, as the signal in this smaller annuli will be dominated by background noise except for very bright sources.

- The parameter \( \text{AllowTiltBack} \) should be set to \text{False}.

- The parameter \( \text{fitEllipticalGauss2d} \) should normally be set to \text{False}. This is only useful if attempting to distinguish between elongated and truly unresolved sources.

- The parameter \( \text{modelGauss2dSigma} \) depends on the source brightness and the scan speed. For slow or nominal scan speeds, for sources brighter than 35mJy in the PSW band, 50 mJy in the PMW and PLW bands, \( \text{modelGauss2dSigma} \) should not be set. Allowing the FWHM of the Gaussian to vary usually produces a more accurate fit. However, users should check that the FWHM from the fit falls within nominal ranges for the data (15-21 arcsec for PSW, 19-28 arcsec for PMW, and 28-40 arcsec for PLW). For sources fainter than 35 mJy in the PSW band or 50 mJy in the PMW or PLW bands or for sources observed using the fast scan speed, \( \text{modelGauss2dSigma} \) should be set to 17.6 for PSW, 23.9 for PMW, and 35.2 for PLW. Fixing the width of the PSF normally produces more accurate results in these cases.

- The parameter \( \text{useBackInFit} \) should be set to \text{True}.

- The parameter \( \text{allowVaryBack} \) should be set to \text{True}.

The following optional set-ups can also be used to handle the background:

- The parameter \( \text{allowVaryBack} \) can be set to \text{False} while still specifying \( r_{\text{Background}} \) and setting \( \text{useBackInFit} \) to \text{True}. In this scenario, the background will not be treated as a free parameter in the fit, an initial background offset will be measured and subtracted before the data is fitted.

- The parameter \( \text{allowVaryBack} \) can be set to \text{False} and \( \text{useBackInFit} \) can be set to \text{False} if \( r_{\text{Background}} \) is specified. The data in the background annulus will still be used to measure and subtract an initial background before a Gaussian function is fit to the PSF, but the data in the background annulus will not be used in the fit, and the background will not be treated as a free parameter.

- The parameter \( r_{\text{Background}} \) does not need to be given as an input, but \( \text{allowVaryBack} \) should be set to \text{False} and \( \text{useBackInFit} \) should be set to \text{False}. This is appropriate if the background has already been subtracted (for example, by using removeBaselines).

To fit sources in regions with significant background variations (such as sources observed in fields with thick cirrus emission), the following settings are recommended:

- The parameter \( r_{\text{Background}} \) should not be specified. No background annulus should be used in the PSF fitting.

- The parameter \( \text{allowTiltBack} \) should be set to \text{True}.

```python
fitMaxIterations=10000)

# Write to ASCII file
AsciiFileResults = AsciiTableTool()
AsciiFileResults.save("/Users/cpearson/"+"SourceListTLF_PSW.dat",sourceList["sources"]
```
• The parameter fitEllipticalGauss2d should be set to False.
• The parameter useBackInFit should normally be set to False.
• The parameter allowVaryBack should normally be set to False.

6.9.1.6. Performing Aperture Photometry on SPIRE Images

Aperture photometry of both point source and extended sources can be made using 3 tasks in HIFI, the annularSkyAperturePhotometry, rectangularSkyAperturePhotometry, fixedSkyAperturePhotometry tasks. These algorithms are explained in detail in the Herschel Data Analysis Guide Section 4.20. In the following 2 sections specific information is provided for aperture photometry on point sources and extended emission respectively.

Recipe for Point Source Aperture Photometry

Although, in principle, aperture photometry is not recommended for point sources in SPIRE maps, the framework to do so does exist. To measure integrated flux densities of point sources via aperture photometry, the following steps are performed (the explicit algorithmic steps within HIFI are also shown in Figure 6.82 and the HIFI script is shown below);

1. The starting point is the Level 2 \texttt{extdPxW} extended emission calibrated maps in MJy per steradian.

2. Convert to point source calibration

SPIRE maps for extended emission have been calibrated for a monochromatic extended source surface brightness using the $K4EdivK4P$ parameter described in Section 6.9.1.2 and Table 6.9. Therefore, the maps must first be divided by the $K4EdivK4P$ parameter.

3. Divide the image by the beam area to convert the image to Jy/pixel, for an image with small (<1") pixels using the pipeline beams for 1" pixels given in Table 6.10. Note that HIFI provides a specific task \texttt{convertImageUnit}, available from the \texttt{tasks} window, to divide the image by the beam and convert the units.

4. Measure the integrated flux density within the desired aperture and background annulus, using the appropriate aperture photometry task as explained in Herschel Data Analysis Guide Chapter 4. For point sources the recommended aperture radii are selected to contain just the main lobe of the beam, and are given below in Table 6.14 along with the values for the annulus to estimate the background level.

5. The SPIRE flux calibration assumes a flat spectrum for the source ($\nu . F(\nu) = \text{constant}$). For other spectral indices, the flux density needs to be multiplied by the appropriate beam ratios $\Omega(\alpha=-1)/\Omega(\alpha)$ for given in Table 6.11, in order to take into account the RSRF, the aperture efficiency and the variation of beam profile with frequency.

6. The SPIRE flux calibration assumes a flat spectrum for the source ($\nu . F(\nu) = \text{constant}$). For other spectral indices, the flux density needs to be multiplied by the appropriate point source color correction for the assumed spectral index using the Colour Corrections for Point Sources given in Table 6.16.

7. Aperture correction factors are required, since this method measures the sky brightness in a fraction of the beam, and therefore underestimates the integrated flux density. The corrections have been tested and bring the integrated flux densities obtained through aperture photometry into agreement with those obtained from fitting timeline data, which is in general a more accurate method. The aperture corrections are also listed in Table 6.15. Note that there are also aperture corrections for arbitrary apertures (from 0 to 700 arcsec) in the SPIRE Calibration Tree in the \texttt{RadialCorrBeam} product under \texttt{normArea} Table. The corrections in this table (by means of the Encircled Energy fraction = normalized beam area) are only for an $\alpha=-1$ source, however, the colour corrections for the normalized beam are very small and in most cases can be ignored without too much concern.
Example

The above procedure is shown in the script below and is adapted from the official SPIRE Photometry script available from the Useful Scripts menu within HIPE. Note as with the earlier examples for SUSSEXtractor and DAOPhot, all the necessary calibration is contained within the SPIRE Calibration Tree which can be interrogated as a function of spectral index, $\alpha$. In the example for Aperture Photometry below, the source position for photometry is supplied via SUSSEXtractor, although, in practice could just as easily be supplied as a string variable for RA and Dec respectively.

```
# Loading an observation (Gamma Draconis) from the HSA
obsid = 1342198160

# For a source with spectrum $S(\nu)$ proportional to $S^{\alpha}$
# SPIRE default pipeline assumes $\alpha = -1$
alpha = 2.0

# Run for an individual SPIRE band
array = "PSW"  # SPIRE Array Bands: "PSW", "PMW", "PLW"

# Import Data
obs     = getObservation(obsid, useHsa=True, instrument='SPIRE')
# For observation from your own local pool use the following line instead
# obs     = getObservation(obsid, poolName='mypool', instrument='SPIRE')

# Extract the Point Source (Jy/beam) and Extended (MJy/sr) calibrated maps
# from the Observation Context
mapPsrc = obs.level2.refs["psrc"+array].product
mapExtd = obs.level2.refs["extd"+array].product

# Correction Parameters
# Values are obtained from the SPIRE calibration tree assuming a point source with the spectral index $\alpha$ specified above
# N.B.: spire_cal_12_02 or later version is needed
# Beam Area for pipeline ($\alpha=-1$)
# Beam Corrections: Beam($\alpha=-1$)/Beam($\alpha$)
# Aperture Corrections
# Colour Corrections for point sources (pipeline assumes $\alpha=-1$)
cal = spireCal()
beamCorrTable  = cal.phot.refs["ColorCorrBeam"].product
aperCorrTable  = cal.phot.colorCorrApertureList.refs[0].product
kCorrPsrcTable = cal.phot.colorCorrKList.refs[0].product
beamArea  = beamCorrTable.meta["beamPipeline"+array.title()]+"Arc".double
beamCorr  = beamCorrTable.getAlphaCorrection(alpha, array)
kCorrPsrc = kCorrPsrcTable.getAlphaCorrection(alpha, array)
aperCorr  = aperCorrTable.getApertColorCorrection(alpha, array)

# K4p is the conversion factor (colour correction) applied in the standard pipeline to give flux for nu S_nu= constant point source
# (converts from alpha=0 to alpha=-1)
# Since this factor is automatically applied in the standard pipeline,
# we need it for removal and replacement by that for extended sources.
k4P = cal.phot.fluxConvList.refs[2].product[array].meta["k4P_"+array].double

# Therefore the conversion between extended and point source calibration is K4EdivK4P = k4E / k4P

# FWHM of point source beam profile taken from Calibration Tree
fwhm = cal.phot.getBeamProf(array).meta["FWHM_gMean%s"%array.title()].value
```
# radius (aperture radius needed by timeline fitter and aperture photometry)
# Default map pixel size
radiusValues = {"PSW":22, "PMW":30, "PLW":42}
pixelSizeValues = {"PSW":6.0, "PMW":10.0, "PLW":14.0}

# Radii of inner and outer annulus for background estimation
innerArcsec=60.0
outerArcsec=90.0

########################################################################
# Select parameters for the given array
radius = radiusValues[array]
pixelSize = pixelSizeValues[array]
format = '%30s = %5.1f mJy'            #Format string for printout
########################################################################

###########################################################################
# Running Sussextractor
###########################################################################
# Define region of interest (ROI) around requested position from FITS header
# Find source position within 3 * FWHM of requested position
# Alternatively remove ROI and provide source position as inputSourceList=[ra,dec]
wcs = mapPsrc.wcs
roi = SkyMaskCircle(wcs.crval1, wcs.crval2, fwhm*3./60.)
srcSussex = sourceExtractorSussextractor(image=mapPsrc, 
detThreshold=5.0, fwhm=fwhm, roi=roi)

# The default PRF for SUSSEXtractor from HIPE is 5x5 pixels
# To change the PRF to a different size please use the following
# e.g. for 13x13 pixels (NOTE: the flux will change!).
#prfNew = PrfGaussian([13,13],fwhm/pixelSize)
#srcSussex = sourceExtractorSussextractor(image=mapPsrc, detThreshold=5.0, 
#                                        fwhm=fwhm, prf=prfNew, roi=roi)

if len(srcSussex["sources"]["ra"].data) > 1:
    print "Warning: More than one source found around requested position!"
# Get position of first source in list.
ra  = srcSussex["sources"]["ra"].data[0]
dec = srcSussex["sources"]["dec"].data[0]

###########################################################################
# Aperture Photometry
###########################################################################
# For optimal results begin with MJy/sr calibrated maps
# ExtdPxW maps include the relative gain correction which flatfield the
# detectors of the arrays for their integral beam profiles rather than their peaks
# Remove colour correction for extended sources and apply that for point sources
mapExtd = imageDivide(image1= mapExtd, scalar=K4EdivK4P)
mapExtd = convertImageUnit(mapExtd, newUnit="Jy/pixel", beamArea=beamArea)

###########################################################################
# Running Annular Aperture Photometry
###########################################################################
# We use standard radii for aperture and background annulus
resultAper = annularSkyAperturePhotometry(image=mapExtd, 
centerRA=\%s\%(ra), centerDec=\%s\%(dec),\nfractional=1, centroid=False, radiusArcsec=radius, 
innerArcsec=innerArcsec, outerArcsec=outerArcsec)

# Apply beam correction, colour correction, aperture correction for given alpha
fluxAper = resultAper.getTargetTotal() * 1.03 * beamCorr * aperCorr * kCorrPsrc

print format%(\'Flux from Aperture Photometry\', fluxAper)

print
###########################################################################
# End of script
###########################################################################
Recipe for Aperture Photometry on Extended Emission Maps

To measure integrated flux densities of extended sources or to work with surface density measurements for extended emission (see the SPIRE Handbook for details of the calibration for extended emission), the following steps should be performed (the explicit algorithmic steps within HIPE are also shown in Figure 6.86 and the HIPE script is shown below):

1. Start from the Level 2 `extdPxW` extended emission calibrated maps in MJy per steradian.

2. Convert image units to Jy/pix

   The HIPE Aperture Photometry task requires images specifically in units of Jy/pixel. Convert the image to units of Jy/pixel by using the `convertImageUnit` task, available from the HIPE tasks window. Since the extended emission maps were originally in MJy/sr, no beam area is required.
3. The SPIRE flux calibration assumes a flat spectrum for the source \( \nu F(\nu) = \text{constant} \). For other spectral indices, the flux density needs to be multiplied by the appropriate color correction for **Extended Emission** for the assumed spectral index using the Colour Corrections for Extended Sources given in Table 6.16. Note that for extended emission, the colour correction also takes into account the various beam effects so the values in Table 6.11 are not required.

4. Measure the flux density within the desired aperture if integrated flux densities are required. The aperture photometry tasks are explained in [Herschel Data Analysis Guide Chapter 4](#).

**Example**

The procedure for aperture photometry for extended emission is summarized below for a single SPIRE band. To run the script for a different SPIRE band, simply edit the line array = "PSW" # SPIRE Array Bands: "PSW", "PMW", "PLW".

```python
# Example for performing Aperture Photometry for Extended Emission within HIPE

# Loading an observation (Gamma Draconis) from the HSA
obsid = 1342189432

# For a source with spectrum S(\nu) proportional to S^{alpha}
# SPIRE default pipeline assumes alpha = -1
alpha = -3

# Run for an individual SPIRE band
array = "PSW"  # SPIRE Array Bands: "PSW", "PMW", "PLW"

# Loading an observation of Gamma Dra from the HSA
obs = getObservation(obsid, useHsa=True, instrument='SPIRE')

# Extract Extended (MJy/sr) calibrated maps from the Observation Context
mapExtd = obs.level2.refs["extd"+array].product

# Values are obtained from the SPIRE calibration tree assuming
# a extended source with the spectral index alpha specified above
# N.B.: spire_cal_12_0 or later version is needed
# Beam Area for pipeline (alpha=-1)
beamArea = beamCorrTable.meta["beamPipeline"+array.title()+"Arc"].double

# Colour Corrections for extended sources (pipeline assumes alpha=-1)
cal = spireCal()
beamCorrTable = cal.phot.refs["ColorCorrBeam"].product
kCorrExtTable = cal.phot.colorCorrKList.refs[0].product

# Target RA and Dec and string values
ra = "6.659144"
dec = "64.20587616"

# Colour Correct the map for a alpha=-3 spectral index
```
mapExtendedCorrected = imageMultiply(image1=mapExtended, scalar=kCorrExtd)

# Carry out the Aperture Photometry
annularPSW = annularSkyAperturePhotometry(image= mapExtendedCorrected, \
    centerRA=ra, centerDec=dec, fractional=1, \
    radiusArcsec=350.0, innerArcsec=400.0, outerArcsec=450.0)

# Final target brightness in Jy/pixel
flux = annularPSW.getTargetTotal()
print 'PSW flux = %5.3f Jy' % (flux)

Figure 6.86. Summary of Photometry methods for Extended Emission in HIPE. The necessary inputs are summarised in the table below.

<table>
<thead>
<tr>
<th>Input</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>K4Ed/K4P conversion factor</td>
<td>Table 6.9</td>
</tr>
<tr>
<td>Pipeline Beam Areas ($\alpha = -1$)</td>
<td>Table 6.10</td>
</tr>
<tr>
<td>Colour Corrections</td>
<td>Table 6.16</td>
</tr>
</tbody>
</table>
6.9.1.7. Image Map Gaussian Source Fitting in HIPE

HIPE also provides a general task for the fitting of Gaussians to a source position in an image map. Note that this is a general task within HIPE and not a specific SPIRE photometry task, therefore this task is not generally recommended for use on SPIRE data.

The `sourceFitting` task will fit a 2-D Gaussian to a source in a specified rectangular region in an image map. The code example below shows the typical call to the `sourceFitting` task. The inputs to the task are the min x-pixel and y-pixel position of the rectangle, the width and height of the rectangle in pixels, whether the source is assumed to be elongated and whether the background has a gradient. Note that since the Gaussian fitting task is measuring data that has been mapped onto pixels the appropriate pixelization factors need to be applied to account for the Gaussian fit underestimating the peak flux (other SPIRE photometry tasks in HIPE do not require this correction).

```python
obsid=1342226998
obs=getObservation(obsid,useHsa=True)
mapPSW = obs.level2.refs["psrcPSW"].product

# Get a source position with SUSSEXtractor
fwhmPSW = 18.4
sourceListPSW = sourceExtractorSussextractor(image=mapPSW, detThreshold=5.0, 
       fwhm=fwhmPSW, fitBackground=True)

# input to Source Fitting is an assumed source position
SXTflux = sourceListPSW["sources"]["flux"].data[0]
x = sourceListPSW["sources"]["x"].data[0]
y = sourceListPSW["sources"]["y"].data[0]

boxSize = 5  # width of fitting box in pixels

# Source Fitting Task
ps = sourceFitting(elongated=False, image=mapPSW, 
       minX=x-boxSize, minY=y-boxSize, 
       width=2*boxSize+1, height=2*boxSize+1)

# Pixelization Correction
pixelCorrPSW = 0.941
pixelCorrPMW = 0.917
pixelCorrPLW = 0.903

# *1000 to convert to mJy
# divide by pixelization correction
fittedFlux = ps["Column1"][0]*1000/pixelCorrPSW

# Compare with SUSSEXtractor flux
print "SUSSEXtractor flux = %f"%(SXTflux)
print "Fitted flux = %f"%(fittedFlux)
```

The output of the `sourceFitting` task is a parameters box containing the fitted flux in Jy/beam, centre x,y and RA, Dec positions, Gaussian width in both pixels and arcsec and the measured background in Jy/beam (see Figure 6.87).
SPIRE photometry is based on the assumption of a spectrum of the form $F(\nu) = \text{constant}$. In the case of a source having a different spectral shape multiplicative corrections must be applied to any point source photometry. To obtain the colour correction for a given spectral shape the following values in Table 6.16 must be used to multiply the data to get the correct photometry.

Similarly, for extended emission the colour correction values in Table 6.16 must be used to multiply the data to get the correct photometry.

Further examples of colour corrections for grey bodies with varying values of $\beta$ and temperatures can be found in the SPIRE Handbook and can be accessed within the SPIRE Calibration Tree. Colour corrections can be accessed from the SPIRE CAlibration Tree using the following commands;

```python
cal = spireCal()
# Load Point Source Colour Correction Table
kCorrPsrcTable = cal.phot.colorCorrKList.getProduct("point")
array = "PSW"
alpha = 2
beta =1.5
tempK =100
# Query Colour Corrections for a given spectral index alpha
kCorrPsrcAlpha = kCorrPsrcTable.getAlphaCorrection(alpha, array)
# Query Colour Corrections for a given grey body Beta and Temperature
kCorrPsrcBeta = kCorrProduct.getTempBetaCorrection(tempK, beta, array)
```

<table>
<thead>
<tr>
<th>Spectral Index ($F(\nu) = \nu^\alpha$)</th>
<th>Point Source Colour Correction</th>
<th>Extended Emission Colour Correction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PSW</td>
<td>PMW</td>
</tr>
<tr>
<td>-4</td>
<td>0.98</td>
<td>0.9791</td>
</tr>
<tr>
<td>-3.5</td>
<td>0.9884</td>
<td>0.9875</td>
</tr>
<tr>
<td>-3</td>
<td>0.9948</td>
<td>0.994</td>
</tr>
<tr>
<td>-2.5</td>
<td>0.9992</td>
<td>0.9986</td>
</tr>
<tr>
<td>-2</td>
<td>1.0016</td>
<td>1.0011</td>
</tr>
</tbody>
</table>
### 6.9.1.9. Calibration Beam Information Bundle

From HIPE version 13 onwards, additional scripts are provided to enable advanced Users to produce calibration corrections for beam corrections and colour corrections for a wide variety of input spectra. Two scripts are available from the HIPE Useful Scripts menu (see Figure 6.88). The Photometer Calibration Bundle point/ext source script produces results for point and extended sources. The Photometer Calibration Bundle semi-ext source script produces results for semi-extended sources. These scripts also Users to implement the beam model. Users should consult the scripts and the (SPIRE Handbook) for more information.
6.9.1.10. List Driven Photometry

SPIRE maps are normally very crowded (confused) which can make the measurement of faint flux densities problematical. A standard method of dealing with this confusion is to take source positions derived from ancilliary data at shorter wavelengths and simultaneously find the SPIRE fluxes for these sources via a matrix inversion technique (e.g Roseboom et al., 2010, MNRAS, 409, 48).

The `sourceExtractorSimultaneous` task within HIPE allows for such simultaneous list driven photometry. The most straightforward way to use the `sourceExtractorSimultaneous` task is outlined in Figure 6.89, where the source list obtained in the SPIRE 250 micron band is used to derive flux densities in the SPIRE PLW band image.

The `sourceExtractorSimultaneous` task requires the following inputs (see Figure 6.90): an input image, an input list in the form of a `sourceListProduct` or alternatively positions as `[RA, Dec]`, and the beam FWHM. Optional inputs are the `beamArea`, `prf`, `roi` and `fitBackground` parameter which are all the same as used in the `SUSSEXtractor` task (described in Section 6.9.1.4). The output of the task is a `sourceListProduct` containing positions and the measured flux densities of the sources at the supplied input positions. A simple code example is shown below:

```python
obsid=1342226998
obs=getObservation(obsid,useHss=True)

mapPSW = obs.level2.refs['psrcPSW'].product
mapPLW = obs.level2.refs['psrcPLW'].product
fwhmPSW = 18.416
fwhmPLW = 36.701

srcSussexPSW = sourceExtractorSussextractor(image=mapPSW, detThreshold=5.0, fitBackground=True)

sourceList = sourceExtractorSimultaneous(image=mapPLW, inputSourceList=srcSussexPSW, fwhm=fwhmPLW, fitBackground=True)
```

Note that the memory required increases approximately as \( n_{\text{Sources}} \times n_{\text{Sources}} \) of the input source list so small chunks of a map are preferable rather than many degrees. The final simultaneously measured flux list should be examined carefully for negative sources (especially around the edge of the map).

Figure 6.89. Measuring source fluxes with list driven photometry.
6.10. Absolute Calibration of SPIRE Maps

6.10.1. SPIRE Map Zero Point from Herschel-Planck Calibration

6.10.1.1. Introduction

Herschel-SPIRE detectors are only sensitive to relative variations, as a consequence the absolute brightness of the observed region is unknown and maps are constructed such that they have zero median.

Although the Planck-HFI detectors are similar to SPIRE, Planck's observing strategy allows it to (almost) observe a sky's great circle every minute (having a 1 rpm spinning rate). By comparing the sky brightness as measured by COBE-FIRAS at the galactic poles (where the dust emission is lower), HFI is capable of setting an absolute offset to its maps. Moreover, since SPIRE and HFI share two channels with overlapping wavebands it is possible to use the Planck HFI maps to obtain an absolute calibration for the SPIRE maps.

The details of the Herschel-Planck cross calibration are covered extensively in the SPIRE Handbook (formerly the SPIRE Observers Manual) and here we explain the algorithm used to create the absolute calibrated SPIRE maps (referred to as extdPxW in a SPIRE observation).

6.10.1.2. The Zero Point Correction Task

As of HCSS 10, a new task named zeroPointCorrection is available that will calculate the absolute offset for a SPIRE map based on cross-calibration with HFI-545 and HFI-857 maps, colour-correcting HFI to SPIRE wavebands assuming a grey body function with fixed beta.

In addition a new Useful Script, "Photometer_ZeroPointCorrection.py" is available to enable the production of these Level-2 absolute calibrated maps for extended emission from the Level 1 timelines.
The zeroPointCorrection task is shown in Figure 6.91 and estimates the absolute offset of SPIRE maps, in MJy/sr, comparing them to the equivalent sky area as observed by HFI. To run properly, the task requires the following input:

- A level2 context with 3 maps named either psrcPxW or simply PxW. This ideally should come from a destriped Level 1 with relative gain correction applied. The output of the task will be the input level2 with appended the extdPxW maps.

- The two Planck HFI-545 and HFI-857 full-sky maps are assumed to have been downloaded and linked to the HIPE build. The Useful Script has initialization code that can be run the first time to automatically download the Planck maps and set up the appropriate links (Note that a restart of HIPE may be required the first time the script is run). Alternatively the Planck maps can be downloaded from the HSC website.

- Planck–SPIRE Colour Correction calibration file (contained within the SPIRE Calibration Tree) to make the colour correction factors to go from HFI-545 to PLW, HFI-857 to PMW and HFI-857 to PSW bands.

- SPIRE Flux Calibration product which contains the necessary parameters (K4P, K4E, see Section 6.9.1.1) to convert point source to extended calibrated maps.

- As an option, the user may also change the full width to half-maximum (FWHM) of the 2D Gaussian function used in the convolution of SPIRE maps to HFI beam (default value: 8 arcmin) and the gain correction factors (default values: HFI-545 = 1.029, HFI-857 = 0.991, where a value of 1 would mean that no correction is applied).

The level2,flux conversion product and Colour Correction table can either be dragged to the dialogue window in Figure 6.91 directly or alternatively, if they exist, can be dragged from the variable window (as shown in Figure 6.91). These products can be generated with the following lines of code, assuming an observation obs has already been read into memory:

```python
cal = obs.calibration
# or
# cal = spireCal()
level2 = obs.level2
colorCorrHfi = cal.phot.colorCorrHfi
fluxConv = cal.phot.fluxConvList.getProduct(obs.meta["biasMode"].value, obs.startDate)
```

Figure 6.91. Zero point correction task dialogue
6.10.1.3. The Zero Point Correction Task: Initial Set Up

The very first time the Zero Point Correction Task is used (either in the Useful Script or User Pipeline) the Planck Maps need to be downloaded locally and the zero-point correction environment needs to be set up. Although, the Planck maps can be retrieved from the Herschel Science Centre manually, the Useful Script, "Photometer_ZeroPointCorrection.py" provides all the code to efficiently set up the environment:

```python
import urllib, os
planckMapPath = '/path/for/planck/maps/

hfi545Map = "DX11d_map_545_smooth_8arcmin.fits"

hfi857Map = "DX11d_map_857_smooth_8arcmin.fits"

urllib.urlretrieve("ftp://ftp.sciops.esa.int/pub/hsc-calibration/SPIRE/PHOT/  \
   HFI_Maps/"+hfi545Map, os.path.join(planckMapPath,hfi545Map))

urllib.urlretrieve("ftp://ftp.sciops.esa.int/pub/hsc-calibration/SPIRE/PHOT/  \
   HFI_Maps/"+hfi857Map, os.path.join(planckMapPath,hfi857Map))

Configuration.setProperty("spire.spg.hfi.545map",planckMapPath+hfi545Map)
Configuration.setProperty("spire.spg.hfi.857map",planckMapPath+hfi857Map)
Configuration.save("user.props", ["spire.spg.hfi.545map","spire.spg.hfi.857map"])
```

The code imports the necessary libraries into HIPE and then fetches the HFI 545GHz and 857GHz Planck maps from the ESA FTP site and stores them in a path specified by the variable `planckMapPath`. The location of the Planck maps is then hard-coded into the configuration properties `spire.spg.hfi.545map` and `spire.spg.hfi.857map`, stored in your `user.props` (usually found in your home directory within a `.hcss` directory). Note that a restart of HIPE may be required to make HIPE recognize the new set-up.

6.10.1.4. Creating Absolute Calibrated SPIRE maps

Then, the `zeroPointCorrection` runs the following steps, also illustrated in Figure 6.92

```python
```
Figure 6.92. Step-by-step description of the calculations performed internally by the zeroPointCorrection task to estimate the absolute offset to add to the SPIRE maps.

1. It loads the the three SPIRE maps $psrcPLW$, $psrcPMW$ and $psrcPSW$ and converts them from Jy/beam to MJy/sr using the $K_{monP}$ and $K_{monE}$ factors;

2. In order to avoid edge effects, it enlarges the SPIRE maps by 0.5 degrees in each direction (i.e. it adds a ring of 1 deg$^2$);

3. It reads the HFI maps, extracts and re-projects HFI data based on SPIRE astrometry;

4. It applies the colour correction factors $K_{545 \rightarrow PMW}$, $K_{857 \rightarrow PMW}$, $K_{857 \rightarrow PSW}$ to HFI data. The colour correction factors are computed, pixel by pixel, interpolating the ratio $R_{HFI}$ obtained from the colour correction table and comparing it to the ratio of maps $HFI-545/HFI-857$.

5. The task convolves SPIRE maps with a 2D Gaussian beam function, which by default has an FWHM of 8 arcmin;

6. Then, the difference maps (HFI-545 -> PLW), (HFI-857 -> PMW) and (HFI-857 -> PSW) are computed: a first estimation of the PLW, PMW and PSW offsets, respectively, is obtained as the median of the these difference maps.

7. The offsets are applied full-resolution SPIRE maps obtained in step 1 above;

8. These SPIRE maps are then embedded in the HFI colour corrected maps obtained in step 4.

9. The task runs the convolution on SPIRE maps embedded into HFI ones;
10. SPIRE maps are de-embedded: the result is the original SPIRE maps, convolved with the defined Gaussian beam and with edge effects reduced, especially for maps smaller than ~1 deg².

11. The difference maps (HFI-545 -> PLW), (HFI-857 -> PMW) and (HFI-857 -> PSW) are computed: a second estimation of the PLW, PMW and PSW offsets, respectively, is obtained as the median of the these difference maps. The statistical error on the offset estimation is computed as the difference map standard deviation.

12. Finally, the total offsets are applied to the full resolution SPIRE maps extdPLW, extdPMW and extdPSW. Their values, as well as their estimated error, are reported in the maps metadata and the level-2 context is updated.

As described above, the zeroPointCorrection task computes the offset as the median of the difference map between an HFI colour corrected map and the corresponding SPIRE one. However, this method works under the assumption that the relative gain between SPIRE and HFI is 1. Nevertheless, extended testing has shown that a relative gain is present: with an assumed colour correction table available within the SPIRE Calibration Tree, it is estimated to be HFI-545/PLW ~ 1.029 and HFI-857/PMW ~ 0.991. As a consequence, it is suggested to use these values when launching the zeroPointCorrection task.

6.10.1.5. Output Maps from the Zero-Point Correction Task

In addition to the final absolute calibrated SPIRE maps, the Zero Point Correction task also outputs various auxiliary maps created during the processing steps. These maps are summarized below and illustrated in Figure 6.93.

The Useful Script, Photometer Map Zero Point Correction shows the code necessary to access these auxiliary products:

```python
print "Running the zero point correction task"
zeroPointMaps, zeroPointParam=zeroPointCorrection(level2=level2ZeroPoint, \
    hfiFwhm=hfiFwhm, hfi545Gain=hfi545Gain, hfi857Gain=hfi857Gain, \
    colorCorrHfi=colorCorrHfi, fluxConv=fluxConv, \
    hfi545Map=hfi545Map, hfi857Map=hfi857Map)
```

# Optional auxiliary outputs from the ZeroPointCorrection task

# SPIRE PLW map in MJy/sr with enlarged surrounding area padded with NaNs
enlargedPlw = zeroPointParam.getProduct("PLW").enlarged

# Interpolated and colour corrected Planck-HFI map for PLW filter
hfiMapPlw = zeroPointParam.getProduct("PLW").hfi

# Combination of SPIRE PLW map and Planck-HFI map before convolution
embeddedPlw = zeroPointParam.getProduct("PLW").embedded

# Final zero point corrected map (same as zeroPointMaps)
finalPlw = zeroPointParam.getProduct("PLW").final

1. **Enlarged SPIRE Map:** SPIRE map in MJy/sr with enlarged surrounding area padded with NaNs. For PSW this is a regridded map to 10 arcsec pixels, matching the PMW wcs. All other maps have standard pixel sizes. The map is accessed as zeroPointParam.getProduct("Psw").enlarged

2. **HFI Map:** Planck-HFI map derived from 545GHz and 857GHz maps, convolved to 8' Gaussian resolution, colour corrected for the respective SPIRE filter, and interpolat-
ed to the smaller pixels of the enlarged SPIRE map in MJy/sr. Note that for PSW this is a wavelength extrapolation and larger errors are expected. The map is accessed as zeroPointParam.getProduct("PxW").hfi

3. **Embedded Map:** SPIRE map before convolution embedded into colour corrected / interpolated Planck-HFI map. An offset was added to the SPIRE map derived from a first estimate (first iteration) to improve the match. The map is accessed as zeroPointParam.getProduct("PxW").embedded

4. **De-embedded Map:** Extraction of all pixels that belong to the original SPIRE map from the SPIRE/Planck-HFI combined map convolved with an 8" Gaussian. The map is accessed as zeroPointParam.getProduct("PxW").deEmbedded

5. **Difference Map:** Difference map of de-embedded SPIRE map and colour corrected / interpolated Planck-HFI map. This represents the difference map of the second iteration. The map is accessed as zeroPointParam.getProduct("PxW").difference

6. **Final Calibrated SPIRE Map:** The final zero point corrected map (same as zeroPointMap) derived as adding the sum of the offsets from the two iterations to the original SPIRE image that is also the basis for the enlarged map. The map is accessed as zeroPointParam.getProduct("PxW").final

![Output maps from the Zero Point Correction Task](image)

**Figure 6.93. Output maps from the Zero Point Correction Task**

The offset appearing in the metadata as “zPointOffset” can also be derived as

```python
final = zeroPointParam.getProduct("PxW").final
enlarged = zeroPointParam.getProduct("PxW").enlarged
differenceMap = imageSubtract(image1=final, ref=1, image2=enlarged)
print MEDIAN(NAN_FILTER(differenceMap.image)), final.meta['zPointOffset'].value
```

Both values should be the same.
The offset from the first iteration can be derived as:

```python
embedded = zeroPointParam.getProduct("PxW").embedded
differenceMap1 = imageSubtract(image1=embedded, ref=1, image2=enlarged)
print MEDIAN(NAN_FILTER(differenceMap1.image))
```

The offset from the second iteration can be derived as:

```python
difference = zeroPointParam.getProduct("PxW").difference
print MEDIAN(NAN_FILTER(difference.image))
```

for all arrays. For PSW this is the only iteration that is performed.

The sum of both iterations yields the total offset added.

```python
print MEDIAN(NAN_FILTER(differenceMap1.image)) + MEDIAN(NAN_FILTER(difference.image)), final.meta['zPointOffset'].value
```

Note that there is a caveat for absolute calibrated maps made in SPIRE Small Map Mode. In this case, the Planck beam for the HFI-857 is comparable to the size of a SPIRE Small Map. For most small maps, there is no problem, however, care should be taken in the vicinity of bright sources and users can check by inspecting either the HFI Map or SPIRE Embedded Map auxiliary outputs from the zeroPointCorrection task which cover the surrounding area around the small map.

---

### 6.11. Tools for Manipulating SPIRE Maps

#### 6.11.1. SPIRE Mapping World Coordinate System

SPIRE photometer data always includes the corresponding equatorial (RA/Dec) coordinates for an individual bolometer at each sample time. World Coordinate System (WCS) parameters (Greisen and Calibretta, 2002) are used to project this spherical celestial-coordinate data into a 2-dimensional (x,y) map. The WCS parameters are defined by a number of FITS keywords. Prior to the HIPE 11 release the (x,y) map frame always represented a range of equatorial coordinates but may now alternatively represent the transform of these coordinate values into either of the other supported (galactic or ecliptic) celestial frames.

#### 6.11.1.1. SPIRE WCS Parameters

As an example of a complete SPIRE WCS defined by its FITS parameter values:

```hipe
HIPE>print Wcs
World Coordinate System
------------------------
naxis : 2
ctype1 : RA---TAN
ctype2 : DEC---TAN
crota2 : 0.0
cdelt1 : -0.0016666666666666668
cdelt2 : 0.0016666666666666668
crval1 : 215.11653306366367
crval2 : 52.629081799736426
```
naxis1 : 692
naxis2 : 2008
crpix1 : 268.0
crpix2 : 1005.0
equinox : 2000.0

These keywords and their values define:

- **CTYPE1, CTYPE2** = the map “x-axis” and y-axis coordinate system and Celestial to Planar “Map Projection”
  - The default values (“RA---TAN” and “DEC-TAN”) define Equatorial map coordinates and a tangential projection.
- **NAXIS** = Number of map axes (2).
- **EQUINOX** = Coordinate equinox (2000.0).
- **CRVAL1, CRVAL2** = image centre coordinates (degrees) – in Equatorial coordinates these values will be in Right Ascension and Declination (or their equivalents in other coordinate systems). They are derived from the first valid values of (in order)
  - The value of the “ra” and “dec” product metadata.
  - The value of the “raNominal” and “decNominal” product metadata.
  - The Ra/Dec of the first bolometer in the first source scan (with logging of this fact to the HIPE log)
- **CRPIX1, CRPIX2** = image centre nearest pixel coordinates (at spatial coordinates CRVAL1, CRVAL2).
  - These values are based on x-axis and y-axis pixel coordinates that run from 1 to NAXIS1 and 1 to NAXIS2 respectively with increments of 1 per pixel.
- **CROTA2** = image rotation about the image centre (pixel = (CRPIX1, CRPIX2)) (degrees).
- **CDELT1, CDELT2** = x axis and y-axis map pixel resolutions respectively (degrees per pixel) ((user/default resolution)/3600.).
  - NAXIS1, NAXIS2 the number of x-axis and y-axis images pixels respectively – these ranges are defined by the (ra,dec) ranges of the data samples divided by CDELT1/CDELT2 respectively.

Note that (x,y) as used here define a conventional coordinate system starting at bottom-left with a horizontal x-axis and vertical y-axis. The HIPE DISPLAY Task returns coordinates to the user as (y,x) in this scheme.

### 6.11.1.2. SPIRE Mapping tasks

The coordinate mappings are performed as the raw data are read into an internal data structure called a TodBuffer – so this discussion is only valid when the INPUT parameter to the mappers (for example naiveScanMapper) is derived from a Level1 product – or the related task `createTodBuffer` is used. The mapping have an optional ‘wcs’ parameter which can be used define a set of WCS values to control the behaviour in three ways and described below:

1. The **wcs** parameter is omitted
2. Selected FITS WCS keywords are supplied to change the coordinate system, projection or rotation of the map (new in HIPE Version 11)
3. A fully defined WCS is specified
Default – no user supplied WCS

The mapping tasks will internally calculate WCS values to create an equatorial coordinate map with its x-axis “Right Ascension aligned” and its y-axis “Declination aligned” (rotation angle \( \text{CROTA2}=0 \)). The “map resolution” (angular resolution per map-pixel) is set either from either the task parameter (“resolution”) or taken as a default value for the appropriate bolometer array (6.0 arcsec for “PSW” for example). The complete set of calculated WCS keyword values is stored in the map metadata and is available from the \text{map.wcs} HIPE value. As an example in HIPE we can make a map from a SpireListContext type Product. This map will use all defaults (Equatorial coordinates and aligned such that the x and y-axes are parallel to the RA and Dec coordinates respectively (\( \text{crota2}=0 \)).

```
HIPE> s1c=baselineRemovalMedian(obs.levell)
HIPE> map = naiveScanMapper(s1c)
HIPE> print map.wcs
```

### World Coordinate System
```
naxis : 2
naxis1 : 323
naxis2 : 347
crpix1 : 154.0
crpix2 : 168.0
crval1 : 185.73231411677688
crval2 : 15.822236057092217
cdelt1 : -0.0016666666666666668
cdelt2 : 0.0016666666666666668
ctype1 : RA---TAN
ctype2 : DEC--TAN
equinox : 2000.0
crota2 : 0.0
```

Maps in other coordinates, projections or rotations

A feature added from the HIPE 11.0 release makes it possible to create a map in either of the two other available celestial coordinate systems (galactic or ecliptic coordinates), to change the default spherical->planar map projection (from tangential projection = “TAN”) or to alter the image rotation within the map. To do this the user must supply a subset of WCS values to the mappers. These will be used in preference to the automatic values – any omitted values will still be calculated - and the complete WCS will be available as the value of “\text{map.wcs}” – HIPE – exactly as before. As a first example we could make our map in galactic coordinates.

```
HIPE> wcs = Wcs(ctype1='GLON-ARC',ctype2='GLAT-ARC')
HIPE> map = naiveScanMapper(s1c,wcs=wcs)
HIPE> print map.wcs
```

### World Coordinate System
```
naxis : 2
naxis1 : 307
naxis2 : 339
crpix1 : 147.0
crpix2 : 176.0
crval1 : 271.1496762062858
crval2 : 76.90048460066771
cdelt1 : -0.0016666666666666668
cdelt2 : 0.0016666666666666668
ctype1 : GLON-ARC
ctype2 : GLAT-ARC
equinox : 2000.0
crota2 : 0.0
```

(\(\text{crval1,crval2}\)) are now in Galactic coordinates. With the default rotation angle (\( \text{CROTA2}=0.0 \)) the resulting map will be aligned with its x-axis along the galactic plane and its y-axis perpendicular to this.

Or, alternatively, in ecliptic coordinates:
HIPE> wcs = Wcs(ctype1='ELON-ARC',ctype2='ELAT-ARC')
HIPE> map = naiveScanMapper(slc,wcs=wcs)
HIPE> print map.wcs
World Coordinate System
-----------------------
naxis : 2
naxis1 : 298
naxis2 : 335
crpix1 : 143.0
crpix2 : 172.0
crval1 : 178.78589527744876
crval2 : 16.760987984218204
cdelt1 : -0.0016666666666666668
cdelt2 : 0.0016666666666666668
ctype1 : ELON-ARC
ctype2 : ELAT-ARC
equinox : 2000.0
crota2 : 0.0

(crval1,crval2) are now in ecliptic coordinates and the default axes alignment will be similar to that for galactic coordinates.

As a final example we can alter the image rotation by specifying a value for the angle “crota2” – the parameter checking currently requires the WCS values “ctype1” and “ctype2” to be defined in any input WCS (even if not altered from the defaults).

HIPE> wcs = Wcs(ctype1='RA---TAN',ctype2='DEC--TAN',crota2=60.0)
HIPE> map = naiveScanMapper(slc,wcs=wcs)
HIPE> print map.wcs
World Coordinate System
-----------------------
naxis : 2
naxis1 : 340
naxis2 : 309
crpix1 : 164.0
crpix2 : 148.0
crval1 : 185.73231411677688
crval2 : 15.822236057092217
cdelt1 : -0.0016666666666666668
cdelt2 : 0.0016666666666666668
ctype1 : RA---TAN
ctype2 : DEC--TAN
equinox : 2000.0
crota2 : 60.0

Fully specified WCS

If the user specifies a value for the Wcs parameter to the mapper this can either be used define a subset of FITS keywords (as above) – with the mapper calculating missing values - OR it can define a complete WCS with all keywords specified (including values for the map size – NAXIS1 and NAXIS2). In this latter case (which would normally be a slightly modified version of a WCS already calculated by the mapper!) the “automatic size calculation” normally done by the mapper is turned off and the supplied keywords are used to simply project each data sample onto the 2-dimensional map which these keyword values represent. As it would be unrealistic to generate a complete WCS “by hand” such an example is not given. To use such a WCS the syntax is, obviously:

HIPE> map = naiveScanMapper(slc,wcs=wcs)
6.11.2. Merging two or more SPIRE Maps together

There will be instances when it is desirable to be able to merge SPIRE observations taken at different times into a single final map. Most commonly, this will be needed for SPIRE Parallel Mode observations which are not crosslinked (i.e. the 2 orthogonal scans are taken separately and must therefore be combined at a later time), although one could also imagine instances where we may wish to combine many maps to obtain a new deeper final image of our target(s). Such a utility exists within HIPE under the Scripts menu (See Chapter 3, Figure 3.1). Alternatively, the script resides within the HIPE Build directory structure, accessible through scripts/spire/ia/scripts/useful/Photometer_MapMerge.py.

6.11.2.1. SPIRE Map Merging

This map merging script will merge two or more observations performed in SPIRE Large Map, Small Map or Parallel Mode together to produce a single combined map. The main use for this script arises from the fact that it is not possible to have a single cross-scan Parallel Mode observation. Up to the Level 1 Product generation, SPIRE data is divided by, and processed as building blocks of individual scan lines. Individual scan lines will have different baseline subtraction and so it is desirable to combine all the Level 1 scan lines from all the required observations before the baseline subtraction and map making stages. The merging script will carry out the following tasks:

1. Load multiple observations (requires the Observation ID and Pool name for each observation)
2. Collect the the Level 1 products (scan lines) together from each observation
3. Apply bolometer relative gains to the timelines as a default to optimize the timelines for extended emission.
4. Perform baseline subtraction on Level 1 timelines using the destriper as the default
5. Perform Map Making on scan lines from all observations for each SPIRE array

The user must specify 3 inputs to the script (i.e. the script should be edited):

1. An array of Observation IDs
2. A corresponding array of Pool names (this list must be the same length as the Obs ID list above)
3. An output directory for writing the combined maps to FITS files

In addition, the user may set 3 Boolean flags to dictate how the script processes the observations

```python
# applyExtendedEmissionGains = True : Map optimized for extended emission
# applyExtendedEmissionGains = False : Map optimized for Point Sources
applyExtendedEmissionGains = True
useDestriper = True
useHsa = False
```

By default the baseline removal method is the Destriper however, simpler Median Baseline subtraction can be selected by setting useDestriper = False. By default the relative bolometer gains are applied to the timelines to optimize the maps for extended emission and aperture photometry. Note that if these maps are to have photometry performed on point sources using either the sussExtractor algorithm or the timeline Fitter then set applyExtendedEmissionGains = False. Finally, to access observations directly from the Herschel Science Archive (HSA) set useHsa = True

Below we show 2 examples: The first example shows how to combine two individual orthogonal scans from a parallel mode observation. It is assumed that the observations with observation IDs 1342183046, 1342183047 have already been downloaded from the archive and placed in the
Pools `spirephot_ngc6946_nom_rp` and `spirephot_ngc6946_orth_rp` respectively. To run the merging script we edit the following lines (where `outDir` is the path to a directory where you wish the final combined maps, as FITS files, to be saved):

```python
obsids = [1342183046, 1342183047]
pools = ['spirephot_ngc6946_nom_rp', 'spirephot_ngc6946_orth_rp']
outDir = '/Users/cpearson/jython/localstore/plots/
```

Running the script displays the input maps (from a chosen array, in this case the PLW array), and the final combined maps in all arrays as shown in Figure 6.94. The FITS files of the combined maps are saved to the path specified in the input parameter `outDir`.

In the second example (not taken from the HIPE script), 4 individual maps, each of four repetitions, of the calibration source Gamma Draconis, taken in Small Map Mode over approximately 150 days are combined to produce a single map. The input to the map merging script is shown below. In this example we have two of the observations (1342212361, 1342211593) in separate pools but the final two observations are contained within the same pool (i.e. observations 1342207048, 1342210562 are both contained with a single pool called `GammaDra1342207048and10562`). Note that this final Pool must be included twice in the input list:

```python
obsids = [1342212361, 1342211593, 1342207048, 1342210562]
pools = ['GammaDra1342212361', 'GammaDra1342211593', 'GammaDra1342207048and10562', 'GammaDra1342207048and10562']
outDir = '/Users/cpearson/jython/localstore/plots/
```

Running the script displays the input maps (from a chosen array, in this case the PSW array), and the final combined maps in all arrays as shown in Figure 6.95.
6.11.3. Aligning SPIRE Maps

Current SPIRE maps are usually accurate to around 2 arcsec, however there may be instances where some maps may have slightly higher offsets or occasions where more precise astrometry is required. For these scenarios, a script is provided within HIPE to align SPIRE observations and to improve their astrometry. The script Photometer_AstrometryCorrection.py, can be accessed from Scripts menu in HIPE in a similar manner as shown in Figure 6.72. Note that the script not only aligns the astrometry in the maps but also changes the astrometry in the scan lines.

6.11.3.1. Aligning two SPIRE maps

Two SPIRE observations can be aligned with each other where the first is the reference image obs_ref and the second is the map to be aligned obs1. The input in this case is a pair of observation contexts as shown in Example 1 in the Photometer Astrometry Correction script. The astrometry in the observation context obs1 is then updated.

\[
\text{obs1} = \text{astrometryFix(data=obs1, reference=obs_ref)}
\]

6.11.3.2. Aligning Multiple SPIRE maps

Multiple SPIRE maps can be aligned by defining a list of observations as shown in Example 2 in the Photometer Astrometry Correction script. The astrometry of the observations in the list will be corrected relative to the reference SPIRE observation.

6.11.3.3. Aligning a SPIRE observation to an ancillary reference image

A SPIRE observation can be aligned to an ancillary image (in Example 3 in the Photometer Astrometry Correction script we use a radio image from NVSS).

6.11.3.4. Aligning a SPIRE observation using a source list

A SPIRE observation can also be aligned using a source list containing positions of known sources. This is shown in Example 4 in the Photometer Astrometry Correction script. Note that the source list must be in the format of a SourceListProduct. The sourceList is produced by using DAOPHOT or Sussextractor tasks. The two tasks are listed in the Applicable folder of the Tasks view whenever an image is selected in the Variables view. In this case the call to the task will be;

\[
\text{obs} = \text{astrometryFix(data=obs, reference=sourceList)}
\]

6.11.4. Creating Superresolution Maps for SPIRE Observations

With good knowledge of the Herschel-SPIRE beam profile, it is possible to deconvolve the data in cases of high signal-to-noise data to produce high/super resolution maps. For SPIRE observations, HiRes techniques typically increases the resolution of the image by a factor of 2, and is best suited to structure in extended structure with high signal-to-noise (see the SPIRE Handbook for details).

There is a good correlation between the flux density and the signal-to-noise in a SPIRE map. For HiRes processing, the primary consideration is the signal relative to the extragalactic background. The
selection is made based on the absolute calibrated nominal maps in surface brightness units. The maps that are selected to undergo HiRes processing have two criteria applied (see Table 6.17):

- Criterion 1: A sufficient number of pixels (i.e. an area of sky) must be above a threshold surface brightness.
- Criterion 2: The 99th percentile of the surface brightness must be above a threshold value.

### Table 6.17. Criteria used for selecting SPIRE observations for HiRes processing.

<table>
<thead>
<tr>
<th>Criteria</th>
<th>PSW</th>
<th>PMW</th>
<th>PLW</th>
</tr>
</thead>
<tbody>
<tr>
<td>Criterion 1 (no. of pixels)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Surface brightness threshold (MJy/sr)</td>
<td>100</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>Number of pixels above threshold</td>
<td>544</td>
<td>196</td>
<td>100</td>
</tr>
<tr>
<td>Equivalent Sky Area (sq.arcmin)</td>
<td>5.44</td>
<td>5.44</td>
<td>5.44</td>
</tr>
<tr>
<td>Criterion 2 (99th percentile threshold)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>99th percentile threshold (MJy/sr)</td>
<td>30</td>
<td>15</td>
<td>10</td>
</tr>
</tbody>
</table>

A script to produce Superresolution maps (based on the HiRes mapping algorithm) from SPIRE data is available within HIPE as [Photometer Super Resolution Mapping](Useful Scripts) (See Chapter 3, Figure 3.1). Alternatively, the script resides within the HIPE Build directory structure, accessible through `scripts/spire/ia/scripts/useful/Photometer_HiResMapping.py`. The script allows Users to set up beam profiles and to run the HiRes superresolution mapper (HiResMapperTask, or hiresMapper) on SPIRE Level 1 data. (Note that an alternative superresolution method is provided by the SUPREME plugin.)

### 6.11.4.1. Preparing to run the HiresMapper task

The first part of the user script shows the important user-selectable options. These include the observation ids to process, the size of the HiRes image in arcminutes, the center of the image in decimal Right Ascension and Declination, and the SPIRE band to process.

```python
# # User Selectable Options  #
# (A) List of OBSIDs in the form of an integer or hexadecimal (0x) number:
#   (Example data for M33)
# (B) band to process, e.g. 'PSW', 'PMW' or 'PLW'
# The example obsids are for M33
obsids = [1342189079, 1342189080]
band = 'PLW'  # PSW, PLW, or PMW
```

The next section of the script shows how to prepare the beam profiles. The following code retrieves the current beam profiles from the SPIRE Calibration Tree, and crops them to a size suitable for HiRes. The HiRes images in the archive use the SPIRE beam as measured on Neptune with 1 arcsecond resolution. The same beam is assumed for all detectors and all observation, regardless of orientation. While the SPIRE beam is not completely azimuthally-symmetric, the variations caused by a beam rotation, are very small.

```python
# Get the necessary beams and other calibration files
cal = spireCal()
fullBeam = cal.getPhot().getProduct('BeamProfList').getProduct(array,'fine')
chanRelGains = cal.phot.chanRelGain
# set the beam size (recommended sizes)
```
Since the HiRes task is best suited to extended (or partially-extended) emission, the detector relative gains need to be applied to the detector timelines before destriping. The following code snippet retrieves the user-specified observations from the archive, applies the relative gains and runs the destriper.

```python
# Create Level 1 timelines
level1Corrected = Level1Context()
# Retrieve timeline data
for obsid in obsids:
    print "Retrieving observation %i"%(obsid)
    obs = getObservation(obsid, useHsa=True, instrument='SPIRE')
    # Apply relative gain correction for beam variation over detectors
    print "Applying Relative gains for %i"%(obsid)
    for i in range(obs.level1.getCount()):
        psp = obs.level1.getProduct(i)
        psp.meta['type'].string = "PSP"
        psp = applyRelativeGains(psp, chanRelGains)
        level1Corrected.addProduct(psp)
    print
print "Start Destriping"
# (Re-)run destriper on new Level1Context
level1Corrected.map, diag, p4, p5 = destriper(level1=level1Corrected, 
array=array, nThreads=2, withMedianCorrected=True, useSink=False)
print "End Destriping"
```

The `hiresMapper` task requires a World Coordinate System (WCS) object to be provided as input. Since HiRes roughly doubles the image resolution, the resulting FITS files have identical World Coordinate System (WCS) to the nominal FITS files, but with half the pixel size (3 arcseconds for PSW (250 microns), 5 arcseconds for PMW (350 microns), and 7 arcseconds for PLW (500 microns). The following code copies the WCS object from the Level-2 map and modifies it according to the user inputs.

```python
# Prepare Wcs with half the pixel size of standard map
wcs = obs.level2.getProduct('psrc'+array).wcs.copy()
wcs.cdelt1 /= 2.0
wcs.cdelt2 /= 2.0
wcs.naxis1 *= 2
wcs.naxis2 *= 2
wcs.crpix1 *=2
wcs.crpix2 *=2
```

For the deconvolution to work well, the data should include as few negative values as possible. The timeline data therefore has the absolute offsets (zero point correction) from the Planck maps converted from surface brightness (MJy/sr) to flux density (Jy/beam) by dividing by the beam solid angle (Once HiRes has been run, the offset is removed and the conversion from flux density (Jy/beam) to surface brightness (MJy/sr) applied as for nominal maps).

```python
# Get zeroPoint offset in Jy/beam to use to ensure no negative values
zOffsetExtd = obs.level2.getProduct('extd'+array.upper()).meta['zPointOffset'].value
beamAreaPipSr = cal.phot.colorCorrBeam.meta["beamPipeline%Sr"array.title()].value
zOffsetPsrc = zOffsetExtd * beamAreaPipSr * 1e6
```

With the beams, the Level-1 data, and the WCS object all properly prepared, the `hiresMapper` is ready to be deployed.
6.11.4.2. Running the HiresMapper task to produce SPIRE super-resolution maps

The next part of the superresolution script shows the default way to run the hiresMapper task. The HiRes task is an iterative procedure, but excessive iterations can over-process the data. A good compromise is to run HiRes for 20 iterations.

```python
# (1) 20 iterations (default for maxIter parameter), output final iteration only
hiresImage, hiresBeam = hiresMapper(level1Corrected, array=array,
    beam=beam, wcs=wcs, fluxOffset = zOffsetPsrc)
```

It is possible to run the task for only one iteration. The first HiRes iteration is simply a coaddition.

```python
# (2) Output first iteration only
hiresImageIter1, hiresBeamIter1 = hiresMapper(level1Corrected,
    array=array, beam=beam, wcs=wcs, fluxOffset = zOffsetPsrc, maxIter=1)
hiresImage = hiresImageIter1
```

These outputs are depicted in Figure 6.96. At upper left is the first iteration of HiRes (the coadd). This coadd has a slightly lower resolution than the pipeline default, but it has the advantage of interpolating the data onto smaller pixels. The 20th iteration image is shown in upper right. The lower left is the "beams" image for the 20th iteration, produced by placing spikes on a grid and running the HiRes iterations. The lower right is the cfv or Correction Factor Variance image, which is one of the planes in each hiresImage. Where the cfv is low, the correction factors for the input data samples are in good agreement. The cfv image in this example contains two black spots of high variance, which indicates disagreement between the input data samples, as could be caused by a residual glitch or other artifact in the input data.

Note that the Hires Mapper task does not restore all the original NaNs in the final hires images. However, it should be straightforward for the user to identify these areas, using the coverage map. The user can safely assume that regions of coverage less than 10^-10 (effectively zero coverage) are where the signal image should be NaN.
The `hiresMapper` task can also output multiple iterations as shown in the next code block.

```python
print "Hi Res Mapping (3)"
# (3) 20 iterations, output multiple iterations
hiresImages, hiresBeams = hiresMapper(level1Corrected, array=array,
    beam=beam, wcs=wcs, fluxOffset = zOffsetPsrc,
    storeIter=[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,20])
hiresImage1 = hiresImages[1]
hiresBeam1 = hiresBeams[1]
hiresImage20 = hiresImages[20]
hiresBeam20 = hiresBeams[20]
hiresImage = hiresImage20
```

The final part of the script shows an example of running a few iterations, modifying one of the inputs, and restarting the HiRes iterations with a corrected startImage. Note that it is possible to begin the `hiresMapper` task with a starting image -- by default the starting image is simply a flat, constant image.

```python
# (4) Stop after N-iterations, adjust the image, then continue
n = 2
hiresImageIterN, hiresBeamIterN = hiresMapper(level1Corrected,\
    array=array, beam=beam, wcs=wcs, fluxOffset = zOffsetPsrc, maxIter=n)
```
Some post processing of the HiRes images is required and is detailed in the last part of the script below. The regions with zero coverage or with spuriously high values can be replaced with NaNs using the code below (HiRes maps with flux density greater than 5 times the maximum value in the nominal map are set to NaN value.).

Super resolution maps calibrated for extended emission in MJy/sr can also be derived from the nominal HiRes maps by dividing by the bema and reapplying the zero point offset.

The defaults for the hiresMapper task include fluxOffset = 0 and minSampleFlux = 1.e-200. The HiRes algorithm can only work on data samples with positive flux values. Any input samples below minSampleFlux will be set to the value of minSampleFlux. The fluxOffset parameter may be used to add a constant value to all the input values to ensure positivity. It is necessary to keep the data as close to zero as possible to achieve superresolution. However the truncation to minSampleFlux can skew the noise properties. To attempt to recover a realistic noise distribution, a fluxOffset can be added and hiresMapper run for a few extra iterations.

6.12. Tools for Moving Objects

6.12.1. Correcting observations of Solar System Objects

When the target of a SPIRE observation is a Solar System Object (SSO) having a proper motion across the sky, the spacecraft re-adjusts its position after each scan, in order to always be centred on the target as shown in Figure 6.97. SSO products within the Herschel Science Archive also contain Level 2 products labelled ssoPxW that are maps corrected for the target's proper motion. Observations reduced with earlier versions of HIPE may include products only reduced using a standard pipeline which does not correct for the target's proper motion. As a result, the background of SSO observations will be well guided while the SSO itself will appear blurred (for short observations or slow objects) or as a streak (for longer ones or faster objects, see Figure 6.98).
The Solar System Object Motion Correction script (Photometer_SSOMotionCorrection.py) can be accessed from the Scripts menu in HIPE in a similar manner as shown in Figure 6.72. The script performs the following: load the SSO observation (which requires the obsID and Pool name); correct the Level 1 products (scan lines) for the differential velocity of the object, using the ephemeris and horizons file (which are stored within the Observation Context); perform baseline subtraction on level 1 timelines using a median subtraction if necessary; create new maps in the moving object frame; display original PSW map, the new corrected Level 2.5 map (and the difference map) and finally save the corrected observation back to pool (or optionally a new pool specified by user).

Note that Destriping using 2nd Level Deglitching should not be performed on observations of SSOs, before the SSO motion correction has been applied, as this will result in misidentifying source samples as glitches.

The script computes the SSO speed in RA and Dec coordinates, applies the required shift to Level 1 timelines, computes the corrected maps and eventually saves the modified products to a local pool. The results are maps centred on the SSO (i.e. the target will appear focused) with a smooth/trailed background as shown in Figure 6.98. The script takes as input 3 parameters. The observation ID and the name of the Data Pool on disk of your SSO observation and a parameter newPool which by default is set to the input Pool name (i.e. it will be saved back to the same pool). Alternatively, this parameter may be set to a new pool name if you want to save the observation back to a new separate Pool.

```
obsid    = 1342201382
dataPool = "mySSOobservation"
newPool  = dataPool
```

Figure 6.97. Telescope tracking an SSO during a mapping observation.
6.12.1.1. Problems with early processed observations

The Solar System Object Motion Correction relies on the objects NAIF ID being present in the meta data of the observation and the Level 1 context. SSO observations processed with early versions of HIPE may not have the NAIF ID (or the AOT parameter also required by the task) in their meta data and may cause the Solar System Object Motion Correction script to fail. The recommendation is to download the observation again from the Herschel Science Archive since this will have been processed by latest pipeline version without problem.

Note that it is possible to manually add meta data to the observation if necessary. This can either be done with the following lines of code or by selecting the Level 1 context in the HIPE viewer, opening the meta data tab and selecting the Add Parameter option as shown in Figure 6.99. The new meta data for the appropriate NAIF ID and AOT can then be added as shown in the figure.

```python
naifd="899"
aot="Photometer"
obs.level1.meta['naifId']=StringParameter(value= naifd, 
    description='SSO NAIF identifier')
obs.level1.meta['aot']=StringParameter(value= aot, 
    description='AOT Identifier')
```
6.12.1.2. Caveat on WCS for SSO centred Observations

Note that the HIFI instrument have found some cases where the WCS encoded in the FITS headers of HIFI SSO cubes is incorrect. No such problem has been seen in SPIRE SSO data however, users are advised to check their WCS, making sure not to use CTYPE = 'RA-xxx' or 'DEC-xxx' if it is not appropriate.

6.12.2. Locating a Solar System Object in a map

When the target of a SPIRE observation is a Solar System Object (SSO) having a proper motion across the sky, it can be useful to locate the start and end positions on the map since the spacecraft re-adjusts its position after each scan, in order to always be centred on the target as shown in Figure 6.97. Moreover, in HCSS versions prior to 9.0, the parameters raNominal and decNominal in the observation meta data did not contain appropriate values for the observation. Note that in version HIPE 9.0 and later the raNominal and decNominal meta data will contain the start position of the SSO.

A useful script is provided to assist with faint moving objects. The Calculate Ephemeris SSO Position (Photometer_getSSOposition.py) script can be accessed from the Scripts menu in HIPE in a similar manner as shown in Figure 6.72.

The script takes a photometer map and then uses the the Horizons and Ephemeris products contained within the observation Context, together with the SSO naifID (which distinguishes individual
moving objects) to calculate the position as a function of time of the object. The script also places a circle on a display of the map at the beginning and end point of the observation (see Figure 6.100).

Figure 6.100. Marking the start and end position of an SSO on a map.
Chapter 7. SPIRE Spectroscopy Mode Cookbook

7.1. Introduction to processing FTS data

7.1.1. Basics of Fourier transform spectroscopy

Although more complex, the design of an imaging Fourier Transform Spectrometer (iFTS) builds on the optical design of a standard FTS, which is relatively simple. An incoming beam of light is divided using a beamsplitter, and the two component beams are reflected and recombined to form an interference pattern. One of the beams reflects from a moving mirror, which introduces a necessary optical path difference (OPD). A linear translation stage is used to modulate the OPD between the two arms of the spectrometer. The interference pattern (fringes) formed at the focus is called the interferogram.

The spectral resolution of an FTS depends on the maximum possible OPD, where the larger the maximum OPD of an FTS system the better its spectral resolution. Mechanically, the most difficult aspect of an FTS is usually the linear translation stage. The longest throw of the linear stage sets physical limits for the best resolution possible. The resolution element of an FTS ($\Delta \sigma$), i.e. the width of one spectral bin where one independent data sample has been collected, is given by $\Delta \sigma = 1 / (2 \ OPD_{\max})$.

The SPIRE interferometer is of the Mach-Zehnder variety, following a design from [Ade et al. 1999] (see Figure 7.1). Powered mirrors fold the two beams towards the line of symmetry so that the optical path difference is modulated by a single linear translation stage mechanism (SMEC). The stage mechanism carries two back-to-back rooftop mirrors. A displacement of $\Delta x$ of the mechanism shortens one beam path by $2 \Delta x$ and lengthens the other beam path by $2 \Delta x$, leading to an overall modulation of OPD by $4 \Delta x$.

The maximum OPD modulation of the SPIRE FTS is $4 \times 3.2 \ cm = 12.8 \ cm$ and the resulting resolution element is $0.04 \ cm^{-1}$ or $1.2 \ GHz$. The best achievable resolution of an FTS is $1.20671$ times the resolution element, which gives a full width at half maximum (FWHM) for SPIRE’s high resolution mode of $0.0483 \ cm^{-1}$ or $1.447 \ GHz$.

The SPIRE FTS employs an optical encoder as the metrology system to determine relative changes of the position of the stage to an accuracy of $1 \ \mu m$ through the whole range of the stage translation.
with an interpolation algorithm providing spatial information to a precision of ~10 nm. The optical encoder enables the speed of the linear translation stage to be controlled with high precision using a servo control system.

The SPIRE FTS has two input ports and two output ports. One input port is illuminated by the telescope (with a temperature range of 80 - 90 K) and the astronomical source. The other input port is illuminated by the Spectrometer Calibration source, however this is not powered and its temperature varies within a range of 4 - 5 K, with the instrument itself. At each output port there is a bolometer array - the SPIRE Long and Short Wavelength arrays (SLW and SSW).

Zero path difference (ZPD) is a unique point of symmetry of the FTS system, where the path lengths of the two beams are identical and radiation of all wavelengths interferes constructively, leading to a bright central maxima of the interference pattern. At ZPD the total power difference between both input ports is directed towards one output port, see Figure 7.2 for an example. The difference between the peak at ZPD and the interferogram baseline represents half of the total power difference between the two input ports.

![Figure 7.2. The centre burst of a sample interferogram I(x).](image)

For a comprehensive introduction to Fourier transform spectrometry see [Davis et al. 2001], while additional details on the design of the SPIRE FTS can be found in section 2.3 of the SPIRE Handbook.

### 7.1.2. Detection

As the SMEC scans, OPD changes and an interference pattern of bright and dark fringes is formed on the focal plane. This pattern is recorded by the detectors in the two SPIRE FTS arrays at a fixed read-out frequency of about 80 Hz. The nominal speed of the SMEC is 2 mm/s in OPD, leading to a nominal sampling interval (ΔOPD) of v/f = 25 μm. This OPD interval determines the highest frequency (commonly called the Nyquist frequency, σNy) that can be measured by the FTS, of σNy = 1 / (2 ΔOPD). The final spectrum contains information at all frequencies from zero up to the Nyquist frequency at equidistant intervals. In the case of SPIRE, the Nyquist frequency is σNy = 200 cm⁻¹ or ~6000 GHz. However, the scientifically useful information is restricted by the "optical passband" to which the detectors, filters, feedhorns, waveguides, etc. contribute. The optical passbands for the two arrays are 447-1018 GHz (671-294 μm) for SLW and 944-1568 GHz (318-191 μm) for SSW.
With the highest (scientifically meaningful) frequency at 1568 GHz (52.3 cm\(^{-1}\)) and a Nyquist frequency of 6000 GHz, the measured data are almost four times oversampled under nominal operating conditions.

The SPIRE FTS and Photometer use bolometric detectors of the same design. These detectors must have sufficient dynamic range to record the entire interferogram from minimum to maximum signal. The dynamic range is set by the Analogue-to-Digital Converters in the detector electronics, which limit the signal range that can be measured. Any signal outside of the range of SPIRE’s 16-bit Analogue-to-Digital Converters will be “clipped”. Note that the SPIRE FTS automatically subtracts a constant offset level (see the description of the SPIRE Analogue Signal Chain). Detector offsets are measured once, at the beginning of each observation, with the exception of jiggle observations in the bright source mode, where they are measured once per jiggle position to account for source contrast.

The SPIRE FTS detectors only respond linearly to incoming radiation within a limited signal range. In particular, the central burst of the interferogram often leads to a non-linear detector response, which must be corrected when processing the data. A non-linearity correction, which is derived from a model of the bolometers and their measured performance, is applied during data processing.

### 7.1.3. Fourier Transformation

Interferograms bear little intuitive resemblance to the measured spectrum, but they do contain all necessary information as the Fourier components.

A Fourier transform (FT) converts the interferogram \(I(x)\) into a spectrum \(B(\sigma)\), and is defined as:

\[
B(\sigma) = FT(I(x)) = \frac{1}{N} \sum_{x=-\infty}^{\infty} I(x) \exp(-i2\pi \sigma x)
\]

\[
I(x) = FT^{-1}(B(\sigma)) = \sum_{\sigma=-\infty}^{\infty} B(\sigma) \exp(i2\pi \sigma x)
\]

The FT is the core step for any processing of FTS data. The computational complexity of the discrete FT is of the order of \(O(n^2)\), because for each sample point, a sum across all other sample points has to be calculated. In the 1960’s, a Fast FT (FFT) algorithm was developed (see [Cooley Tukey 1965]), which reduces the computational complexity to \(O(n \log(n))\). Further performance improvements have been integrated into state-of-the-art FFT code. HIPE contains a Java implementation of FFTPACK, see [Swarztrauber 1982]. Standard FFT packages make the assumption that Fourier samples are available at equidistant intervals and the FT puts data onto an equidistant, spectral axis. The spectral resolution element is defined by the maximum OPD of the interferogram, see Section 7.1.1.

### 7.1.4. Efficiency Losses

Any FTS system suffers from an imperfect overlap of the two recombining light beams, which leads to a loss in efficiency. If this loss is independent of OPD, it results in a sudden decrease in the modulation efficiency. In practice, the efficiency loss often worsens with increasing OPD and leads to a gradual information loss in the interferogram. Known as 'natural apodization', this effect reduces the spectral resolution but also reduces the side-lobes associated with line features, hence the name apodization, which is Greek for 'removing feet’.

It is also possible to deliberately introduce an efficiency loss by multiplying the interferogram with an apodizing function, which trades spectral resolution for a reduction of the sidelobes of the instrumental line shape. There is an optimal boundary of how much spectral resolution has to be sacrificed in order to reduce side-lobes by a given amount. Optimal apodization functions approaching this boundary have been derived by [Naylor Tahic 2007], and are available within HIPE (see Section 7.3.28).
Detectors towards the edge of the array, i.e. away from the optical axis, receive decreasing amounts of power with increasing OPD, due to vignetting by the instrument optics. This gradual loss in power is highly reproducible and can be well fitted, for example with a 4th order polynomial function. Only low-order spectral components, well below the optical passbands of the SPIRE detectors, are required to encode this feature in the interferogram. Similarly, 1/f-like noise on the baseline of interferograms (see Figure 7.3) does not affect scientifically meaningful data as long as it remains below the optical passbands of the detectors. Measurements of very bright point sources may show a significant amount of 1/f-like noise due to limited pointing stability.

![Figure 7.3. 1/f-like variation of the baseline of eight repeated scans during the observation of a very strong point source.](image)

### 7.1.5. Interferogram Asymmetries - Spectral Phase

The power measured at the focus of an FTS should depend strictly on the difference between the optical path lengths of the two beams of the interferometer, i.e. the interferogram should be perfectly symmetric with respect to ZPD, which is the unique point where the path lengths of the two beams are identical and constructive interference occurs for radiation at all frequencies. SPIRE's interferograms are not perfectly symmetric for several reasons, which include random noise, discrete sampling, dispersive elements within the interferometer, non-linear detector response, read-out electronics, and thermal inertia of the detectors. Data must be available on either side of ZPD in order to identify and characterise asymmetry in the interferogram. An interferogram that records data on either side of ZPD is called double-sided. In order to obtain the best spectral resolution, i.e. in order to maximize OPD\(_{\text{max}}\), FTS systems are designed to record only a short section of double-sided interferogram, while most of the OPD modulation extends one of the sides. An interferogram that records data on mostly one side of ZPD, is called single-sided. SPIRE records double-sided interferograms when making observations in low and medium resolution and a single-sided interferogram in high resolution, as illustrated by Figure 7.4.
Figure 7.4. Sample interferograms from SPIRE taken in High (black) and Low (red) resolution mode.

Figure 7.5. The imaginary part of five double-sided spectra (left) and the corresponding phase (right).

The asymmetry in a double-sided interferogram is commonly characterised by its phase:

\[
\tan(\Phi(\sigma)) = \frac{\text{Imaginary (FT(I(x)))}}{\text{Real (FT(I(x)))}}
\]

Examples of the imaginary parts of double-sided spectra and the corresponding phase are shown in Figure 7.5.

Asymmetries in the interferogram derive from various sources:

- Photon noise, stage jitter, electrical noise all contribute to noisy phase.
- The maximum signal may not occur precisely at ZPD. Any missampling of ZPD by \(\Delta x\) leads to an asymmetrical interferogram and a linear phase: \(\Phi(\sigma)_{\text{ZPD}} = 2\pi \Delta x \sigma\).
The optical path length may differ for different frequencies as the refractive index of dielectric media depends on frequency. For SPIRE, the only transmission through dielectrics within the interferometer occurs at the beamsplitters which are known to be the source of non-linear phase.

Non-linear detector response for varying flux levels can cause phase features that depend on source brightness and cannot be fully accounted for even by performing an in-depth instrument characterisation.

The read-out electronics which convert the analog detector signal into digital, computer-readable format introduce a frequency-dependent phase, which will generally be non-linear. The electrical phase can be calculated from the characteristics of the read-out electronics.

The detection of electromagnetic radiation is not instantaneous and suffers a time-lag (equivalent to a linear phase) that can be well modelled as a simple RC filter, which introduces a non-linear phase.

The last two of these effects, can be characterized to sufficient accuracy and accounted for in the time domain, see Section 7.3.12. The remaining phase can be characterised in a double-sided interferogram and subsequently be removed from the data. If the phase is linear, then the symmetry of the continuous interferogram is preserved, however the point of symmetry, ZPD, is shifted.

### 7.2. SPIRE Spectroscopy Data Structure

#### 7.2.1. Introduction to FTS data

This section details SPIRE FTS data products, how these are structured within HIPE, and how to inspect them.

For any FTS observation, the form of the final data products in the Herschel Science Archive (HSA) depends on the combination of observing modes used, i.e. the Astronomical Observation Template (AOT). Figure 7.6 shows the FTS observing mode options that were available during the mission. Observers had a choice of spatial sampling. Sparse sampling, where no jiggling was performed, and for higher spatial resolution, Intermediate or Fully sampled mapping (with 4 or 16 jiggle positions). For each sampling mode, pointing modes of Point (single pointing) or Raster (multiple pointings in a hexagonal pattern) were available, however Point was used for most science and calibration observations. For all observing modes, the spectral resolution was set as high resolution (HR) or low resolution (LR), or high plus low resolution (H+LR), and the detector bias as nominal or bright. The FTS could also observe in medium resolution (MR), but as MR was never used for science observations during the mission, MR observations taken for calibration purposes are now converted to LR by the pipelines. Full details on FTS spatial sampling, pointing modes and spectral resolution can be found in the SPIRE Handbook.

![Figure 7.6. SPIRE FTS observing modes. Six observing mode combinations were available, which depended on the spatial sampling and pointing mode selected. The name of the associated pipeline script used to reduce the observed data is indicated in the third row. Note that the pipeline names are for the SPG scripts. For the User pipeline scripts available in HIPE, the mapping version of the script must be used for sparse raster observations.](image)

An Observation Context is a HIPE object that holds all data and calibration products associated with a processed Herschel observation. Each Observation Context contains a number of levels that are filled at key stages of data processing, with the final products stored in the Level-2 Context. The form of the final FTS pipeline products is dictated by the AOT of the observation (see Figure 7.6 for the name of the pipeline script used to reduce data for each AOT). For sparse point observations, the Level-2 products are composite datasets for detectors from both arrays. One set of products contains
extended-source calibrated spectra and a second set contains the corresponding point-source calibrated spectra, for those detectors for which there is point-source calibration available. Each spectrum in the Level-2 products is the average over all scans for the respective detector. For intermediate and fully sampled mapping observations (and sparse raster), there are a set of Level-2 products per detector array, which are made up of the individual spectra collated into a 2D list (a Spectrum2d) and the spectral cubes these data are mapped into. Both types of Level-2 mapping products are calibrated using the extended source calibration. For observations taken in H+LR, there is a set of HR and a set of LR products, regardless of observing mode. And for all Level-2 spectra and spectral cubes, an apodized version is also produced.

Two types of calibration have been mentioned in the description of the final FTS data products above. Extended calibration, which assumes uniformly extended emission within the FTS beam and point-source calibration, which assumes point-source emission. The extended calibration is based on a model of the Herschel telescope emission-spectrum. As the telescope emission completely fills the beam in a uniform way, it provides appropriate calibration for a smooth uniformly extended source. After the extended-source calibration has been applied, the resulting data are in surface brightness units of W/m$^2$/Hz/sr, so a measure of the beam area is needed to convert into in-beam flux density. The point-source calibration applies a frequency dependent point-source conversion factor that was empirically derived using a comparison of the extended calibrated spectrum of Uranus to the standard Herschel Uranus model. This correction produces a spectrum calibrated as an unresolved point source with flux density units of Jy. As there is no point-source calibration available for the outer ring of partially-vignetted detectors, these are not included in the point-source calibrated products. Both FTS calibration schemes are detailed in Swinyard et al. (2014).

The FTS Observation Context structure was simplified in HIPE version 13 by moving the extended calibrated spectra into Level-2. Prior to this, the extended calibrated spectra were contained in the Level-1 Context, where the individual spectra were stored separately for each jiggle/raster position. Since HIPE version 13, the Level-1 Context contains only the raw interferograms for each scan. The names of FTS products was also changed from HIPE version 13, to be more instantly understandable. The names of FTS Level-2 products are detailed in Section 7.2.3. In HIPE version 14, a second set of spectral cubes was added to the Level-2 products for mapping observations. Both sets of cubes are mapped from the same pre-processed cubes (the Level-2 Spectrum2ds) onto matching rectangular, equidistant grids, but whereas the cubes included prior to HIPE 14 are mapped using the naive projection algorithm (as details in Section 7.8.5), the additional cubes are created with the convolution projection algorithm using a Gaussian kernel. This algorithm combines the spectra by taking the weighted mean of all spectra within a specified radius, thus providing a spatial smoothing that improves the coverage of the resulting cubes, so there are no "holes" as seen in the naive cubes. The convolution projected cubes (CP cubes) are discussed in more detail later in this section and in Section 7.8.5, where a comparison with the naive cubes is made.

The products for each observing mode are summarised in Table 7.1 and Table 7.2.

<table>
<thead>
<tr>
<th>Pipeline</th>
<th>Sampling</th>
<th>Pointing</th>
<th>Product</th>
<th>Units</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>All</td>
<td>All</td>
<td>Interferogram (before processing) for every detector</td>
<td>V interferogram</td>
<td></td>
</tr>
</tbody>
</table>

Table 7.2. Summary of FTS Level-2 pipeline products. Product names all start with the spectral resolution of the data. For mapping observations the array name is included next. The type of product is then included for all names. Finally, for sparse point observations, extended or point-source calibration is indicated. The pipeline used to produce the product is given as SOF1 and SOF2. These are the SPG scripts, with SOF1 given for sparse point and sparse raster observing modes. If reducing the data with the HIPE User pipeline scripts, sparse raster observations must be reduced with the mapping pipeline.

<table>
<thead>
<tr>
<th>Pipeline</th>
<th>Sampling</th>
<th>Pointing</th>
<th>Product</th>
<th>Units</th>
<th>Example name</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOF1</td>
<td>Sparse</td>
<td>Point</td>
<td>Extended calibrated spectra</td>
<td>W/m$^2$/Hz/sr</td>
<td>HR_spectrum_ext</td>
</tr>
</tbody>
</table>
7.2.2. The SPIRE FTS Observation Context

Data downloaded from the Herschel Science Archive (HSA) is provided in an Observation Context. The example data used in this section are from a 16-repetition HR observation (observation ID 1342227790) of the nominated SPIRE region of dark sky. If the observation has been saved to a data pool called "1342227790", the Observation Context can then be loaded into HIPE using:

```python
myObsid = 1342227790
myDataPool = "1342227790"
obs = getObservation(myObsid, poolName=myDataPool)
```

Alternatively, the Observation Context can be downloaded directly from the HSA using:

```python
obs = getObservation(myObsid, useHsa=True)
```

All data in the HSA is processed with HIPE version 14, but for data processed with versions prior to HIPE 13 (and therefore stored in the old Observation Context layout) there is a dedicated task (updateObsContext) provided in HIPE to update the format. This task moves the extended calibrated spectra from the Level-1 Context to Level-2 and updates the product names. For mapping observations, the Spectrum2d pre-processed cube product is created, which may take some time. However, the CP cubes are not generated, so only the renamed naive cubes will be present in Level-2 after the update is complete. This task will only convert Observation Contexts from HIPE version 12, as data from earlier versions are missing essential meta data. It is highly recommended to use FTS data that has been processed with HIPE version 14, as these data benefit from being calibrated with the last version of the FTS calibration to be released.

The Observation Viewer provides a summary view of the Observation Context in HIPE. Double clicking on `obs` in the HIPE Variables panel will automatically open the Observation Context with the Observation Viewer. This result can also be achieved by right-clicking on `obs` and then selecting Open With > Observation Viewer. In order to expand the view of one of the Context levels, click on the "+" symbol to the left of its folder icon.

7.2.3. The Final FTS Spectral Data Products (Level-2)

As explained in Section 7.2.1 and illustrated by Figure 7.7, the final FTS spectral data products are contained in Level-2 of the Observation Context. For sparse point observations, the extended calibrated spectra are stored in Spectrometer Detector Spectrum (SDS) products, and the point-source calibrated data in Spectrometer Point Source Spectrum (SPSS) products. For mapping data, the individ-
ual spectra are stored in pre-processed cubes (Spectrum2d) and in gridded spectral cubes (SpectralSimpleCube). In HIPE, all products within a Context level will be listed in alphabetical order.

Figure 7.7. Products in the Level-2 Context for a HR sparse observation (left), a HR mapping observation (middle) and a H+LR sparse observation (right).

For sparse data there are four entries inside the Level-2 Context. For example, the left panel of Figure 7.7 shows the Level-2 entries for an example HR observation. Two products provide the standard and apodized extended calibrated spectra and two the standard and apodized point-source calibrated spectra, in the following order:

- HR_spectrum_ext
- HR_spectrum_ext_apod
- HR_spectrum_point
- HR_spectrum_point_apod

The example Observation Context shown in the middle of Figure 7.7 shows the entries for a high resolution mapping observation. There are twelve entries in total, with six for each detector array. For each pre-processed cube and cube product there is an apodized version.

For each detector array there are two pre-processed cubes

- HR_SXW_spectrum2d
- HR_SXW_spectrum2d_apod

and four cubes

- HR_SXW_cube
- HR_SXW_cube_convol
• HR_SXW_cube_apod

• HR_SXW_cube_convol_apod

where “SXW” represents the detector array name, which can be either “SLW” or “SSW”.

The standard FTS products contain spectra with an instrumental line shape that is well approximated by a sinc profile, whereas the apodized versions have essentially been smoothed, which gives a more Gaussian line shape. The apodized data is not recommended for science analysis as it has been degraded in spectral resolution. However, this data may be useful for creating cleaner plots and visual inspection. See Section 7.11 and Section 7.12 for more details on spectral analysis of FTS data.

Any of the Level-2 spectral products can be opened in the Herschel Spectrum Explorer tool (see Chapter 5 and Chapter 6 of the Herschel Data Analysis Guide). This includes a spectrum from a single detector or a full SPSS, SDS, Spectrum2d or spectral cube. Right-click on the chosen products and then select Open With > Spectrum Explorer to open the product in the necessary view.

### 7.2.3.1. Sparse point observations

Although the generic HIPE Spectrum Explorer can be used to view all spectral products, it is not optimised for FTS products. There is a dedicated SPIRE tool called the SDS Explorer, which was developed specifically for FTS sparse observations. Any sparse Level-2 product can be opened with the SDS Explorer by right-clicking on the product in the Context Viewer or in the variables panel and selecting Open With > SDS Explorer. This will open the product in a footprint view of the SLW and SSW arrays, and allow the spectrum from one or multiple detectors (from more than one observations) to be plotted for easy examination and comparison. To the bottom right there is the Preferences Panel, which contains options to manipulate the plot. There are additional options available in the bottom left Control Panel, including an option to plot thumbnails from all detectors in a single array or the coaligned detectors from SLW and SSW. The SDS Explorer is described in more detail in Section 8.1.

The individual spectra within a product can also be examined in HIPE as a table of data using the Dataset Viewer (see Figure 7.8). Expand the product for inspection using the + symbol to show the scan dataset “0000” (labeled zero because all scans have been averaged). Expanding the scan folder will show the individual datasets for each FTS detector. Right-click on any detector name and select Open With > Dataset Viewer to open a view of the numeric values of the dataset. These values can also be written into a text file by simply right clicking on the detector name and selecting Send To and then Text file, which opens the asciiTableWriter task.
Figure 7.8. Inspecting spectral data from a Level-2 product in table form

The command line syntax required to access the standard Level-2 product for a sparse observation is as follows:

```python
extendedSds = obs.level2.getProduct('HR_spectrum_ext')
pointSourceSds = obs.level2.getProduct('HR_spectrum_point')
```

Note that the right hand side of these commands can be obtained from the top of the Data area of the Context Viewer in HIPE, where they are written in a longer, but equivalent format. Clicking the copy icon at the top right corner will copy the command string into the clipboard (see the red box on the right hand side of Figure 7.10).

Each spectral product contains a single averaged spectrum for each detector, where all of the individual scans of the FTS mirror have been averaged, so the scan number is set to zero. The individual scan data for each detector can be accessed on the command line using square bracket notation, e.g.,

```python
spectrum = pointSourceSds[0]['SLWC3']
```

This will extract the simplest SPIRE FTS spectral dataset, a SpireSpectrum1d, which is described in the Herschel Scripting Guide, Section 3.2 Spectrum1d.

There are a range of useful Spectrum Explorer tools that can be applied to the Level-2 spectral products as a whole, or to individual SpireSpectrum1d datasets. These tools are described in the Herschel Data Analysis Guide, Chapter 5. For example, there are tasks available to add, subtract, divide and multiply, while the corresponding arithmetic operators +, -, /, and * can be used as proxies for these tasks when scripting. There is also a task to convert the wavescale units, e.g. to convert GHz to µm:

```python
pointSourceSds = convertWavescale(pointSourceSds, to="micrometer")
```

Spectral data for a single detector spectrum can be extracted from a sparse Level-2 product using:
The following example will plot the spectra from the two centre detectors:

```
# Create a PlotXY() object
p = PlotXY()
# Loop over the centre detectors to add them to p as a new layer
for detector in ['SLWC3', 'SSWD4']:
    freq = pointSourceSds[0][detector].wave
    flux = pointSourceSds[0][detector].flux
    layer = LayerXY(freq, flux, name=detector)
    p.addLayer(layer)
# Set the plot title and axis labels
p.titleText = 'Plot example for Level-2 data'
p.xtitle = 'Frequency (GHz)'
p.ytitle = 'Flux Density (Jy)'
# Set the legend visible.
# The detector names are automatically assigned.
p.legend.visible = True
# Save the plot as a PDF
p.saveAsPDF('/path/to/folder/plotName.pdf')
```

The result from this code is shown in Figure 7.9. See the section on Plotting in the Herschel Data Analysis Guide for an in-depth guide to PlotXY, which has a wide range of options available.

![Plot example for Level-2 data](image)

**Figure 7.9.** Plot of the centre detectors of a point-source calibrated Level-2 product, which was generated using the example code given in the main text.

### 7.2.3.2. SpecMosaic: Plotting data from many detectors

There is a Useful script provided in HIPE, which makes use of the SpecMosaic class in order to generate thumbnail plots of spectra from different detectors at their location in their respective array. The script can be found in the the HIPE Scripts > SPIRE Useful scripts menu, under the name Spectrometer Thumbnail Mosaic Plot. The script works with a Spectrometer Point Source Spectrum product, or a Spectrometer Detector Spectrum product (i.e. the Level-2 point or extended calibrated products from the pipeline), although the extended calibrated product must be used if all detectors are to be available for plotting. The script provides the option to load the product from a local pool (by selecting `getData='fromObsContext'` and supplying an observation ID and pool name) or from a FITS file (by selecting `getData='fromFitsContext'` and supplying the path and filename of a Level-2 product save as FITS). If `getData` is set to `'fromObsContext'` and the
observation is not found on the local hard disk, the script will automatically attempt to download the relevant Observation Context from the HSA. An example observation ID is provided in the script, which means if there is internet connection, the demo will run without need for additional information.

The script presents five worked examples that will:

1. Reproduce the SDI/SDS Explorer thumbnail plots, by plotting data directly from the SDS product, see Section 8.1.4.

2. Explicitly set the plots to include in the mosaic. A loop iterates through all detectors and the result will look identical to the first example if left unedited. However, each plot could be customised by setting the range, symbols etc., before adding it to the mosaic. Note that example four shows how to set a global plot range for all of the mosaic plots.

3. Make a customised mosaic plot of the coaligned detectors.

4. Select specific detectors to be plotted. For example, the partially vignetted detectors (the outer ring of each array), or the thermistors, or resistors and dark detectors could be included. Set a global axis range for all plots.

5. Set the title and subtitles for the mosaic plot and save the plot to file.

### 7.2.3.3. Mapping observations

The Level-2 data for mapping observations consist of SPIRE preCube products, and spectral cubes (SpectralSimpleCube objects), which have an identical format to spectral cubes produced by PACS and HIFI. The preCube contains a Spectrum2d dataset which collects together all of the spectra observed by different detectors at each jiggle/raster position. The preCube contains columns which give the longitude and latitude, detector name, and jiggle and raster position for each spectrum.

For each observation, there are spectral cubes and preCubes for each array and for standard and apodized data. The two cubes produced have different WCS for SLW and SSW, with pixel sizes set according to the average beam size for each array. Inside each cube, there are several 3D datasets, which are:

- **flux**: the mean of all spectra within a grid square for the naive (naive projected) cubes or for the CP (convolution projected) cubes, the Gaussian weighted mean of all spectra falling within the width of the kernel.

- **error**: the standard error on the mean of spectra within a grid square for the naive cubes or the weighted error for the CP cubes.

- **coverage**: the number of spectra (scans) within a grid square, or for the CP cubes, the sum of the weighted contribution from all spectra.

- **flag**: available for flagging, although not currently set by the pipeline.

These products and the datasets they contain, are also described in Section 7.8.5 and Section 7.8.6. For cubes produced with HIPE version 12 or later, the frequency information is stored as part of the WCS third axis, which is defined by the keywords CRVAL3, CRPIX3 and CDELT3. For previous versions of HIPE, the frequency information was contained within the cubes as a table dataset called ImageIndex, which provided the mapping of the third dimension index to frequency, in GHz.

The following command will extract the preCube, the naive cube and the CP cube for the SSW band, from a high resolution Observation Context loaded into HIPE as the variable obs:

```python
preCubeSsw = obs.level2.getProduct('HR_SSW_spectrum2d')
cubeSsw = obs.level2.getProduct('HR_SSW_cube')
scpCubeSsw = obs.level2.getProduct('HR_SSW_cube_convol')
```
FTS cubes are best viewed in HIPE using the *Spectrum Explorer* tool. Details of using this tool with spectral cubes are given in Chapter 6 of the *Herschel* Data Analysis Guide.

### 7.2.3.4. Extracting Spectra for use in External Applications

The sparse point Level-2 spectral products are in a format specific to SPIRE: the *Spectrometer Detector Spectrum* (SDS) for extended calibrated products and the *Spectrometer Point Source Spectrum* (SPSS) for point-source calibrated products. These products can be written to FITS files by right-clicking on their variable name in the HIPE variables panel and selecting *Send To > FITS File*. Or a single spectrum can be extracted from these product and saved in FITS format. The following commands show how to write an SDS (sds) to a FITS file, using the simpleFitsWriter task, and how to do the same for the sds SLWC3 spectrum.

```python
simpleFitsWriter(sds, file='Spectrum.fits')
simpleFitsWriter(sds[0]["SLWC3"], file='Spectrum_1_SLWC3.fits')
```

For mapping observations, the spectral cube product uses a format common with HIFI and PACS, so the data can easily be shared between instruments. The cube can be written to a FITS file in a same way as for the spectra:

```python
simpleFitsWriter(cube, file='Cube.fits')
```

HIPE also offers a generic spectral format to store a single spectrum that can be used for data from any of the three *Herschel* spectrometers (or external telescopes). This format is called a *Simple Spectrum*, which is described in more detail in the *Herschel Scripting Guide*. This product only contains essential information for a single spectrum, but can be written to a FITS or ASCII file that can be read into a HIFI or PACS only install of HIPE.

A dedicated task (*spireProduct2SimpleSpectrum*) is available from the command line to extract a single spectrum from any SPIRE spectral product (SDS, SPSS or Spectral Cube) and convert to the *Simple Spectrum* format. For example:

```python
# To extract a single spectrum from an SDS:
mySimpleSpectrum = spireProduct2SimpleSpectrum(sds, detector="SSWD4", scan="0000")
# Or to extract a single spectrum from a spectral cube:
mySimpleSpectrum = spireProduct2SimpleSpectrum(cube, spaxelX=1, spaxelY=2)
# And then write this spectrum to FITS:
simpleFitsWriter(mySimpleSpectrum, file='mySimpleSpectrum.fits')
```

The *spireProduct2SimpleSpectrum* GUI can be opened from the *Tasks* panel of HIPE.

The data from a *Simple Spectrum* can also be exported to ASCII. For this, a *Spectrum1d* needs to be extracted from the *Simple Spectrum*. To create a *Spectrum1d* from a *Simple Spectrum* and write to a comma separated ASCII file, use the following commands:

```python
mySpectrum1d = mySimpleSpectrum['dataset']
asciiTableWriter(table= mySpectrum1d, file='MySpectrum1d.txt')
```

The *asciiTableWriter* can also be run as a GUI, accessed by right clicking on the *Spectrum1d* variable and selecting "Send to" and "Text file". The *asciiTableReader* can be used to read the saved ASCII file:

```python
mySpectrum1d = asciiTableReader('MySpectrum1d.txt')
```

For additional formatting options for the *asciiTableWriter* see the *Herschel Data Analysis Guide, Chapter 2*.

### 7.2.4. The Level-1 Interferogram Data Products

Each product in Level-1 of the Observation Context represents a single scanning *building block* observed by the instrument. The naming convention for each *building block* prod-
uct is "Point_pointingIndex_Jiggle_jiggleIndex_resolution". For a sparse point observation, Level-1 contains a single entry. So for a HR observation, this would be a product called Point_0_Jiggle_0_HR, which contains the data of the first and only raster point of index 0 and the first and only jiggle position of index 0. For sparse raster, there will be a single product for each pointing made, all with the jiggle position of index 0. For mapping observations, there is one entry for each of raster and jiggle position, e.g., Point_0_Jiggle_1_HR, Point_1_Jiggle_1_HR, Point_3_Jiggle_4_HR and so on. Expanding the entry of one of these building block products, by clicking on the "+" symbol, shows that there is one product contained inside called the interferogram, as shown by the red box on the left hand side of Figure 7.10. The interferogram is a Spectrometer Detector Interferogram (SDI) product that has the units of Volts.

Figure 7.10. Viewing the SPIRE Level-1 Context. The interferogram is selected in the Data panel, which opens it in the SDS Explorer in the Display Panel tab.

The command line syntax required to access the interferogram product from the Level-1 Context is as follows:

```python
sdi = obs.level1.getProduct("Point_0_Jiggle_0_HR").getProduct("interferogram")
```

Individual interferograms are then accessed in the same way as the spectral products:

```python
interferogram = sdi[scanNumber][detectorName]
```

But note that whereas the spectrum products have been averaged over all scans and the scan number set to 0, the interferogram product has not been averaged, and contains one interferogram per scan, starting from scan 1.

The data inside the interferogram products are accessed in a slightly different way to the spectra, with the general syntax as follows:

```python
opd = sdi[scanNumber][detector].opd
signal = sdi[scanNumber][detector].signal
```

The specific syntax for scan number 1 and detector SLWC3 is:
7.2.5. The Spectrometer Level-0.5 Data Products

The Level-0.5 data products are created by processing the raw Level-0 data through the Common Engineering Conversion Pipeline. The Level-0.5 data consist of timelines converted into physical units (e.g. voltages). The Level-0.5 Context can be opened by clicking on the + next to level0_5.

The data contained in the Level-0.5 Context is the normal starting point for user reprocessing, if using the User Pipeline scripts provided in HIPE. Figure 7.11 shows an example of a Level-0.5 Context opened with the HIPE Observation Viewer, highlighting the different building blocks of the observation. For example in the observation with ID 1342227790, there are a total of 16 entries, each one representing one instrument building block, with the building block name and ID number given for each one. Each type of building block contains data relating to a single instrument operation, which are described in Table 7.3.

Figure 7.11. Anatomy of Level-0.5 Building Block structure for the Observation Context of observation 1342227790.
Table 7.3. Description of the Building Blocks in the Spectrometer Level-0.5 Context shown in Figure 7.11

<table>
<thead>
<tr>
<th>BB Hex prefix</th>
<th>BB Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xb6c80001</td>
<td>SpireBb_StartObsAll</td>
<td>Begin Observation</td>
</tr>
<tr>
<td>0xa1050001</td>
<td>SpireBbSpecSerendipity</td>
<td>Serendipity data during slew to object</td>
</tr>
<tr>
<td>0xa0b00001</td>
<td>SpireBbSOF1Config</td>
<td>Initial configuration of the Spectrometer SOF1 AOT</td>
</tr>
<tr>
<td>0x82130001</td>
<td>SpireBbSmecInit</td>
<td>Initialize the SMEC</td>
</tr>
<tr>
<td>0xa0b10001</td>
<td>SpireBbSOF1Init</td>
<td>Initialize the AOT</td>
</tr>
<tr>
<td>0xa1070001</td>
<td>SpireBbBsmMove</td>
<td>Move BSM to position for this set of FTS scans</td>
</tr>
<tr>
<td>0x86410001</td>
<td>SpireBbSetBsmSampling</td>
<td>Set BSM sampling rate for FTS scanning</td>
</tr>
<tr>
<td>0x86420001</td>
<td>SpireBbSetSmecSampling</td>
<td>Set SMEC sampling rate for FTS scanning</td>
</tr>
<tr>
<td>0xa106a0001</td>
<td>SpireBbFtsScan</td>
<td>Science FTS scans</td>
</tr>
<tr>
<td>0xb6c20001</td>
<td>SpireBb_MoveSmec2Home</td>
<td>Move SMEC to home position after scanning</td>
</tr>
<tr>
<td>0x86410002</td>
<td>SpireBbSetBsmSampling</td>
<td>Reset BSM sampling rate after scanning</td>
</tr>
<tr>
<td>0x86420002</td>
<td>SpireBbSetSmecSampling</td>
<td>Reset SMEC sampling rate after scanning</td>
</tr>
<tr>
<td>0xb6b90001</td>
<td>SpireBb_PcalFlash</td>
<td>Calibration Lamp Flash</td>
</tr>
<tr>
<td>0xb6c30001</td>
<td>SpireBb_MoveSmec</td>
<td>Move SMEC to rest position</td>
</tr>
<tr>
<td>0xa0b20001</td>
<td>SpireBbSOF1End</td>
<td>End AOT Observation</td>
</tr>
</tbody>
</table>

The individual building blocks in the Level-0.5 Context contains a number of different Product types, the names of which can be viewed by clicking on the + sign for a given Building Block. Figure 7.12 shows an example of the products contained in a building block, referred to by the acronyms:

- **SDT**: the Spectrometer Detector Timeline, which contains the Level 0.5 detector data
- **BSMT**: the Beam Steering Mechanism Timeline, which contains the information of the BSM
- **SMECT**: the spectrometer Mechanism Timeline, which contains the information of the position of the SMEC (the moving FTS mirror) as a function of time
- **NHKT**: the Nominal House Keeping Timeline, which contains the housekeeping data with all the settings for this observation
- **CHKT**: the Critical House Keeping Timeline, which contains all the critical parameters of the instrument such as the electronics
- **SCUT**: the Sub Control Unit Timeline, which contains monitoring data for the instrument operation for this observation
- **SOT**: the Spectrometer Offset Timeline, which contains all the raw DC offsets in ADU that have already been used in the raw data processing to set the dynamic range of the detectors
- **MCUET**: the Mechanism Control Unit Engineering Timeline contains, which information on the SMEC (position sensors etc)
Figure 7.12. Inside the Level-0.5 Building Block structure for a spectrometer observation

The CHKT, NHKT, BSMT, SOT, SCUT products all contain a signal table of data arrays and a mask table containing flag information. The Level-0.5 SDT Spectrometer Detector Timeline Products contain 5 Table Dataset arrays of:

- **voltage**: A table containing the Sample Time (in seconds) and a column for the signal measured in Volts for every bolometer including both detector and non-detector (e.g. thermistor, resistor) channels.

- **resistance**: A table containing the Sample Time (in seconds) and a column for the Resistance measured in Ohms for every bolometer, including both detector and non-detector (e.g. thermistor, resistor) channels.

- **mask**: A table containing the Sample Time (in seconds) and a column for every bolometer, including both detector and non-detector (e.g. thermistor, resistor) channels with a mask value corresponding to which processing flags have been raised. The masks are defined in Section 8.4.1.

- **quality**: A table containing any Quality Flags raised for each detector.

- **temperature**: A table containing the Sample Time (in seconds) and the temperature of the Thermistors in Kelvin.

In Figure 7.12 the SDT Building Block has been selected. Right-clicking and selecting Open with > Dataset Viewer, opens the voltage table in a new window. The SDT Table Datasets can also be viewed graphically using the Detector Timeline Explorer (see Section 8.2) by selecting the SDT and right clicking Open with > Detector Timeline Viewer as shown in Figure 7.13. The bolometer signal to plot can be selected from the array footprint on the left-hand-side. In this example, the signal versus sample time for bolometer SSWD4 has been selected. The forward and reverse scans of the SMEC are highlighted in this figure.
7.2.6. The Spectrometer Level-0 Data

The "raw" data, formatted from the satellite telemetry, is available within the Observation Context as the Level-0 Products. Under normal circumstances, it should not be necessary to look at these data, or re-run any processing that goes back to Level-0. It is much more informative to look at the Level-0.5 data which has been converted from engineering to physical units. For background information, a short description of the Level-0 products follows.

The Level-0 Context, open inside the Observation Context Viewer in HIPE is shown in Table 7.3. It contains entries corresponding to the "building blocks" that make up the observation (see Table 7.3). In order to reduce the raw data volume at the Level-0 stage, all the data from a single building block are packed into a single Level-0 product, referred to as a raw SPIRE Timeline (RST). A single RST product contains many separate Table datasets (one for each telemetry packet produced from the satellite).

Note that the Level-0 products contain data in engineering units, which extends to the detector names. These names are specified in the raw Spectrometer Detector Timeline (SPECF) Table Dataset using their channel number, as SPECFARRAY001 to SPECFARRAY072. A full description of the Level-0 data structure can be found in the Herschel Products Definition Document.
7.3. Spectroscopy Pipeline Step-by-step

For the standard processing applied at the Herschel Science Centre (HSC), FTS data is divided between two data-reduction pipelines, SOF1 and SOF2, where SOF stands for Spectrometer Observatory Function. The choice of pipeline for any FTS observation depends on the observing modes used, see Section 7.2.1 for an explanation of the different combinations of FTS observing modes that were available during the Herschel mission. The SOF1 pipeline is used for reducing sparse point and sparse raster observations, while SOF2 is for intermediate and fully sampled point and raster observations, where 4-point jiggle and 16-point jiggle positions were observed. These pipeline scripts, known collectively as the Standard Product Generation (SPG) scripts, can be found in the HIPE Pipelines > SPIRE > SPG scripts menu. It is not recommended to use the SPG scripts for reprocessing FTS data, as they were not developed for general use outside of the HSC and are overly complicated for reprocessing purposes. If reprocessing is required, particularly if the intermediate pipeline products need to be examined, then there are two clearly laid out user pipeline scripts available in the HIPE Pipelines > SPIRE menu, which are described in detail in this section. For sparse point observations, the Spectrometer Single Pointing user pipeline should be selected, and the Spectrometer Mapping user pipeline for all other observation modes (sparse raster and mapping). The sequence of processing steps applied to the data (strictly following the user pipelines) are illustrated schematically in Figure 7.15 and described in this section, while in-depth detail of FTS processing and the SPG pipelines are presented in Fulton et al. (2015), MNRAS, in press. Most figures included in this section are to illustrate the Single Pointing pipeline steps and make use of a calibration observation of NGC 7027 (observation ID 1342197486).
Figure 7.15. The SPIRE Spectrometer pipeline. Note that the "Bright Gain" correction is only applied to Bright-mode observations and the "LR Correction" is only applied to low resolution observations.

### 7.3.1. Reprocessing SPIRE spectrometer data

The *Herschel* Science Archive (HSA) reprocesses every *Herschel* observation for each new HIPE release. It takes considerable time before this bulk reprocessing is completed, therefore the quality of an observation obtained from the HSA may be improved by reprocessing using the latest pipeline and calibration products. In order to check whether reprocessing is necessary, you can verify the version of the calibration tree associated with an Observation Context `obs` with the following command:

```
print obs.calibration.version
```

See [Chapter 5](#) for more details on the calibration context. If the calibration tree is not the most recent version, there are two ways to update an observation to the latest data processing:

1. Using the "on-demand" reprocessing available from the HSA, which is described in more detail in Chapter 1 of the *Herschel* Data Analysis Guide. The reprocessing is carried out at the HSC and once complete, the updated Observation Context is made available for download. This is the recommended and simplest way to reprocess your data with the latest pipeline.
2. Reprocessing in HIPE with the user pipeline scripts. If the default settings are not changed, the products produced by these scripts should match the data obtained via option 1. However, unlike for on-demand reprocessing, the script can be edited to try and improve on the final products. For example, for a mapping observation of a faint source, the vignetted detectors could be omitted when creating the spectral cubes, to see if this improves signal-to-noise, or a different baseline subtraction of the interferogram could be tested (see Section 7.3.15). Using these pipelines also gives easy access to intermediate products that are not stored in the Observation Context during data reduction.

While data reprocessing may result in an improvement to the final spectra, it may not eliminate all spectral artefacts, particularly for faint sources. If spectral artefacts persist, it is recommended to consult Section 7.5 and Section 7.6.2 to determine the best way to further process the data.

There are several documents that provide complementary information on SPIRE data and data processing:

- The SPIRE Handbook (formerly the SPIRE Observers Manual) gives more detail on the design and characteristics of the SPIRE instrument.
- The SPIRE User's Reference Manual describes each HIPE command specific to the SPIRE instrument and how to use it.
- The SPIRE Pipeline Specification in SPIRE Pipeline Specification Manual describes each data processing module that can be employed in the processing pipelines and how to use it.
- The SPIRE Spectrometer Pipeline Description explains the processing modules in mathematical detail.

### 7.3.2. Memory requirements

Several pipeline tasks run internal loops over detectors or scans as parallel processes, i.e. using multi-threading. Usage of the multi-threaded mode can result in significantly faster execution of these tasks, typically by a factor equal to the number of processors used. The drawback to this mode is that a task operating with multiple threads will require additional RAM in order to execute, with the amount needed depending on the number of processors used. Therefore it is difficult to estimate the total memory required without knowing how many processing threads are possible on a particular machine.

Multi-threaded processing is the default setting of the FTS pipeline tasks. It can be controlled by way of the HIPE property spire.maxthreads. By default, this property is set to zero (meaning use all available processors), but it may be modified using the advanced preference panel inside HIPE. From the HIPE top menu bar, go to Edit > Preferences > Advanced, and then search for the spire.maxthreads property. In order to disable multi-threaded execution, the value of spire.maxthreads should be set to 1.

On a machine with 3GB of RAM allocated to HIPE, it should be possible to use the Single Pointing user pipeline script to process the centre detectors for:

- between 108 and 124 repetitions with a single thread
- between 78 and 91 repetitions with 4 threads

For fully sampled mapping observations, up to 10 repetitions can be processed with 3GB, assuming only one thread. For sparse point observations, when processing all detectors, up to 16 repetitions can be processed with one thread. The amount of memory required increases with the number of threads used.

### 7.3.3. Introduction to the User Pipeline Scripts

HIPE contains two user pipeline scripts for reprocessing FTS data. These scripts can be found in the Pipelines > SPIRE menu. The Spectrometer Single Point user pipeline is for sparse point observations
and Spectrometer Mapping user pipeline is for sparse raster and mapping observations based on 4- or 16-point jiggle and raster patterns.

At the top of each script is an explanation of its purpose and usage options, which is followed by a summary of major changes or additions to the script since HIPE version 9. Two functions are then defined (as detailed in Section 7.3.32). After which are lists of SLW detectors required to apply the low resolution correction to extended-source calibrated data (see Section 7.3.26). The main part of the script starts after these, with the User Selectable Options section, where input of key information is required before the scripts can be successfully run.

When running the scripts, the data processing tasks overwrite the input products whenever possible to minimise the amount of required computer memory. To keep a copy of the input into a task, the `product.copy()` method should be used prior to applying the task, as this creates a deep copy, i.e. one that will not change when the original product is changed. `copy()` is available for data products of the type Spectrometer Detector Timeline (SDT), Spectrometer Detector Interferogram (SDI), and Spectrometer Detector Spectrum (SDS).

For example, to compare the input and output products of the wavelet deglitcher task, execute the following two commands and view the two data products in the Detector Timeline Viewer:

```
sdtOriginal = sdt.copy()
sdt = waveletDeglitcher(sdt)
```

### 7.3.4. User Selectable Options

The user pipeline scripts are configured by the following options, which can be found and edited in the User Selectable Options section, near the top of the script. Before running either script, a minimum of providing an observation ID for option A and an output director for option C are required. The scripts assume that the latest Calibration Tree is required, and will download this directly from the HSA. There are example commands given in option G for downloading the Calibration Tree and saving it to a local pool, or if already saved, loading it from a local pool. Option F sets how the script should access the Observation Context containing the data to be reduced, either locally or directly from the HSA. The full list of options to configure the scripts are:

#### A. Specify OBSID

Specify the observation ID number.

```
myObsid = enterOBSID
```

#### B. Limit the number of detectors

This option only applies when reprocessing with the Single Point user pipeline, since the mapping script makes use of data collected by all the detectors.

The default setting for this option is 1 (True), which results in only the central detectors SLWC3 and SSWD4 being processed. If this option is set to 0 (False), all the detectors will be processed. Only processing the central detectors significantly reduces memory usage.

```
processOnlyCenterDetectors = 1
```

#### C. Specify the output directory

The output directory is where the final processed products will be written to FITS files. Note that apodized data are created, but not saved unless `apodize` is set to 1.

```
outDir = '/enter/path/here/*
apodize = 0
```

#### D. Process high or low resolution data
This option only applies to the processing of high plus low resolution (H+LR) observations. Changing this option has no effect for observations that were not observed in H+LR mode. Observations in H+LR mode contain building blocks for high resolution (HR) and for low resolution (LR). For such an observation, the script can process data in either HR (only from the HR scans) or in LR (from the HR and LR scans). This selection is made by setting the variable `processRes` to either "HR" or "LR".

```
processRes = "LR"
```

E. Set the spatial grid

This option only applies to reprocessing with the mapping pipeline script.

The spacing of the grid used to create spectral cubes for the SSW and SLW arrays must be specified in degrees. Three options are defined in a Jython dictionary and the script selects the appropriate spacing depending on the map sampling (full, intermediate or sparse) and the detector array. The spacings are set to half beam (full), one beam (intermediate) or two beam (sparse), with assumed beam sizes of 19 arcseconds for SSW and 35 arcseconds for SLW. These numbers can be adjusted to reduce holes in the final map (see Section 7.8).

```
sswFwhm = 19.0 / 3600.0
slwFwhm = 35.0 / 3600.0
gridSpacing= {"full":   {"SSW": 0.5 * sswFwhm, "SLW": 0.5 * slwFwhm}, \
              "intermediate": {"SSW": 1.0 * sswFwhm, "SLW": 1.0 * slwFwhm}, \
              "sparse":      {"SSW": 2.0 * sswFwhm, "SLW": 2.0 * slwFwhm}}
```

F. Read the input data

`getObservation` loads the input data in the form of an Observation Context (`obs`). The default option results in `getObservation` searching for an observation with the observation ID (`myObsid`). Three areas on disk will be searched, in the following order: a pool in your local pool directory whose name includes `myObsid`, all other pools in your local pool directory, and your MyHSA pool. A specific pool in your local pool directory can be set using `poolName="poolName"`. Or data can be loaded directly from the HSA by using `useHsa=True`. See the Herschel Data Analysis Guide for more options.

```
# To search for the observation in your local store use:
obs = getObservation(myObsid)
# To specify a pool name in your local store use:
#obs = getObservation(myObsid, poolName="poolName")
# To load data directly from the HSA use:
#obs = getObservation(myObsid, useHsa=True)
```

G. Load the Calibration Context

Calibration files are included with all observation data and follow the naming convention `spire_cal_X_Y`, where `X` and `Y` specify the Calibration Tree version, e.g. `spire_cal_14_3`. However, these files will correspond to the version of HIPE used to bulk processes the data at the HSA. Therefore it may be necessary to retrieve and store the latest Calibration Context locally. Unless directed otherwise, the script will automatically download the latest Calibration Tree directly from the HSA, but will not save it. There are commented out commands given in the script to download and save the latest Calibration Tree or, if already saved, to load from a local pool:

```
# Load the latest calibration tree from the HSA
cal = spireCal(calTree = "spire_cal")
# To load the calibration tree from the HSA and save it as a pool use:
#cal = spireCal(calTree="spire_cal", saveTree=True)
```
# If the latest calibration tree is already saved
# locally use:
#\texttt{cal = spireCal(pool="poolName")}

See Chapter 5 for more details on SPIRE calibration data.

### 7.3.5. Preparing to process

Before the script starts processing the data, the observation mode and instrument mode of the observation are checked to see if the appropriate pipeline script is being run. If a mapping observation is found by the Single Pointing pipeline or a single pointed observation is found by the Mapping script, then each script prints a warning to this effect and asks if processing should continue. Using the wrong script will lead to poorly calibrated data, so it is best to take the script's advice.

In the user input section, setting the boolean variable `processOnlyCenterDetectors` to `True` will result in only the centre detectors being processed, with the other channels removed. To this end, the centre detectors are defined as a list.

```python
# Define the central detectors:
centreDetectors = ["SLWC3","SSWD4"]
```

The loaded calibration context is attached to the Observation Context. A set of calibration variables is then assigned for use within the script. The calibration applied depends on the bias mode (nominal or bright), which is established prior to accessing the calibration files.

```python
obs.calibration.update(cal)

# Find out the bias mode of the observation {nominal/bright}:
biasMode = obs.meta["biasMode"].value

# Extract necessary Calibration Products from the Observation Context
nonLinCorr    = obs.calibration.spec.nonLinCorr
chanNum       = obs.calibration.spec.chanNum
bolPar        = obs.calibration.spec.bolPar
lpfPar        = obs.calibration.spec.lpfPar
phaseCorrLim  = obs.calibration.spec.phaseCorrLim
chanTimeConst = obs.calibration.spec.chanTimeConst
bsmPos        = obs.calibration.spec.bsmPos
detAngOff     = obs.calibration.spec.detAngOff
smec2pd       = obs.calibration.spec.smec2pd
chanTimeOff   = obs.calibration.spec.chanTimeOff
smecStepFactor= obs.calibration.spec.smecStepFactor
opdLimits     = obs.calibration.spec.opdLimits
bandEdge      = obs.calibration.spec.bandEdge
brightGain    = obs.calibration.spec.brightGain
extCorr       = obs.calibration.spec.extCorr
lrCorr        = obs.calibration.spec.lrCorr

# \texttt{teleModel} contains the OPD-dependent emissivity correction
# factors that are applied to the telescope model.

# teleModel = obs.calibration.spec.teleModel
```

The pipeline requires access to three auxiliary products relating to the Herschel satellite: The Herschel Pointing Product (`hpp`) and the Spacecraft Instrument Alignment Matrix (`siam`) allow accurate pointing information for SPIRE to be calculated; the Herschel Housekeeping Product (`hk`) details the temperatures of the primary and secondary mirrors of the telescope for an accurate telescope removal, see Section 7.3.24.

```python
hpp  = obs.auxiliary.pointing
siam = obs.auxiliary.siam
hk   = obs.auxiliary.hk
```

Level 0.5 is obtained from the Observation Context.

```python
level0_5 = obs.level0_5
```
7.3.6. Start processing from the Level 0.5 products

The user scripts implement a loop over FTS scanning building blocks, starting with building block ID (bbid) 0xa1060001, and loads Level-0.5 products from each one. For point source observations there is usually only one scanning building block that contains the observation data, except for long observations before operational day (OD) 302 and observations taken in H+LR observing mode. Both point source and mapping scripts set up a list to contain the results for each building block. The following is taken from the point source user pipeline.

```python
bbList = SpireListContext()
```

For mapping observations, each jiggle/raster position is contained in a separate building block. The mapping script uses lists with different names and propagates the metadata from Level 0.5 to these lists:

```python
sdsList = SpireMapContext()
sdsList_apod = SpireMapContext()

# Propagate the metadata from the Level 0.5 context to the lists
for key in level0_5.meta.keySet:
    if key != "creator" and key != "creationDate" and key != "fileName" and \
    key != "type" and key != "description":
        sdsList.meta[key]=level0_5.meta[key].copy()
        sdsList_apod.meta[key]=level0_5.meta[key].copy()
```

both scripts then loop over the scanning building blocks:

```python
for bbid in obs.level0_5.getBbids(0xa106):
```

The following lines extract the detector timeline product from the Observation Context and attach to it the version of the calibration context for future reference. The sample observation contains 10 spectral scans, see Figure 7.16. The different features of the interferogram in each scan are described in more detail in Figure 7.17.

```python
sdt = obs.level0_5.get(bbid).sdt
sdt.calVersion = obs.calibration.version
```

![Image of detector timeline](image.png)

Figure 7.16. The SLWC3 detector timeline for the 10 scan example observation of NGC 7027. The timeline can be visualised by double-clicking the sdt variable.
The following line extracts the nominal housekeeping timeline product (NHKT) from the Observation Context.

```
nhkt = obs.level0_5.get(bbid).nhkt
```

The NHKT product contains the instrument "housekeeping" data, for example, temperatures of various instrument components, as well as voltages, phases, etc. The steps below can be followed to view instrument thermometry as shown in Figure 7.18:

1. Open the `nhkt` variable from the variables list with a double click.
2. Right-click on `Signal` and open with `TablePlotter`
3. Activate the drop-down menu for the y-axis to plot any of the instrument sensor measurements, e.g., SCALTEMP.

```
smect = obs.level0_5.get(bbid).smect
```

The TablePlotter can be used to produce a sample plot of the mirror scan distance during the observation, as shown in Figure 7.19.
Figure 7.19. The position of the stage mechanism as a function of time during the observation.

The correlation between the SCALTEMP plot in Figure 7.18 and the mirror stage mechanism plot in Figure 7.19 illustrate how the operation of the stage mechanism affects the instrument temperature.

The beam steering mirror timeline contains the positions of the mirror that direct the precise sky position within the telescope field of view. This product is extracted from the Observation Context using the following command.

```python
bsmt = obs.level0_5.get(bbid).bsmt
```

### 7.3.7. Removing unnecessary channels

In the point source pipeline, if `processOnlyCenterDetectors` is set to `True`, the off-axis detectors are removed at this point, to avoid unnecessary processing. See the SPIRE Pipeline Specification Manual in SPIRE Pipeline Specification Manual for all the options of the `filterChannels` module.

```python
if processOnlyCenterDetectors:
    sdt = filterChannels(sdt, keepChannels=centreDetectors)
```

### 7.3.8. Identify the jiggle position

In the mapping pipeline, it is necessary to identify the jiggle position (BSM angle) and raster position of each building block. The mapping pipeline uses the jiggle position ID (`jiggId`) to retrieve suitable calibration data.

```python
# Extract the jiggle ID from the metadata:
jiggId = sdt.meta['jiggId'].value
# Extract raster ID from the metadata:
raster = sdt.meta['pointNum'].value
```

### 7.3.9. First level deglitching

The first level deglitching task is used to remove the effects of cosmic ray hits from the timelines. Strong glitches (e.g. Figure 7.20) must be dealt with at the beginning of the pipeline to prevent their effect being spread into neighbouring data samples by some of the downstream pipeline tasks, in particular the time-domain phase correction or the baseline correction, e.g. see Figure 7.25. If a glitch remains untreated, its effect on the resulting spectrum depends on its location relative to the Zero Path Difference (see Figure 7.21). A glitch at an Optical Path Difference (OPD$_{glitch}$) will lead to a sinusoidal artefact in the spectral domain with a period of $1 / \text{OPD}_{\text{glitch}}$.

The pipeline and user scripts employ a wavelet-based deglitching algorithm, as described in the SPIRE Pipeline Specification Manual in SPIRE Pipeline Specification Manual. The parameters for
the waveletDeglitcher are conservatively set to avoid any damage to the centre burst of the recorded interferograms. In particular interference patterns from the SSW detectors run the risk of being mistaken as glitches. The parameters are robust and do not need to be adjusted for different types of observations (bright, extended, etc.). Typically, for high resolution observations, this module will flag about 1% of all data samples as glitches and correct them. Even fewer glitches will be identified for observations in low spectral resolution or bright detector settings. Conservative parameter settings does mean that this module will miss some glitches, but 2nd level deglitching will clean up the data further and remove many of the glitches not identified in the timeline, so they will not affect the quality of the final calibrated spectra.

The SigmaKappaDeglitcher, as described in the Section 6.6 in SPIRE Pipeline Specification Manual, also operates in the time domain and implements an alternative deglitching algorithm, based on statistical methods. This could be used instead of the wavelet method, but the main challenge with this module is to avoid damage to the centre burst signature of interferograms, while still detecting glitches. Therefore the wavelet task is recommended for this step.

Data samples that have been identified as glitches must be replaced by either module since an interferogram with gaps cannot undergo a correct Fourier transform.

**Figure 7.20.** A glitch in the signal timeline of SLWC5 about 645.5s into the observation.

**Figure 7.21.** The spectrum will show a sinusoidal artefact if a glitch remains uncorrected. The frequency of the sinusoidal artefact is higher the further away from ZPD the glitch occurs. As an example, see the interferograms without any deglitching and resulting spectra for scans 2 (blue), 3 (black), and 8 (green) of SLWC5.
The user scripts rely on the default values for most of the waveletDeglitcher keywords:

```
sdt = waveletDeglitcher(sdt, optionReconstruction="polynomialAdaptive10", newThresholdCoef=True)
```

All relevant keywords are given below for reference. The SPIRE Pipeline Specification Manual in SPIRE Pipeline Specification Manual can be consulted for more details on these options.

```
sdt = waveletDeglitcher(sdt, reconstructionPointsAfter=3, 
    reconstructionPointsBefore=2, 
    correctGlitches=True, 
    scaleMin=1, scaleMax=8, scaleInterval=5, 
    holderMin=-1.4, holderMax=-0.6, 
    correlationThreshold=0.85, 
    optionReconstruction="polynomialAdaptive10", 
    degreePoly=6, fitPoints=8, newThresholdCoef=True)
```

![Figure 7.22. The signal timeline of SLWC5 before (blue) and after (red) executing the wavelet deglitcher with the settings given above.](image)

The easiest way to change the sensitivity of the deglitching module is to change the corruptionThreshold keyword. Using a number that is closer to 1 will make the module less sensitive to glitches. Using a number that is closer to 0 will make the module more sensitive to glitches. Users should take care to select this parameter such that the deglitching modules does not mistake parts of the centre burst as glitches, as is the case for the example in Figure 7.23. This problem occurs more readily for SSW detectors than for SLW detectors.
7.3.10. Account for non-linearities

The RMS voltage measured by the SPIRE bolometers only responds linearly to incoming radiation within a limited range of power, and therefore a correction for non-linearity is required. The correction is carried out in a similar way to the SPIRE photometer, producing a "linearised voltage" quantity that is proportional to the power on the bolometers. The linearisation parameters are contained within a calibration product and are different for nominal and bright modes.

```python
sdt = specNonLinearityCorrection(sdt, nonLinCorr=nonLinCorr)
```

One way to verify that these corrections have been applied successfully is through inspection of the out-of-band power in the resulting spectra, see Section 7.3.20.

7.3.11. Correct the detector signals for clipping

The detectors have a finite dynamic range and can therefore saturate when observing a particularly strong source. Saturation results in the extrema of the interference pattern being "clipped", which produces an incorrect spectrum. Consequently, saturated signal must be reconstructed prior to the Fourier transform.

```python
sdt = clippingCorrection(sdt)
```

The clippingCorrection module is based on fitting an 8th order polynomial function to data surrounding the clipped samples. By default, 5 data samples to either side of the clipped area are used to calculate the polynomial. The clipping module will run into problems if attempting to correct more than 8 consecutive clipped samples, or if there are fewer than five unclipped samples between the clipped data. See the SPIRE Pipeline Specification Manual in SPIRE Pipeline Specification Manual on how to set this option of the clippingCorrection module.
7.3.12. Correct time domain phase in the detector signals

The thermal response of the detectors and the read-out electronics are not instantaneous and so impart a time delay and slight distortion to the recorded signals, see Section 7.1.5. The resulting effect is dependent on the physical properties of the detection system and is corrected by a convolution with the time domain phase correction function, which is based on the electrical filters and the measured detector time constants.

```python
sdt = timeDomainPhaseCorrection(sdt, nhkt, lpfPar=lpfPar,\n    phaseCorrLim=phaseCorrLim, chanTimeConst=chanTimeConst)
```

Note that it is important to correct for strong glitches with the waveletDeglitcher task before correcting the time domain phase, as the convolution with the time domain phase correction function can create artefacts around strong glitches that remain in the timeline.

7.3.13. Create a SPIRE Pointing product

In order to calculate the absolute pointing of the detectors during the observation, the SPIRE pointing product collates information from four inputs:

1. The absolute pointing of the Herschel telescope (contained in the HPP product).
2. The offset of the detector arrays from the boresight of the Herschel telescope (contained in the SIAM product).
3. The relative offset of different detectors in each array from the centre detectors (contained in the DetAngOff calibration product).
4. The angle of the SPIRE Beam Steering Mirror (contained in the BAT).

```python
bat = calcBsmAngles(bsmt, bsmPos=bsmPos)
spp = createSpirePointing(hpp=hpp, siam=siam, \n    detAngOff=detAngOff, bat=bat)
```

The SPIRE Pipeline Specification Manual in SPIRE Pipeline Specification Manual gives more details about how to determine the position of each detector from the SPIRE Pointing product. During the pipeline, the createIfgm task uses this product to calculate the sky coordinates of each measured interferogram.

7.3.14. Interpolate SDT and SMECT to create interferograms

The FTS mirror (SMEC) position timeline from the SMECT is combined with the timeline of detector signals from the SDT to produce a Level-1 interferogram (i.e. the signal as a function of optical path difference). The timelines are firstly interpolated onto the same grid, then divided into discrete scans, and finally the pointing information for each scan is added (see the SPIRE Pipeline Specification Manual in SPIRE Pipeline Specification Manual for more details).

```python
sdi = createIfgm(sdt, smect=smeCt, nhkt=nhkt, spp=spp, \n    smecZpd=smeCtZpd,\n    chanTimeOff=chanTimeOff,\n    smeCStepFactor=smeCStepFactor)
```

In the point source pipeline, the Spectrometer Detector Interferogram (SDI) and the corresponding NHKT products are appended to a list (`bbList`) during the loop over building blocks, and then
merged into a single pair of SDI and NHKT products. This is required because scans in some observations taken early in the Herschel mission were split into several building blocks. In the case of H+LR mode, there are HR and LR scans contained in separate building blocks. For this mode, all the building blocks are combined if processing as LR, otherwise only the HR building blocks are combined. Combining building blocks is not necessary in the mapping pipeline, as this is done when the spectra are gridded into the final cube.

```python
bbMap = SpireMapContext()
if obs.meta["commandedResolution"]= "H+LR":
    # for processing all scans as LR
    if processRes == "LR":
        sdi.processResolution = "LR"
        bbMap.setProduct("ifgm", sdi)
        bbMap.setProduct("nhkt", nhkt)
        bbList.addProduct(bbMap)
    # for processing the HR scans
    elif processRes == sdi.processResolution:
        bbMap.setProduct("ifgm", sdi)
        bbMap.setProduct("nhkt", nhkt)
        bbList.addProduct(bbMap)
    else:
        bbMap.setProduct("ifgm", sdi)
        bbMap.setProduct("nhkt", nhkt)
        bbList.addProduct(bbMap)
# Loop over building blocks ends here
# -----------------------------------------------------------
# Merge all the building blocks into one:
merged = mergeFtsBuildingBlocks(bbList)
sdi = merged.getProduct("ifgm")
nhkt = merged.getProduct("nhkt")
```

When processing an H+LR observation, the resolution must be set correctly in the mapping script, as follows:

```python
if obs.meta["commandedResolution"]= "H+LR" and processRes == "LR":
    sdi.processResolution = "LR"
```

The next step is to truncate the interferograms to a fixed length:

```python
sdi = makeSameOpds(sdi, opdLimits=opdLimits)
```

### 7.3.15. Subtract the interferogram baseline

The baseline of interferograms must be subtracted as a precursor to statistical deglitching because the baseline of individual scans varies systematically over time. Removing the baseline also allows the interferograms to be easily zero-padded when calculating the Fourier transform, without creating spectral artefacts.

The basic interferogram baseline is approximately parabolic due to increasing levels of vignetting towards higher OPD. The curvature of the baseline also increases for detectors that are further from the optical axis due to higher levels of vignetting towards the edge of the overall SPIRE field of view (see Figure 7.24). Additional structure on top of this curvature is due to variation in the bolometer temperature with time, or for strong point-like sources, it can be related to jitter in the spacecraft pointing during the observation.
Figure 7.24. Thumbnail plot (Section 8.1.4) showing the interferogram for all SSW detectors after subtracting a constant from each one. The curvature of the baseline increases away from the centre of the array.

The baseline correction module can either fit the baseline with a polynomial function or use Fourier components up to a certain frequency threshold. The Fourier option has been found to be the most robust and is used for the standard pipeline. However, Fourier filtering creates sinc-like artefacts around glitches, contaminating a larger OPD range than the original glitch, see Figure 7.25. Such artefacts can be removed by second level deglitching. However, it is better to avoid creating these artefacts in the first place and retain as many valid samples as possible. When processing faint to medium strength sources it may therefore be preferable to use polynomial baseline fitting. Fitting a polynomial of even order (four or six) will remove the baseline equally well for extended sources and faint to medium strength point sources. Polynomial fitting will not cause artefacts around glitches and is also faster than Fourier filtering. See the SPIRE Pipeline Specification Manual in SPIRE Pipeline Specification Manual for the details on how to call baseline correction with polynomial fitting.
Inspecting the results at each pipeline stage provides a more visual representation of the changes in the data due to each processing step. Note that the baseline correction module overwrites the input variable and it is necessary to make a deep copy of the input variable in order to make a side-by-side comparison of the interferogram before and after baseline correction.

sdi = baselineCorrection(sdi, type="fourier", threshold=4)

### 7.3.16. Apply second level deglitching

For any given observation, second level deglitching relies on the SPIRE FTS having taken repeated scans at the same sky position and on the assumption that repeated measurements of the same source yield identical results, within the associated random noise. Two types of statistical algorithms are available to identify and remove outliers on an OPD-per-OPD basis: one uses a window that runs across OPD and the other compares interferograms at individual OPD locations. These algorithms are explained in the SPIRE Pipeline Specification Manual in SPIRE Pipeline Specification Manual. The windowed algorithm is recommended for observations with fewer repetitions (between 2 and 10), but this algorithm will not identify and remove outliers well if glitches have been spread out by Fourier filtering of the baseline, see Section 7.3.15. Both algorithms can use either the standard deviation (STD) or median absolute deviation (MAD) to define thresholds beyond which data are considered to be outliers. STD is not recommended for observations with fewer than 10 repetitions. The standard pipeline uses the MAD algorithm on each OPD bin:

sdi = deglitchIfgm(sdi, deglitchType="MAD")

Each available second level deglitching algorithm can be customised by setting the threshold keyword thresholdFactor to a suitable value. Decreasing the threshold factor increases glitch detection sensitivity. By default, the module sets the threshold factor based on the number of scans, to identify less than 1% of data samples as false positives.

The lower the number of repetitions, the higher the chance that glitches slip past the 2nd level deglitcher and contaminate individual scans. If outlier scans are seen in observations with low numbers of
repetitions, the threshold for interferogram deglitching can be lowered in an attempt to remove the offending glitches. In order to determine what the current default threshold is, either look in the SPIRE Pipeline Specification Manual in SPIRE Pipeline Specification Manual, or look at the log messages in HIPE when the deglitchIfgm task is run. The log message should be similar to:

```
WARNING GlitchFinderTask: No thresholdFactor parameter specified for MAD, using default: 30.0
```

In order to verify the changes due to second level deglitching, make a deep copy of the original SDI product before deglitching and then subtract the two SDI products and inspect that difference, e.g., with the SDI Explorer:

```python
sdiOrg = sdi.copy()
sdi = deglitchIfgm(sdi, deglitchType="MAD")
sdiDiff = sdiOrg - sdi
```

### 7.3.17. Correct interferogram phase

For the reasons described in Section 7.1.5, the strongest modulation in the SDI may not occur exactly at zero path difference. This modulation must be corrected before the Fourier transform or the resulting spectra will not be correct. The phase correction is calculated from the low resolution part of the average interferogram (i.e. the double sided part of the interferogram):

```python
avgSdiFull = averageInterferogram(sdi)
lowResSdi = avgSdiFull.copy()
lowResSdi.processResolution = "LR"
lowResSdi = makeSameOpds(lowResSdi, opdLimits=opdLimits)
sdi = phaseCorrection(sdi, avgSdi=lowResSdi, 
 avgSdiFull=avgSdiFull, spectralUnit="GHz")
```

The phase correction algorithm symmetrises the interferogram and reduces the residual phase to less than one degree. Correcting the phase below this level does not generally result in further significant improvements in the final spectrum.

### 7.3.18. Fourier transform the interferograms

After phase correction, the SDI undergoes a Fourier transform to produce a spectrum in the variable ssds.

```python
ssds = fourierTransform(sdi, ftType="postPhaseCorr", zeroPad="standard", \ 
 spectralUnit="GHz")
```

At this stage the frequency range is significantly wider than the optical passband (i.e. the scientifically useful range) of the detectors, and ranges from 0 to the Nyquist frequency, which is fixed at 200 cm\(^{-1}\) or 6000 GHz for all detectors (see Section 7.1.2). Zero-padding is used to extend the interferogram to a maximum OPD, such that a standardised frequency grid is maintained for all observations (see Figure 7.26).
Figure 7.26. The phase-corrected spectrum measured by the centre detectors in a single scan.

Zero-padding can lead to strong spectral artefacts if the interferogram baseline has not been removed correctly, see Figure 7.27.

Figure 7.27. The SSWD4 spectrum processed with the full user script (blue) compared to the results after processing the data with the baseline correction module disabled (turquoise).

7.3.19. Fetch calibration files

Three calibration products are retrieved to account for instrument correction and to convert to physical flux. Note that these calibration products depend on the detector bias settings, spectral resolution, and
for apodized data, the apodization applied. The `teleRsrfr` product also depends on jiggle position (in mapping observations). The products are retrieved from a list in the calibration context depending on the metadata in the SDS product being processed:

```python
instRsrfr = obs.calibration.spec.instRsrfrList.getProduct(ssds)
teleRsrfr = obs.calibration.spec.teleRsrfrList.getProduct(ssds)
beamParam = obs.calibration.spec.beamParamList.getProduct(ssds)
```

These commands will only work if the raw data was processed with HIPE version 7 or later, because the required metadata were not passed through the pipeline to the SDS products in earlier versions. In particular the `mapSampling` metadata value was not present before HIPE 7, which will lead to the following error: `java.lang.NullPointerException: Parameter [mapSampling] does not exist`. The version of the pipeline used to produce the raw data can be checked by looking at the `creator` metadata item in the Observation Context, which gives the version of the SPG pipeline used. In order to avoid this error, the observation should be re-downloaded from the *Herschel* Science Archive. Alternatively the `mapSampling` metadata could be copied by hand from the Observation Context:

```python
ssds.meta["mapSampling"] = obs.meta["mapSampling"]
```

### 7.3.20. Remove out-of-band power

To save processing time and computer memory, spectral data is truncated to the scientifically useful range (the optical passband) using:

```python
ssds = removeOutOfBand(ssds, bandEdge=bandEdge)
```

Inspecting out-of-band spectral data, prior to running this module, can help locate problems with processing steps that could potentially move power outside of the optical passband. **Figure 7.28** shows an example spectrum prior to truncation.

![Averaged SSWD4 spectrum showing little systematic power outside of the optical passband.](image)

**Figure 7.28.** Averaged SSWD4 spectrum showing little systematic power outside of the optical passband.

### 7.3.21. Apply bright gain correction

Observations taken with bright source settings require the application of an additional gain factor.
if biasMode == "bright":
    ssds = specApplyBrightGain(ssds, brightGain=brightGain)

7.3.22. Correct for instrument emission

The SPIRE instrument itself acts as an unwanted source of radiation measured by the SPIRE spectrometer. This radiation follows Planck's law for black body emission and therefore depends on the instrument temperature, which varies from within a range of 4 to 5 Kelvin. The instrument correction module uses the instrument thermometry (from the nhkt) measured during a given observation to subtract a black body model from the measured spectrum. The relative spectral response of the instrument is also given as an input:

\[
\text{ssds} = \text{instCorrection}(\text{ssds}, \text{nhkt}=\text{nhkt}, \text{instRsrf}=\text{instRsrf})
\]

The temperature between individual scans varies during an observation, due to the dissipation of power driving the instrument subsystems (see Figure 7.18). Therefore the correction for instrument emission must be performed before the scans have been averaged.

The SLW detectors are more sensitive to the black body radiation in this temperature range compared to the SSW detectors, as can be seen in Figure 7.29, which shows an example comparison of spectra before and after instrument correction.

![Figure 7.29. Spectra for the centre detectors before (blue) and after (red) instrument correction.](image)

7.3.23. Apply the extended source flux conversion

In order to calibrate the signal from V/GHz to brightness, the telescope Relative Spectral Response Function (RSRF) is used, which is derived from observations of the SPIRE dark sky field (i.e. of the telescope). This calibration is valid for uniformly extended emission across the beam. The resulting spectrum has units of W/m²/Hz/sr and is provided in the variable extended. In addition, applying this flux conversion removes the sinusoidal fringes related to the telescope emission that can be seen in the raw spectrum (e.g. Figure 7.29). This calibration is applied separately for the forward and reverse mechanism scanning directions.
The flux conversion modules takes the telescope RSRF into account. Whereas the response to radiation through the second input port was already accounted for by the instrument correction module.

```python
extended = specExtendedFluxConversion(ssds, teleRsrf=teleRsrf)
```

### 7.3.24. Correct for telescope emission

The primary and secondary mirrors of the *Herschel* telescope emit at temperatures between 80 and 90 Kelvin, which is a significant contributor to the flux measured by the SPIRE FTS, and in fact dominates the continuum for most observations. The primary and secondary mirror temperatures are derived from the satellite housekeeping product \( (hk) \) and are used to model the flux contributions as black body profiles, modified by the mirror emissivity. The \( \text{teleModel} \) calibration product contains an OD dependent empirical correction to the emissivity of the primary mirror:

```python
extended = telescopeCorrection(extended, hk=hk, teleModel=teleModel)
```

*Figure 7.30* shows an example of the telescope model, while *Figure 7.31* compares the associated spectra before and after the model is subtracted to remove the telescope contribution.

![Telescope Model (M1: 88.16K / M2: 84.28K)](image)

*Figure 7.30. The telescope model subtracted from the spectra in Figure 7.31.*
Figure 7.31. Spectra for the centre detectors before (blue) and after (red) telescope correction.

## 7.3.25. Apply the point-source flux calibration

In order to determine the flux density from an extended flux calibrated spectrum, a conversion table is used to apply a point source flux calibration. This calibration is valid when the extent of the source is much smaller than the area of the beams (the diameters of the centre SSW and SLW detectors are of the order 19 and 35 arcseconds). The point source conversion table is contained in the \texttt{beamParam} calibration product, which was derived from observations of Uranus and a detailed model, and has units of steradians. The table includes the beam area, and the ratio of the beam coupling efficiency for extended and point-like emission. See Section 7.6.1 for more details on the difference between extended and point source calibration.

The point-source flux conversion translates the spectra from surface brightness units of W/m$^2$/Hz/sr into flux density units of Jy (10$^{-26}$ W/m$^2$/Hz). Applying the conversion table also removes differences in the fringing pattern between extended and point-like emission. There is no point-source calibration available for the partially vignetted detectors, so these are removed before the flux conversion module is applied using the \texttt{filterChannels} task. The resulting point source calibrated spectra are provided in the SPSS variable \texttt{pointSourceSds}:

```python
pointSourceSds = filterChannels(extended.copy(), \
    keepChannels=beamParam.uniqueDetectors)
pointSourceSds = specPointFluxConversion(pointSourceSds, beamParam=beamParam)
```
Figure 7.32. Point source calibrated spectra from the centre detectors.

The point source calibrated data is a Level-2 pipeline product. An example of point-source calibrated data is shown in Figure 7.32, which plots spectra from the centre detectors for the example NGC 7027 observation. The mismatch in the overlap region of the continuum at around 900 GHz and the interrupted progression of line strengths of the $^{12}$CO transitions reflects the slight spatial extension of the source (see Section 7.6).

### 7.3.26. Apply the low resolution correction

For low resolution (LR) data, a correction is applied to remove artefacts from the extended-source and point-source calibrated spectra in the SLW array. These artefacts manifest as a characteristic double bump with peaks around 550 and 900 GHz, as illustrated by Figure 7.33. This systematic noise originates in fast temperature changes in the instrument, which is not properly accounted for in the LR calibration. A full analysis of this issue and the derivation of the empirical SLW LR correction is presented in Marchili et al. (2016).
Figure 7.33. The difference between the HR and LR point-source calibrated spectra for the centre detectors of the H+LR observation 1342253971. SSW is essentially flat, whereas there is a bumpy distortion in the difference for SLW. The peaks seen at 500 and 900 GHz are introduced by artefacts in the LR data. The LR data were interpolated onto the HR frequency scale before subtraction from the HR data.

There is a range of "bumpiness" seen for LR SLW spectra, with some observations showing insignificant effect and others exhibiting pronounced bumps. Therefore the applyLrCorr task takes a measure of the bumpiness present, so the right strength of correction will be applied. The SLW continuum level is also assessed by fitting the spectrum with a second order polynomial. If the continuum level is found to be above the value set by the limit parameter (0.5 Jy by default) then the fitted polynomial curve is subtracted before the bumpiness is measured and the correction applied. The value of limit can be raised to prevent a fit to the continuum or lowered if a continuum fainter than 0.5 Jy in SLW is known to exist.

The LR correction is applied per scan to account for fast changing instrument temperatures. As this correction was derived using point-source calibrated data, there is no correction for the vignetted detectors, however these are retained (uncorrected) in the sparse extended-source calibrated Level-2 product. For the mapping script, the uncorrected vignetted detectors will be present in all the pre-processed cubes, but are not used to create the SLW spectral cubes. Therefore the SLW cubes processed with HIPE versions prior to 14.1 will have a larger map area. This includes the LR SLW cubes from H+LR observations.

The correction is applied in the point pipeline as:

```python
if pointSourceSds.processResolution=="LR":
    # Extended
    extendedFull = extended.copy()
    extendedFull = filterChannels(extendedFull, removeChannels=String1d(lrSlwChannels))
    extended = filterChannels(extended, keepChannels=String1d(lrSlwChannels))
    extended = specPointFluxConversion(extended, beamParam=beamParam)
    extended = applyLrCorr(extended, lrCorr=lrCorr)
    extended = specLrExtFluxConv(extended, extendedFull, beamParam=beamParam)

    # Point
    pointSourceSds = applyLrCorr(pointSourceSds, lrCorr=lrCorr)
```

And in the mapping pipeline as:

```python
if ssds.processResolution=="LR":
    lrCorr = obs.calibration.spec.lrCorr
    beamParam = obs.calibration.spec.beamParamList.getProduct(ssds)
    ssdsFull = ssds.copy()
    ssdsFull = filterChannels(ssdsFull, removeChannels=String1d(lrSlwChannels))
    ssds = filterChannels(ssds, keepChannels=String1d(lrSlwChannels))
```

```python
```
ssds = specPointFluxConversion(ssds, beamParam=beamParam)
ssds = applyLrCorr(ssds, lrCorr=lrCorr)
ssds = specLrExtFluxConv(ssds, ssdsFull, beamParam=beamParam)

7.3.27. Average the spectra

```python
# Average across all scans:
extended = averageSpectra(extended)
pointSourceSds = averageSpectra(pointSourceSds)
```

The `averageSpectra` task is applied to average the scans for each detector. The mean is taken to give a single spectrum per detector, which is called scan “0000” in the SSDS. The task populates the “error” column with the standard error on the mean and adds a column to the output called “numScans”. This new column records the numbers of scans that have been averaged together, so a reapplication of the task will correctly propagate the errors.

7.3.28. Apodization

Copies of the `extended` and `pointSourceSds` variables are created and apodized using a standard apodizing function. The apodization is applied in the spectral domain by applying a Fourier transform to the apodization function and then convolving that with the spectrum. The apodization is carried out to the spectra rather than at the interferogram stage so that only a single set of unapodized flux calibration products is required:

```python
# For the extended calibrated data
extended_apod = apodizeSpectra(extended.copy(), apodName="aNB_15")
# For the point source calibrated data
pointSourceSds_apod = apodizeSpectra(pointSourceSds.copy(), apodName="aNB_15")
```

The default apodizing function is the adjusted Norton-Beer 1.5 function (aNB_15), which leads to a loss in spectral resolution by a factor of 1.5. It is also possible to change the value for the keyword apodName to apply a different apodizing function, however aNB_15 is a good compromise between loss of spectral resolution and a reduction in the side lobes. See the SPIRE Pipeline Specification Manual in SPIRE Pipeline Specification Manual for more details of the other available apodizing functions. Figure 7.34 shows the difference between the line shape for data before and after applying the standard apodization function. Applying a different apodization function would change the characteristic apodized shape. Note that if apodized lines are fitted by a Gaussian function, the line flux derived from the fitted parameters will overestimate the true flux from the source by around 5% (Hopwood et al. 2015). For the most accurate line measurements it is recommended to fit lines in the standard (non-apodized) data with sinc functions.
7.3.29. Correct the frequency axis to LSR

The frequency scale of the spectrum is corrected from the satellite frame to the Local Standard of Rest (LSR) using the velocity of the satellite, which is recorded in the metadata. For the single pointing pipeline, the task is run on both extended calibrated and point-source calibrated Level-2 products. For the mapping pipeline, it is run on the extended calibrated spectra before the pre-processed cubes are created. The task interpolates the data back onto the original frequency grid after the shift so that the frequencies of each sample do not change.

```python
# Correct the frequency scale to be in the Local Standard of Rest
extended = applyRadialVelocity(extended, targetFrame="lsr")
pointSourceSds = applyRadialVelocity(pointSourceSds, targetFrame="lsr")
# and for apodized data
extended_apod = applyRadialVelocity(extended_apod, targetFrame="lsr")
pointSourceSds_apod = applyRadialVelocity(pointSourceSds_apod, targetFrame="lsr")
```

7.3.30. Sort the meta data order

The metadata of the products are then sorted into a logical order using the `metaDataSorter` task:

```python
extended = metaDataSorter(extended)
pointSourceSds = metaDataSorter(pointSourceSds)
# and for apodized data
extended_apod = metaDataSorter(extended_apod)
pointSourceSds_apod = metaDataSorter(pointSourceSds_apod)
```

For the mapping script, the metadata is sorted again after the extended calibrated spectra are gridded into spectral cubes.

7.3.31. Apply the extended calibration correction

The extended calibrated spectra need to be corrected for the far-field coupling efficiency of the FTS feedhorns. This correction was introduced in HIPE version 14, after a comparison of extended calibrated FTS spectra with SPIRE photometer maps identified a factor of 1.4-1.7 difference. The far-
field (or extended-source) feedhorn coupling efficiency, as derived in Wu et al. (2013), is shown in Figure 7.35 in comparison to SLW ground measurements from Chattopadhyay et al. (2003) (green circles) and ratios of photometer surface brightness to spectrometer synthetic intensity in the three photometer bands (blue squares). The ratios shown are the average for 24 spatially flat homogenous sources (see Valtchanov et al. 2016 for details). Applying the correction brings the spectrometer and photometer extended calibrations into agreement, within the uncertainties of the Planck derived zero offsets (~10%) and within the uncertainties associated to the far-field coupling efficiency.

Note that this correction is not applied to the point-source calibrated spectra, as the point-source conversion factor is based on a model and observed data of Uranus, and hence invariant with respect to the extended source calibration scheme. The extended-source calibration correction is applied to the extended-source calibrated spectra as:

```python
# Apply the extended calibration correction to the extended-calibrated spectra
extended = applyExtCalCorr(extended, specExtCorr=extCorr)
extended_apod = applyExtCalCorr(extended_apod, specExtCorr=extCorr)
```

For the mapping script, this correction is applied to the pre-processed cubes.

Figure 7.35. Comparison of the average photometer surface brightness to synthetic FTS photometry ratios for 24 spatially flat sources (see Valtchanov et al. 2016). The blue symbols are the median ratio of SPIRE photometer vs spectrometer. The error bars include the 10% uncertainty of the Planck zero offset and the median absolute deviation for the ratio of the 24 flat sources. The green circles are ground based measurements from Chattopadhyay et al. (2003). The far-field feedhorn coupling efficiency is shown for the two FTS frequency bands (dashed red lines). The SSW level is increased by 10% with respect to Wu et al. (2013), to minimise discontinuity in the overlap region for extended calibrated spectra and to better match with the photometer.

### 7.3.32. Add overlapping photometer observation IDs

Two functions can be found at the top of each pipeline script: getPhotObsidsForFts and addMultiObsMeta. getPhotObsidsForFts restores a python dictionary from the HIPE save file photObsidsForFts.ser (provided within the HIPE build), which contains pre-matched overlapping photometer observations for each FTS observation. These matches include SPIRE/PACS parallel observations. If there are any photometer observations available, addMultiObsMeta adds the associated observation ID(s) (obsid(s)) to bottom of the extended- and point-source calibrated product metadata. The obsids are added as photObsid###, where “###” is the counter, which starts from "000". If there are no matching photometer obsids, then no photObsid###s are added to the metadata.
For the single pointing script, the following code adds any available photometer obsids to the extended and point-source calibrated products:

```python
# Check to see if there are any photometer observations that overlap
# with this observation and if so, append their obsids to the metadata
obsList = getPhotObsidsForFts(obs.obsid)
if obsList!=None:
    #
    # for l2ProdRef in obs.level2.refs:
    addMultiObsMeta(extended.meta, obsList)
    addMultiObsMeta(pointSourceSds.meta, obsList)
    addMultiObsMeta(extended_apod.meta, obsList)
    addMultiObsMeta(pointSourceSds_apod.meta, obsList)
    pass
    pass
```

For the mapping script, `addMultiObsMeta` is similarly applied, but to the `preCubes` and `cubes`.

### 7.3.33. Save the Level-2 products as FITS

In the single pointing script, the following lines store the extended and point source calibrated spectral products into FITS files for further analysis:

```python
# Save the final spectra to FITS (both extended and point source calibrated):
simpleFitsWriter(extended, "%s%i_%s_spectrum_extended.fits"%
    % (outDir, myObsid, extended.processResolution))
simpleFitsWriter(pointSourceSds, "%s%i_%s_spectrum_point.fits"
    % (outDir, myObsid, pointSourceSds.processResolution))
# and if required, save the apodized data
if apodize:
    simpleFitsWriter(extended_apod, "%s%i_%s_spectrum_extended_apod.fits"
        % (outDir, myObsid, extended_apod.processResolution))
    simpleFitsWriter(pointSourceSds_apod, "%s%i_%s_spectrum_point_apod.fits"
        % (outDir, myObsid, pointSourceSds_apod.processResolution))
```

Right at the end of the mapping script, similar commands are used to save the `preCubes` and `cubes` products to disk as FITS files:

```python
# Save the preCube and cubes to FITS:
simpleFitsWriter(preCube, "%s%i_%s_%s_spectrum2d.fits"%(outDir, myObsid, 
    res, array))
simpleFitsWriter(cube, "%s%i_%s_%s_cube.fits"%(outDir, myObsid, 
    res, array))
simpleFitsWriter(cube_convol, "%s%i_%s_%s_cube_convol.fits"%(outDir, myObsid, 
    res, array))
# and if required, save the apodized data
if apodize:
    simpleFitsWriter(preCube_apod, "%s%i_%s_%s_spectrum2d_apod.fits"%(outDir, 
        myObsid, res, array))
    simpleFitsWriter(cube_apod, "%s%i_%s_%s_cube_apod.fits"%(outDir, myObsid, 
        res, array))
    simpleFitsWriter(cube_convol_apod, "%s%i_%s_%s_cube_convol_apod.fits"%(outDir, 
        myObsid, res, array))
```

### 7.3.34. Create spectral cubes (mapping script)

An additional processing step is performed when creating spectral maps, where the individual spectra (per detector array) are projected onto a spatial grid that is equidistant in both spatial dimensions. The standard pipeline produces two sets of cubes. One uses the same "naive" mapping strategy as the SPIRE photometer to calculate the arithmetic mean of all spectra that are located within a given grid square. The other uses the "convolution" projection algorithm, which for each grid square, sums the Gaussian weighted contribution from spectra that fall within the kernel. The pixel size is defined at the beginning of the script by the `gridSpacing` parameter, see Section 7.3.4. The default pixel sizes are half the beam FWHM for full spatial sampling and the beam FWHM for intermediate spatial...
sampling. Reducing the pixel size will lead to more “holes” in the “naive” gridded cubes, as the number of grid squares that do not contain at least one spectrum is increased.

For each detector array, the first step in producing the spectral map is to collect the individual SDS products for each raster and jiggle position into a list of spectra and positions. This product is called the preCube and uses a Spectrum2d format, which is similar to HIFI products. The grid spacing and the preCube WCS are used to setup an optimised grid, centred on the source coordinates. Note that the WCS object created includes information on the spectral axes as well as spatial axes. The spireProjection task is then used to project the spectra from the preCube onto the grid specified by the WCS object. Finally, the metadata are sorted into a more logical order.

```python
mapSampling = obs.meta['mapSampling'].value
for array in ['SSW', 'SLW']:
    if array=='SLW':
        beamDiamDet='SLWC3'
    if array=='SSW':
        beamDiamDet='SSWD4'
    # -----------------------------------------------------------
    # Create a listing of all spectra and positions in a spectrum2d
    preCube = spirePreprocessCube(context=sdsList, arrayType=array, unvignetted=False)
    preCube_apod = spirePreprocessCube(context=sdsList_apod, arrayType=array, unvignetted=False)
    # -----------------------------------------------------------
    # Apply the extended calibration correction to extended spectra
    preCube = applyExtCalCorr(preCube, specExtCorr=extCorr)
    preCube_apod = applyExtCalCorr(preCube_apod, specExtCorr=extCorr)
    # -----------------------------------------------------------
    # Set up the grid - covering the RA and Dec of observed points using
    # specified gridSpacing:
    wcs = SpecWcsCreator.createWcs(preCube, gridSpacing[mapSampling][array], gridSpacing[mapSampling][array])
    # -----------------------------------------------------------
    # If LR, remove the SLW vignetted detectors from cube
    if preCube.meta['processResolution'].value=='LR' and array=='SLW':
        preCubeCopy = preCube.copy()
        preCube_apodCopy = preCube_apod.copy()
        preCube.deleteSpectra(lrSlwChannelsToRemove)
        preCube_apod.deleteSpectra(lrSlwChannelsToRemove)
    pass
    # -----------------------------------------------------------
    # Regrid the data using the Naive Projection algorithm:
    cube = spireProjection(spc=preCube, wcs=wcs, projectionType='naive')
    cube_apod = spireProjection(spc=preCube_apod, wcs=wcs, projectionType='naive')
    # -----------------------------------------------------------
    # Regrid the data using the Convolution algorithm:
    beamParam = obs.calibration.spec.beamParamList.getProduct(res, preCube.startDate)
    beamDiam = beamParam[0][beamDiamDet]['beamDiam'].data
    cube_convol = spireProjection(spc=preCube, wcs=wcs, projectionType='convolution', beamWidthArray=beamDiam)
    cube_convol_apod = spireProjection(spc=preCube_apod, wcs=wcs, projectionType='convolution', beamWidthArray=beamDiam)
    # -----------------------------------------------------------
    # Sort the metadata into a logical order
    cube = metaDataSorter(cube)
    cube_apod = metaDataSorter(cube_apod)
    cube_convol = metaDataSorter(cube_convol)
    cube_convol_apod = metaDataSorter(cube_convol_apod)
    # -----------------------------------------------------------
    # Check to see if there are any photometer observations that overlap
    # with this observation and if so, append their obsids to the metadata
    obsList = getPhotObsidsForFts(obs.obsid)
    # -----------------------------------------------------------
    # If LR, save the full preCubes
    if processRes=='LR' and array=='SLW':
        preCube = preCubeCopy.copy()
        preCube_apod = preCube_apodCopy.copy()
    if obsList!=None:
        # ...
```
for l2ProdRef in obs.level2.refs:
    addMultiObsMeta(preCube.meta, obsList)
    addMultiObsMeta(cube.meta, obsList)
    addMultiObsMeta(cube_convol.meta, obsList)
    addMultiObsMeta(preCube_apod.meta, obsList)
    addMultiObsMeta(cube_apod.meta, obsList)
    addMultiObsMeta(cube_convol_apod.meta, obsList)
    pass
    addMultiObsMeta(obs.meta, obsList)
    pass
# -----------------------------------------------------------
# Tweak the cube variable name to specify the array:
exec("cube%s = cube"%array)
exec("cube%s_apod = cube_apod"%array)
exec("cube%s_convol = cube_convol"%array)
exec("cube%s_convol_apod = cube_convol_apod"%array)

7.3.35. Update and store the Observation Context

The script will attempt to update the Observation Context only if Level-1 and -2 already exist.

In order to save the products back into the Observation Context, the structure needs to be taken into account. A reminder of the Spectrometer Observation Context layout for the Level-1 and Level-2, and how this relates to variables in the user pipeline scripts is given in Table 7.4, where x is the raster point position (starting at zero), y is the jiggle position (starting at zero) and z is the resolution (C, H, M, or L).

Table 7.4. Summary of Spectrometer Observation Context

<table>
<thead>
<tr>
<th>Level</th>
<th>Sub-context name</th>
<th>Sub-Sub-context name</th>
<th>Product Type</th>
<th>variable in User Script</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 1</td>
<td>&quot;Point_x_Jiggle_y_zR&quot;</td>
<td>&quot;interferogram&quot;</td>
<td>SDI</td>
<td>sdi</td>
</tr>
<tr>
<td>Level 2</td>
<td>&quot;zR_spectrum_ext&quot;</td>
<td></td>
<td>SDS</td>
<td>extended</td>
</tr>
<tr>
<td></td>
<td>&quot;zR_spectrum_ext_apod&quot;</td>
<td></td>
<td>SDS</td>
<td>extended_apod</td>
</tr>
<tr>
<td></td>
<td>&quot;zR_spectrum_point&quot;</td>
<td>SPSS</td>
<td>SPSS</td>
<td>pointSourceSds</td>
</tr>
<tr>
<td></td>
<td>&quot;zR_spectrum_point_apod&quot;</td>
<td>SPSS</td>
<td>SPSS</td>
<td>pointSourceSds_apod</td>
</tr>
<tr>
<td></td>
<td>&quot;zR_SLW_spectrum2d&quot;</td>
<td>SPC</td>
<td>SPC</td>
<td>preCube</td>
</tr>
<tr>
<td></td>
<td>&quot;zR_SLW_spectrum2d_apod&quot;</td>
<td>SPC</td>
<td>SPC</td>
<td>preCube_apod</td>
</tr>
<tr>
<td></td>
<td>&quot;zR_SLW_cube&quot;</td>
<td>SpectralSimpleCube</td>
<td>cubeSLW</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;zR_SLW_cube_convol&quot;</td>
<td>SpectralSimpleCube</td>
<td>cubeSLW_convol</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;zR_SLW_cube_apod&quot;</td>
<td>SpectralSimpleCube</td>
<td>cubeSLW_apod</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;zR_SLW_cube_convol_apod&quot;</td>
<td>SpectralSimpleCube</td>
<td>cubeSLW_convol_apod</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;zR_SSW_spectrum2d&quot;</td>
<td>SPC</td>
<td>SPC</td>
<td>preCube</td>
</tr>
<tr>
<td></td>
<td>&quot;zR_SSW_spectrum2d_apod&quot;</td>
<td>SPC</td>
<td>SPC</td>
<td>preCube_apod</td>
</tr>
<tr>
<td></td>
<td>&quot;zR_SSW_cube&quot;</td>
<td>SpectralSimpleCube</td>
<td>cubeSSW</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;zR_SSW_cube_convol&quot;</td>
<td>SpectralSimpleCube</td>
<td>cubeSSW_convol</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;zR_SSW_cube_apod&quot;</td>
<td>SpectralSimpleCube</td>
<td>cubeSSW_apod</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;zR_SSW_cube_convol_apod&quot;</td>
<td>SpectralSimpleCube</td>
<td>cubeSSW_convol_apod</td>
<td></td>
</tr>
</tbody>
</table>
For the point source script the products are saved back into the Observation Context with:

```python
if obs.level1 and obs.level2:
    res = pointSourceSds.processResolution
    # Save the products back into the right places inside the Observation Context
    obs.level1.getProduct("Point_0_Jiggle_0_%s"%res).setProduct("interferogram", sdi)
    obs.level2.setProduct("%s_spectrum_ext"%res, extended)
    obs.level2.setProduct("%s_spectrum_point"%res, pointSourceSds)
    obs.level2.setProduct("%s_spectrum_ext_apod"%res, extended_apod)
    obs.level2.setProduct("%s_spectrum_point_apod"%res, pointSourceSds_apod)
```

For the mapping script the interferograms are saved back into the Observation Context at the end of the loop over building blocks. The Level-2 Context is updated before the data is saved to FITS files:

```python
if obs.level2:
    res = preCube.meta["processResolution"].value
    # Save the preCube
    obs.level2.setProduct("%s_%s_spectrum2d"%(res, array), preCube)
    obs.level2.setProduct("%s_%s_spectrum2d_apod"%(res, array), preCube_apod)
    # Save the cube (removing old style products if they exist)
    obs.level2.refs.remove("%s_%s_unapodized_spectrum"%(res, array))
    obs.level2.refs.remove("%s_%s_apodized_spectrum"%(res, array))
    obs.level2.setProduct("%s_%s_cube"%(res, array), cube)
    obs.level2.setProduct("%s_%s_cube_convol"%(res, array), cube_convol)
    obs.level2.setProduct("%s_%s_cube_apod"%(res, array), cube_apod)
    obs.level2.setProduct("%s_%s_cube_convol_apod"%(res, array), cube_convol_apod)
```

For both scripts, it is necessary to uncomment the following line in order to commit the Observation Context to hard disk:

```python
#saveObservation(obs, poolName="enter-a-poolname", saveCalTree=True)
```

If the Observation Context is saved back into the same pool it was loaded from, the default behaviour is to create new versions of the products that have changed, rather than overwriting. Only products that have changed will actually be saved to disk as new versions. It is possible to overwrite rather than creating new versions by modifying the HIPE property hcss.ia.pal.pool.lstore.version. The possible values and the effect they have on the files that are saved to disk are:

- **new**: create a new version by appending a timestamp to the name (default)
- **overwrite**: overwrite the existing file with the same name; no timestamp is added
- **error**: produce an error if a file with the same name already exists; no timestamp is added

This property can be set initially in the `hipe.props` file (located in your .hcss directory). Once set, `hcss.ia.pal.pool.lstore.version` can be modified in the properties panel inside HIPE (go to `Edit > Preferences > Advanced`). It is also possible to modify the behaviour separately for individual pools. This can be specified with property names of the form, `hcss.ia.pal.pool.lstore.<pool-name>.version`.

Note that it can be dangerous to change this property away from the default value because it affects all operations that write data to pools. For example, the photometer SPG pipeline writes out temporary files and will go wrong if the property is set to **overwrite**. Changing to overwrite files should be done with great care.

## 7.4. Pointing Considerations

The *Herschel* telescope pointing information is stored in an auxiliary product, which is used by the pipelines in order to calculate the final pointing of each SPIRE detector. This calculation involves correcting the telescope boresight to the instrument “aperture”, and then taking account of the position of the SPIRE beam steering mirror (BSM) and the offset of each detector within the array. The instrument apertures are defined relative to the telescope boresight in the Spacecraft Instrument Alignment
Matrix (SIAM), which is also stored in an auxiliary product in the Observation Context. The most commonly used instrument apertures were "S24" (the centre of the SSW array) and "S14" (the centre of the photometer PSW array). Figure 7.36 illustrates the way that the final pointing for each detector is built up.

Figure 7.36. The different elements involved in calculating the pointing of each detector. The array shown is SSW.

The pointing accuracy is particularly important for observations of point sources that used the FTS sparse mode. Offsets greater than 5'' can lead to a 10-20% frequency dependent apparent flux loss in the SSW band where the beam is smallest (see Valtchanov et al. 2014). The pointing effects have significantly less of an impact for the SLW band, due to the relatively larger beam size. For mapping observations, the pixel size over which individual pointings are averaged is generally larger than any pointing offset, and so the effects are less noticeable.

The Herschel pointing performance and the a posteriori improvements to the pointing products are explained in greater detail in Sanchez-Portal et al. (2014). These improvements have been put into practice in the pipeline from HIPE v11.1.0 onwards, and so the Herschel pointing products attached to Observation Contexts produced using this HIPE version (or later) contain the best current knowledge of the telescope position. The uncertainty on the telescope position contained within the Herschel pointing product is referred to as the Absolute Pointing Error (APE).

Table 7.5 summarises the metadata that describe the RA and Dec pointing information and where these are located in the Observation Context and data products.

Table 7.5. Summary of pointing metadata.

<table>
<thead>
<tr>
<th>Metadata name</th>
<th>Metadata location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>posAngle</td>
<td>Observation Context</td>
<td>The +Z axis position angle of the Herschel focal plane during the observation, as measured East of North. See the SPIRE Handbook for more details.</td>
</tr>
<tr>
<td>raNominal, decNominal</td>
<td>Observation Context</td>
<td>The nominal (commanded) sky coordinates in degrees. These are the user requested target coordi-</td>
</tr>
</tbody>
</table>
## SPIRE BSM Position

The Beam Steering Mirror (BSM) is a movable mirror inside SPIRE that changes the angular position of the centre of the beam with respect to the detector arrays. The BSM is used to step the beam around the array for jiggled observations. Its position is measured in ADU along the Spacecraft axes and converted to arcseconds on the sky using a calibration curve stored in the BsmPos calibration product. Normally the BSM is kept at its rest position and should only be moved to create the different joggle positions for a mapping observation. Mapping observations were made using the correct BSM position for all operational days (ODs) and joggle positions, however for sparse observations made prior to OD 1011, the BSM was slightly offset by 1.72". The commanded BSM position was corrected back to the centre of the beam for sparse mode from OD 1011 onwards.

---

### Table: Metadata Information

<table>
<thead>
<tr>
<th>Metadata name</th>
<th>Metadata location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product top level metadata</td>
<td>nates, which were entered into HSpot for the observing request. For moving targets, such as solar-system objects, they are calculated for the epoch of the observation using Horizons, with the value corresponding to the start time of the observation.</td>
<td></td>
</tr>
<tr>
<td>ra, dec</td>
<td>Multiple, see a-d below</td>
<td>The average sky coordinates during the observation, in degrees. These are calculated as the average of the RA and Dec of the telescope pointing during the observation. The calculation excludes the periods of slewing to the target position but does include telescope movements during the observation (e.g. during a raster map).</td>
</tr>
<tr>
<td>a)</td>
<td>Observation Context</td>
<td>As above, but for sparse observations these are copied from the ra and dec of the SSWD4 detector. For mapping observations, these are the map centre coordinates.</td>
</tr>
<tr>
<td>b)</td>
<td>Individual detector tables in Level-1 products</td>
<td>As above, but taking account of the BSM position and the relative detector offset from the array centre (at SSWD4). Each scan for each detector in the Level-1 interferograms has its own scan-averaged ra and dec metadata.</td>
</tr>
<tr>
<td>c)</td>
<td>Individual detector tables in Level-2 sparse products</td>
<td>As above, but whereas the Level-1 interferograms has its own scan-averaged ra and dec metadata, the ra and dec for each detector in Level-2 have been averaged over all scans.</td>
</tr>
<tr>
<td>d)</td>
<td>Level-2 pre-processed cubes and spectral cubes</td>
<td>The Observation Context ra and dec for mapping observations are propagated to the pre-processed cube and spectral cube Level-2 products.</td>
</tr>
<tr>
<td>raDecOffset</td>
<td>Observation Context</td>
<td>The distance in arcseconds between the commanded position (raNominal, decNominal) and the actual reconstructed pointing (ra, dec), which includes any systematic BSM offset (bsmOffset), but not the APE. (FITS keyword: RADECOFF)</td>
</tr>
<tr>
<td>bsmOffset</td>
<td>Observation Context</td>
<td>BSM offset position (0.0 or 1.72&quot;), which is corrected for by the point-source calibration. This systematic offset is included in the raDecOffset value. (FITS keyword: BSMOFF)</td>
</tr>
</tbody>
</table>

Note that the ra and dec coordinates from the telescope pointing only include correction for source proper motion if the observer provided the necessary information in the observing request. The observer-provided target coordinates raNominal, decNominal, however, do not include this correction. Consequently, for those handful of sources with significant proper motion, the calculated difference (provided by raDecOffset) may not be correct.

---

7.4.1. SPIRE BSM Position
Several of the point-source calibration products applied in the pipeline are split into two epochs to take proper account of the two BSM positions (Swinyard et al. 2014). For sparse observations before OD 1011, the point-source calibration accounts for the 1.72” BSM offset, so that the final calibrated flux density is correct, within the APE. Figure 7.37 illustrates the uncertainty in pointing due to the APE for both sparse mode BSM epochs.

![Figure 7.37](image)

**Figure 7.37.** Comparison of the 2σ APE distribution, shown as a blue cloud, before (left) and after (right) OD 1011.

### 7.4.2. Correcting the pointing

It is important to note that all *Herschel* pointings are subject to the APE, which follows a random error distribution with a 2σ spread of 1-2 arcseconds (Sanchez-Portal et al. 2014), as shown by Figure 7.37. Therefore, even though the difference between the average Level-2 RA and Dec and the nominal commanded RA and Dec is known and corrected for, there remains a pointing uncertainty of the APE.

If an object was observed frequently during the mission, pointing accuracy can be inferred in relative terms by comparing each spectrum against the one showing the highest flux level (Valtchanov et al. 2014). The more observations available for a single object, the more likely that the reference spectrum (with the highest flux) was observed close to the true source position, and thus the better the relative pointing offset will be. This relative method can be used to correct for both the known offset to the nominal pointing (reported by raDecOffset) and the APE.

Uranus is used for the FTS absolute flux calibration. More than 30 Uranus observations were performed throughout the *Herschel* mission, allowing their relative pointing offset to be derived (as detailed in Valtchanov et al. 2014). Therefore the Uranus observations used for the calibration are first corrected using these values. Note that the model spectrum was required when estimating the relative pointing for Uranus, as the expected source flux varies with distance to the planet. Sparse mode data observed before OD 1011 (when the BSM was offset from its rest position by 1.72”) are calibrated with Uranus data from before OD 1011, and likewise, data from after OD 1011 are calibrated using Uranus data from the same OD epoch. This means that the point-source flux calibration is correct for all observations, regardless of the OD they were made on.

Unlike the primary or the secondary calibrators, most targets were only observed once, and so it is not generally possible to accurately assess the overall pointing including the APE contribution. In particular cases when the brightness distribution model for a source is well known or the source is point-like, one can use the HIPE useful *Spectrometer Pointing Offset Corrector* script to infer and correct pointing offset. This script can be accessed in HIPE from the Scripts > *SPIRE Useful Scripts* menu.

The *Spectrometer Pointing Offset Corrector* script uses the semi-extended correction tool (SECT, see Section 7.6.2) to iterate over a range of pointing offsets until the SLW and SSW spectra in the SLW/SSW overlap region match. An example observation of Uranus (observation ID 1342259588) with a known pointing offset of 3.4” +/- 0.2” is included in the script. This offset was derived using the relative method of Valtchanov et al. 2014. The basic steps taken by the script are as follows:
1. the median SLW/SSW ratio is calculated using the overlapping region [959.3, 989.4] GHz

2. if the input model is not a point source, the input data is corrected for partial extent using SECT and the overlap ratio retaken

3. a grid of pointing offsets is generated using SECT, along with a corresponding grid of normalised overlap ratios

4. the final pointing offset is interpolated from the grids using the calculated overlap ratio (see Figure 7.38)

5. the ratio of the corrected spectrum is checked

The input spectra and the pointing corrected output are compared in the right panel of Figure 7.38.

Figure 7.38. Plots produced by the Pointing Offset Corrector script for the observation of Uranus given in the script. On the left, the grid of pointing offset is plotted as a function of overlap ratio. The green square marks the final pointing offset. On the right, the input data is compared to the data corrected, output by the script.

If the source brightness distribution model is known and the pointing offset is also known, then SECT can be used directly to correct the data. Note that the Observation Context or Level-2 metadata raDecOffset (as described in Table 7.5) gives an indication of the pointing offset, although this does not include the statistically random APE. Nevertheless, if raDecOffset reports a systematic pointing offset that is significant compared to the APE, it may be worthwhile correcting the spectra using this value.

Observations with pointing offsets larger than 5'' are likely to be significantly affected by flux loss due to this mispointing. Correcting them for pointing offset will further decrease the signal-to-noise of already noisy spectra, as the correction is a multiplicative function. It is better to avoid using these data. Alternative data is usually available for such observations, as during the active phase of the Herschel mission, significantly mispointed observations were re-observed.

Observations with an raDecOffset larger than 4'' have a particular quality control summary comment in the HSA Quality Control block, as well as in the Observation Context quality Control Context. These will report ”Observation with a potentially significant pointing offset”. Such observations should be treated with caution and whether to apply a pointing offset correction should be evaluated by considering the caveats discussed in this section.

7.5. Recipes for faint and medium-strength sources

For the FTS, faint sources are those with continua of up to 10 Jy and medium-strength sources are those with continua up to 100 Jy. An example of a faint source is the ultra luminous infrared galaxy
Mrk 231, which was observed on operational day (OD) 209 (observation ID 1342187893) and has a maximum continuum level of 10 Jy in SSW.

When processing faint to medium-strength sources, the following post-pipeline steps are important to consider:

- Comparing point-source and extended-source calibration (Section 7.5.1; Section 7.6)
- Optimising background subtraction (Section 7.5.2)
- Checking spectral noise with respect to the expected HSpot values (Section 7.5.3)
- Comparing with the SPIRE Photometer (Section 7.5.4; Section 7.10)

These points are described in detail in the sections indicated.

### 7.5.1. Comparing point-source and extended-source calibration

An extended-source is defined as one that causes a flat illumination of the detectors and is much larger than the beam size of an individual detector. Whereas a source is considered point-like, if it is smaller than an SSW detector. The Level-2 Observation Context for sparse observations, contains two sets of spectra. One set calibrated assuming extended-source emission, using calibration based on an extended source (the telescope). The other set are calibrated for point-source emission, using calibration based on observations and a model of Uranus. The FTS calibration scheme is detailed in Swinyard et al. (2014). The most suitable calibration depends on the extension of the source being observed. If well-calibrated, the spectra from the centre detectors are expected to agree where the two bands overlap in frequency. However, between the two extremes of source type (point-like and extended), there is a range of source size, i.e. semi-extended sources. If a source falls within the semi-extended category, neither calibration may provide satisfactory results and the spectra may need to be corrected before they are used for scientific measurements. Section 7.6 provides information on how to diagnose issues, such as source-extent, and how these issues can be corrected with further processing.

### 7.5.2. Optimising background subtraction

Telescope emission is the dominant contributing source for most FTS observations (see Swinyard et al. (2014)), and acts as a high background. The telescope contribution is subtracted by the standard pipeline using a model derived from the primary and secondary mirror temperatures, which were measured during each observation. The model generally reproduces the measured background spectrum to within 0.06%, however residual telescope emission can be significant for faint sources and manifest in the final spectrum as a discontinuity between the two bands and a distortion of the overall spectral shape. There is also contribution from instrument emission, which starts to dominate at the lower end of the SLW frequency band. The pipeline also subtracts instrument emission using a model. The average absolute (additive) uncertainty associated to the telescope and instrument subtraction is ~0.3-0.4Jy (Hopwood et al. 2015). However, as the telescope and instrument act as black body emitters, of different temperatures, therefore the "continuum offset" is also frequency dependant. Figure 7.39 shows the point-source calibrated continuum offset for the centre detectors, over the full range of the FTS frequency bands. There are a number of key features that can be noted. The offset for SSW is relatively flat, due to negligible instrument emission and a smaller beam size compared to SLW. For SLW there is a steep increase in the offset below 700 GHz, where instrument emission starts to dominate. There is also a steep increase above 700 GHz for SLW, as the beam size increases with frequency. Details on the FTS beam are given in Section 7.6.1. The continuum offset rises sharply at the edges of both frequency bands. These bands were widened as of HIPE version 12.1, and are known to contain high levels of systematic noise, but were included as spectral lines can be present and are useable in these regions. See Hopwood et al. (2015) for an example of extracting a line from the band edges. For full details on FTS calibration accuracy, see Swinyard et al. (2014) and Hopwood et al. (2015), with this information summarised in the SPIRE Handbook.
Figure 7.39. The point-source calibrated continuum offset, which is a systematic additive uncertainty associated to residual telescope and instrument emission. The offset is relatively flat in SSW, away from the noisy band edges, whereas SLW sees a steep increase above and below 700 GHz. The difference between the two frequency bands arises as contribution from the instrument is only significant below 600 GHz and telescope emission is more significant for SLW, due to a larger beam size compared to SSW.

Figure 7.40. Four point-source calibrated spectra, processed using the standard HIPE pipeline (SLW in red, SSW in blue) in comparison to the dark sky observation taken on the same operational day (black). The source brightness increases from faint to medium through observations (A) to (D). Source (A) is compared to a reduction with HIPE version 10 (grey) to illustrate the improvement from HIPE version 11 (red and blue).

Figure 7.40 shows the standard pipeline output for four observations of point-sources that span a range of brightness. The plot to the top left (labelled A) shows the observation of Mrk 231 from OD 209, which was detailed at the beginning of the section. There are further plots using data from these observation in figures Figure 7.40, Figure 7.42 and Figure 7.45, which can be reproduced by following the analysis steps detailed in this section. The general characteristics of the four sources are:
• (A) a faint observation with continuum less than 10 Jy (Mrk 231 from OD 209)

• (B) a featureless observation (in this case an asteroid) with continuum up to 45 Jy

• (C) a source of similar brightness to (B), but exhibiting strong CO lines and embedded in an extended background, thus showing a jump between the bands that is significant in comparison to the expected continuum offset uncertainty

• (D) a medium-brightness source with many spectral features

The pipeline processed point-source calibrated spectrum for the the faintest source, example (A), shows a good agreement between the bands and a smooth low frequency spectral shape. Although observations (B) and (C) are brighter than (A), they both exhibit a discontinuity between the two bands. Systematic effects are negligible for (D), which is the brightest of the four examples. In general, the fainter the source, the more likely the related pipeline products will require additional processing.

The corresponding dark sky observations for sources (A) to (D) are also plotted in Figure 7.40. An observation of dark sky was taken for almost every OD that the spectrometer was in use. The details of each sparse-mode dark sky measurement, including the thermal conditions on the OD in question, can be found at the SPIRE Instrument and Calibration web pages http://herschel.esac.esa.int/twiki/bin/view/Public/SpireDailyDarkObservations. All dark sky observations are are all publicly available in the Herschel Science Archive.

When a jump is present between the two bands and cannot be assigned to the expected uncertainty on the continuum due to residual telescope emission (e.g. panel C in Figure 7.40), see Section 7.6 to help diagnose the potential cause. However, if the observed target is known to be point-like, then it can be assumed that extended background emission is causing the jump. Inside HIPE, there is a useful script to correct residual background emission, either from the telescope and instrument or high background. The “Spectrometer Background Subtraction” script is provided in the Scripts menu on the HIPE toolbar. The script provides two methods that can be applied to correct the spectral shape using:

1. The surrounding detectors in the array. Note that this approach is only appropriate for observations of point-like sources.

2. A dark sky observation taken on the same OD as the science observation.

For a given observation of a point-source, method 1 involves subtracting an average of the off-axis detectors from the respective centre detectors. The off-axis spectra are smoothed with a wide Gaussian kernel to leave only the large-scale information, and can either be visually inspected to reject any outliers (i.e. those having a significantly different flux level or spectral shape compared to the others) or outliers can be automatically selected. Examples of spectra that have been automatically selected as outliers by the script can be found in Figure 7.41, but note that this selection becomes more unreliable the lower the signal-to-noise ratio of the data.

![Figure 7.41. Plots produced at the end of the HIPE “Background Subtraction” script for example source (A). They show the smoothed off-axis detectors (black) and the mean used for subtraction (dashed blue) for both SLW (left) and SSW (right). The red curves for SSW show outliers that were excluded from the mean.](image)
Subtraction of the off-axis detectors will also subtract any extended astronomical background emission in the nearby region of the source. The "Spectrometer Array Footprint Plot" useful script can be used to check the background for extended emission or neighbouring sources. This script plots the footprint of the FTS detectors onto a photometer image (see Figure 7.42). If no photometer image is supplied, the metadata of the input FTS product is checked for overlapping SPIRE photometer observations, and if present, the first (photObsid000) is used to download a map from the Herschel Science Archive. Figure 7.42 gives an example of the FTS footprint plots for sources (A) and (C), which are overlaid on corresponding SPIRE photometer maps. This figure shows (A) is point-like with no nearby sources whereas observation (C), although also point-like, is embedded in uneven cirrus with a close neighbouring source within the FTS footprint. For cases such as (C), a more careful selection of the off-axis detectors may be required before subtraction. Using a FIR/sub-millimetre photometer image is ideal to check for emission the FTS is sensitive to.

Even for a well pointed observation of a true point-source, a fraction of source signal falls on the off-axis detectors (as detailed in Hopwood et al. (2015)). Therefore correcting the spectral shape with a subtraction of the off-axis detectors also results in subtracting some real flux. The first ring of SLW is used by the script and for a point-like source these detectors contain around 2% of the total source flux. The significance of subtracting this flux should be considered against other sources of error, e.g. whether it is negligible compared to the random noise in an observation of a faint source, or to the systematic noise for a point-source sitting in an extended background. For SSW, the second ring of detectors is used, which contains only 0.1% of the target source flux.

In addition to the off-axis subtraction, if electing to subtract a spectrum of dark sky (by setting subtractDark to True and providing a dark sky observation ID), the background subtraction script produces two corrected spectra and plots them against the un-corrected data for immediate visual comparison. Considering the following points may help to choose the best result (although this is often obvious by eye):

- Correcting with the smoothed and averaged off-axis detectors (performed by default) will not add any small scale random or systematic noise. Generally the average over several off-axis spectra provides a good approximation to the necessary large-scale shape required to correct the continuum. However, the combined spectrum should be checked by eye to see if it has a satisfactory shape.

- If a suitable dark sky observation is used for subtraction, it is likely to only be as long as the science observation being corrected. So in most cases this subtraction will lead to an increase in the small-
scale spectral noise. However, there are some examples of observations on particular operational
days that contain higher than average correlated noise and subtracting like-for-like detectors using
a dark observation may remove a proportion of this extra detector-dependent noise.

An additional way to check the results is to measure the spectral noise, and this is discussed in Section 7.5.3. The Noise Estimate useful script can be run on the background subtracted results output by the Background Subtraction useful script.

Usually, the dark sky observation from the same OD as the science observation to be corrected is the most suitable for subtraction. However, if the dark sky observation from the same day is not as deep as the science observation, it may be possible to find a deeper dark sky reference with similar telescope temperatures from another OD. The table on the SPIRE Instrument and Calibration web pages shows the number of repetitions used for each sparse dark sky observation taken.

Figure 7.43. Plots resulting from running the HIPE "Spectrometer Background Subtraction" script for the four example sources. The original pipeline spectrum is shown in red, the dark sky subtracted spectrum in black and the off-axis detector subtracted spectrum in green.

Figure 7.43 shows the results of applying the two background subtraction methods described to observations (A) - (D). For (A), there is a clear improvement in spectral shape and agreement between the bands after subtracting the combined off-axis detectors, whereas subtracting the dark sky observation from the same operational day does not improve the shape or noise (see Figure 7.45 in Section 7.5.3). The dark sky subtraction makes little difference to (B) and (C), whereas the off-axis subtraction corrects the step between the frequency bands. (D) is bright enough for background subtraction to be negligible. In addition to visually checking the off-axis detectors with a photometer map (if one is available) and the spectra directly in the background subtraction script, and checking the noise properties, a further comparison can be made with SPIRE photometer photometry points (if there is an associated map) Section 7.5.4. These example sources are point-like and therefore any distortion in the continuum is due to residual background. However this distortion does present similarly to that seen for a semi-extended source. See Section 7.6 to help determine if a source may be semi-extended and if so, how to apply appropriate post-pipeline processing.

There is a GUI version of the background subtraction script available in HIPE (spiaFts-BackgroundRemoval), which can be opened from the HIPE Tasks panel. spiaFtsBackgroundRemoval subtracts the background using the off-axis detectors or a dark sky observation, if
one is supplied. This task follows the same method as the companion useful script, although it takes a full Observation Context as input and updates that with the subtracted results, unless a copy is requested. The GUI also removes the off-axis detectors from the subtracted output and offers the option of fitting a polynomial function to each selected off-axis detector.

The `spiaFtsBackgroundRemoval` GUI layout is shown in Figure 7.44, where the inner and outer (vignetted) SSW detectors are separated into two lists. More information about each GUI field can be found in the tooltips, which can be viewed by hovering the cursor over each field.

Figure 7.44. The GUI layout for the HIPE task `spiaFtsBackgroundRemoval`. This task performs a background subtraction for sparse pointed observations and works with full Observation Contexts.

### 7.5.3. Checking Spectral Noise

There are three main sources of uncertainty associated with processed FTS spectra:

- **Random noise**
- **Large-to-mid-scale systematic noise** associated with the instrument
- **Small-scale systematic noise** associated with the observed signal, i.e. fringing and the instrumental line shape wings

Random noise tends to dominate observations of faint sources, which is reduced by integrating over a greater number of scans. For brighter sources and those with strong spectral features, systematic noise is usually the dominant contributor, and this does not reduce with an increase in integration time.

One way to check data processing is to examine the spectral noise achieved in the point-source calibrated spectrum and compare this to the expected values from HSpot. The "error" column attached to each observation only represents the random component of the overall uncertainty, as this is calculated as the standard error on the mean of the repeated scans. This "error" is generally lower than a direct estimate of the spectral noise, so to provide a more realistic estimate of the expected combined random and instrumental noise components, the sensitivity calculation used by HSpot was derived from observations of dark sky and sources that show no significant spectral features. Before an estimate of the noise can be compared to the HSpot prediction, all strong spectral features must be fitted and subtracted. Line removal can be performed using the Spectrometer Line Fitting useful script, but for line-rich sources, small-scale systematic noise may continue to dominate the residual. If the final spectral noise in a reduced spectrum (or the line subtracted residual) does not at least roughly agree with the values from HSpot then this may indicate a problem in the data or that the data processing was not optimal. However, if the estimated spectral noise agrees with or is better than the HSpot values, this strongly suggests the reduction is optimal in terms of the small-scale noise component. Note that due to calibration improvements, noise estimates for data of sources fainter than Mrk 231, which have been reduced with HIPE version 11 or later, tend to beat the HSpot prediction across the frequency bands.

The sensitivity in HSpot is calculated with respect to the in-beam flux density in Jy. So the noise levels in the two bands are similar, with the lowest noise at the centre of the bands. When data are processed
using the extended source calibration, the source is assumed to be uniformly extended in the beam and the spectrum is calculated in intensity units (W/m\(^2\)/Hz/sr). As the beam size varies significantly with frequency and between the two bands, the noise levels in the extended calibrated data appear far higher for SSW than for SLW, due to the smaller SSW beam size.

**Figure 7.45.** Noise estimates for observations (A) to (D) are shown for the reduced data (red) and the dark sky subtracted spectrum outputted from the Background Subtraction Script (black). All strong lines were subtracted for observations (A), (C) and (D) before assessing the noise. The noise for the off-axis detector subtracted spectrum, obtained from the same script and shown in Figure 7.43, is not included as the error measured sees no significant change after this subtraction. The "error" column for the reduced data is shown in grey and predicted noise from HSpot in blue. The blue dots are those taken directly from the HSpot tool, which it rounds to one decimal place. While the solid blue line represents the values given in the SPIRE Handbook, after they have been scaled.

The "Spectrometer Noise Estimate" useful script is available in the HIPE Scripts/SPIRE Useful scripts menu. Figure 7.45 was produced using the same approach in estimating spectral noise as this script, and shows a comparison of the continuum noise estimated over a range of frequency bins for observations (A) - (D). Noise estimates are plotted for the point-source calibrated pipeline product, the Background Subtraction script dark sky subtracted result, the "error" column for the un-subtracted data and the HSpot estimated sensitivity. Before taking these noise estimates, all strong spectral features were subtracted for sources (A), (C) and (D) using the HIPE "Spectrometer Line Fitting" useful script, which is described in Section 7.11.9.

Considering the faintest source (A), Figure 7.45 shows that after significant spectral lines are subtracted, the noise for the standard reduction is at least as good as the HSpot values. Subtracting the dark raises the noise levels and, as seen in Figure 7.43, distorts the spectral shape. The best continuum corrections and noise improvements come from subtracting an average of smoothed off-axis detectors.

Observations (B) and (C) consist of eight and ten scans, respectively, and show a good agreement between the random noise component (the "error" column) and the noise estimated in bins.

In general, random noise and continuum distortion tend to be the main post-pipeline problems for faint sources. For brighter sources that are rich in spectral features, such as D, even after fitting and subtracting many of the lines, the estimated noise is still contaminated by real signal.
The results of the post-pipeline processing applied to the four example observations can be summarised as follows:

- The off-axis subtraction is necessary for observation (A) to correct the spectral shape.
- Subtracting the corresponding dark of the day for (A) distorts the spectral shape and increases the noise levels.
- For observations (B) and (C), the off-axis subtraction corrects the wide-scale distortion.
- In contrast, for both (B) and (C) the dark sky subtraction does not noticeably improve the spectral shape or noise levels.
- Observation (D) is bright enough so that it does not require any post-pipeline correction.

There is a GUI version of the noise estimate script available in HIPE. It can be opened from the HIPE Tasks panel and is called `spiaFtsNoiseEstimate`. This task follows the same method as the Noise Estimate script, except the GUI version requires an input of the full Observation Context.

The `spiaFtsNoiseEstimate` GUI layout is shown in Figure 7.46. More information about each GUI field can be found in the tooltips, which are viewed by hovering the cursor over each field.

![Figure 7.46. The GUI layout for the spiaFtsNoiseEstimate task in HIPE.](image)

### 7.5.4. Comparing with the SPIRE Photometer

The point-source flux in the three bands of the SPIRE photometer can provide a useful piece of information about the accuracy of the flux calibration and should be in rough agreement with the spectrometer flux densities. See Section 7.10 for details on making such a comparison using synthetic FTS photometry taken at the photometer wavebands. For observations (C) and (D) the photometer points are in-line with the continuum level, see Figure 7.47. For (C), the background subtracted spectrum is used for the comparison, and suggests the off-axis subtraction is good for this source and the surrounding cirrus has been removed satisfactorily.
Figure 7.47. On the left, the off-axis subtracted spectrum is shown for observation (C) and the standard reduction for observation (D) is plotted on the right. The blue points are extracted directly from SPIRE photometer maps using aperture photometry and show there is a good agreement with the respective continuum.

7.6. Recipes for semi-extended sources

This section describes the Semi-Extended Correction Tool (SECT), which is a HIPE task provided to correct the spectra from semi-extended sources, i.e. those that are partially extended within the SPIRE beam.

SECT was developed to correct sparse-mode point-source calibrated spectra, using the 2D beam profile shapes from the SPIRE Calibration Context and a user-specified model of the source brightness distribution. The output is a spectrum corrected to a reference beam size that is constant with frequency. SECT can be applied from a GUI (semiExtendedCorrector) or in a script. The algorithm used by the tool is described in detail in Wu et al. (2013), Astronomy and Astrophysics, 556, 116 and summarised in the SPIRE Handbook.

7.6.1. Identifying partially-extended sources

The standard pipeline processing for sparse FTS observations assumes that the source is either uniformly extended across the beam, or point-like and centred on the optical axis. In practice, however, most astronomical sources fall somewhere between these two extreme cases. The result of applying unsuitable calibration can be pronounced. This is particularly so in the overlap region of the two FTS bands, as the SLW beam diameter is approximately a factor of two larger than that of SSW. There are a number of sharp features in the beam size versus frequency relationship, which will appear in the spectrum if the wrong calibration is applied. Figure 7.48 shows the beam diameter from the SCalSpecBeamParam product. This plot was generated with the following code, which uses the Level-2 point-source calibrated spectrum product (spss) and Calibration Context (cal):

```python
obs = getObservation(1342198279)
cal = obs.calibration
spss = obs.level2.getProduct("HR_spectrum_point")
beamParam = cal.spec.beamParamList.getProduct(spss)
pl=PlotXY()
pl.addLayer(LayerXY(beamParam[0]["SSWD4"]["0000"].wave, beamParam[0]["SSWD4"]["0000"].beamDiam, color=java.awt.Color.BLACK, stroke=4))
pl.addLayer(LayerXY(beamParam[0]["SLWC3"]["0000"].wave, beamParam[0]["SLWC3"]["0000"].beamDiam, color=java.awt.Color.BLACK, stroke=4))
pl.xtitle="Frequency (GHz)"
pl.ytitle="Beam Size (")"
```
Figure 7.48. The beam diameter in arcseconds, plotted from the Sca1SpecBeamParam calibration product.

Figure 7.49 illustrates the effects of applying the extended-source and point-source calibration to three sources that cover a range of angular size. The bottom-right panel shows the case where a single-pointed, sparsely sampled observation of a uniformly extended source has been calibrated as a point source. The resulting spectra show a distorted shape and a large step between the two bands, and the SSW spectrum is significantly underestimated. By comparison, the corresponding extended-source calibrated spectra (top-right) are smooth and have no step between the bands. Similarly, the spectrum of a point source appears smooth when point-source calibrated, as shown in the bottom-left panel, but will be distorted when extended-source calibrated (top-left). There may also be additional fringing in the wrong calibration for a particular source. The fringes should be removed by the RSRF calibration, but this noise is different for point-like and extended sources. So if the wrong calibration is used then fringes are added rather than removed. Also note the steps in the spectra at 700 GHz and 1250 GHz when the wrong calibration is used. These are due to the abrupt changes in beam size at those frequencies (see Figure 7.48).
Figure 7.49. The spectra of three sources, each with a different angular size, which increases left-to-right. On the left is a point source (red), in the middle a partially-extended source (blue) and a fully-extended source on the right (grey). The data are processed with the extended-source calibration (top row) and the point-source calibration (bottom row). When the appropriate calibration is used (bottom left and top right) a good result is achieved. For the other cases fringing and features from the RSRF are introduced, as well as changes in beam size leading to discontinuity between the spectral bands. The observations used are (left-to-right) CRL618 (1342214858), M83 (1342212345) and the Orion Bar (1342204919).

A correctly calibrated spectrum should contain no discontinuity between the SLW and SSW bands. Thus, the difference in intensity in the overlap region of the two bands provides a simple means for estimating the source intensity distribution. This is particularly important for sparsely sampled observations, because in the absence of corresponding higher spatial resolution maps, the intensity jump may be the only indication of angular extent of the observed object. The centre two plots in Figure 7.49 show a source that is neither point-like or uniformly extended, which results in a discontinuity in both the extended-source and point-source calibrated spectra. The fact that neither calibration looks correct indicates that the spectrum should be corrected using SECT. However this source is known to have a semi-extended nature. In general, caution should be exercised before applying SECT, as there can be a number of other reasons for spectral products exhibiting a steps or a discontinuity between the bands. These reasons are discussed in Section 7.6.2.

The planet Saturn provides a good example of a source that is semi-extended within the FTS beam. It has a diameter of around 17" and a source distribution that can be well modelled using a top-hat elliptical profile. An observation of Saturn (observation ID 1342198279), is used later in this section to provide an example of correcting spectra for partial extent using SECT. This observation was made on operational day (OD) 395, when the diameter of the planet disk was 17.7". There are two other FTS observations of Saturn: 1342224754, with a diameter of 16.7"; and 1342247750, with a diameter of 17.4". All three of these observations were made using bright-source mode. Bright-mode processing was significantly improved from HIPE v11 onwards. However, bright data reduced with HIPE versions 13 and 14.0 (i.e. using the calibration spire_cal_13_0 to spire_cal_14_2) need to be reprocessed with HIPE version 14.1. This is because spire_cal_13_0 to spire_cal_14_2 contained out-of-date bright gain corrections. The bright gain factors were rederived for the release of HIPE 14.1 and can be found in spire_cal_14_3.

Other examples of semi-extended sources can be found in Hopwood et al. (2015).

7.6.2. Does my spectrum need correcting?

The best spectral region to check to see if your observation requires a correction is the SLW and SSW overlap. Here there is factor of two difference in beam diameter between the frequency bands. So if SLW and SSW spectra do not agree, this can be an indication of a semi-extended source. However,
care must be taken before interpreting any step between the two bands as the result of source extent, as
the size of the source within the beam is not the only issue that can lead to such a feature.

Figure 7.50 highlights several factors that can cause a discontinuity between the SLW and SSW bands:

1. Background (or foreground) emission
2. Pointing offset
3. Source extension

SECT can be used to correct for 2 and 3, although these issues should dealt with in the order listed.

Figure 7.50. Three different route causes that manifest as similar discontinuity between SSW and SLW, 
illustrated by standard pipeline point-source calibrated spectra plotted on the left and the corresponding 
corrected data on the right. The observations used are 1 AFGL4106 (1342253667), 2 Uranus (1342259588) 
and 3 NGC6302 (1342268288). Issue 1 can be corrected using the Spectrometer Background Subtraction 
useful script, whilst issues 2 and 3 can be corrected using SECT.

1. Background: Considering a point-like or marginally extended source, any extended background 
(or foreground) emission in your field that is bright relative to the target of interest, will lead to a 
step between SLW and SSW in the point-source calibrated spectrum. For fainter sources, an imperfect 
subtraction of the telescope contribution can also lead to a significant step between the bands. The 
spectra observed by other detectors across the SLW and SSW arrays may be used to subtract unwanted 
background, including telescope residual. The Spectrometer Background Subtraction useful script
is available in HIPE to perform background removal (see Section 7.5.2). In essence, the off-axis detectors 
should be inspected for outliers and averaged to provide a background to subtract from the centre 
detectors. Care must be taken in the case of non point-like sources, as the ring of detectors closest to 
the centre detectors can contain significant emission from the source itself Hopwood et al. (2015). But 
also note that a small fraction of source signal falls on the off-axis detectors, even for point-sources. 
For sources that require a background subtraction due to telescope residual, a second option is to 
subtract a long observation of dark sky that was taken under similar observing conditions, preferably 
taken during the same pair of ODs as the science observation to be corrected. Subtracting a dark sky 
spectrum may improve the continuum shape, however this is generally less effective than a subtraction
based on the off-axis detectors. Subtracting dark sky data is also likely to introduce random noise, in which case it is better to fit or smooth the dark, in order to obtain the wide-scale shape for subtraction.

2. Pointing: If the source was observed with an offset from the centre of the SPIRE beam, the reduced spectrum will show a discontinuity between the bands. In this case, the discontinuity comes from pointing offset causing a greater flux loss in the relatively larger SSW beam, compared to the SLW beam. Section 7.4 contains more detailed considerations on pointing offsets. Pointing can be corrected by SECT by specifying an offset to the model source distribution. If the source is point-like, then pointing offset could explain all of the discrepancy between the bands.

3. Source extension: If a jump between the bands remains after correcting for or ruling out an extended background and this jump cannot be explained by a point source observed with a pointing offset, then source extent is likely to be the cause. SECT can be used to correct the data for source extent, and if there is a contribution from pointing offset, this can be folded into the model source distribution inputted into the task.

Prior to HIPE version 13, a step between the bands could also arise due to a calibration issue. In such cases, the SLW continuum level was below that of SSW. The problem affected sources observed near the beginning of an FTS observing cycle and was most notable for faint sources. Figure 7.51 shows two examples of this offset in HIPE version 12 and version 13 data. The issue was fixed in the HIPE version 13 calibration onwards. However, a few observations taken prior to OD 400 suffer from SLW falling short of SSW. If your data exhibits this behaviour, please contact the Herschel help desk.

![Figure 7.51. Two examples showing the improvement in calibration for faint observations processed with HIPE version 13 compared to HIPE version 12. The step between the bands is eliminated or significantly reduced in HIPE version 13.](image-url)
7.6.3. Using the SECT GUI

Figure 7.52 shows the layout of the SECT GUI (semiExtendedCorrector), which can be found in the HIPE Tasks window. It can be selected in the list of All tasks or in the list of Applicable tasks for a point-source calibrated spectrum object (an SPSS product), which is required as an input, along with the SPIRE Calibration Context. Either the whole Calibration Context can be supplied to the tool, or the SLW and SSW Spectrometer beam profile calibration products. In Figure 7.52, the variables pointSourceCalibratedSpectrum (an SPSS) and cal (the Calibration Context) have been dragged and dropped onto the spectrum and calibration buttons, at the top of the semiExtended-Corrector window. The keepDetectors parameter controls the detectors considered for correction. By default keepDetectors is set to a list containing the names of the centre detectors, so for an SPSS, SECT will only apply a correction to these, with the off-axis detectors removed in the correctedSpectrum output. All the point-source calibrated detectors will be considered if keepDetectors is set to all. However, as SECT can only correct spectra that are overlapping with the source model, the off-axis detectors are not likely to be present in the output, and if any are present, these are not likely to be meaningful for further analysis. If a source was observed off-axis, the pair of coaligned detectors pointed at the source can be specified by setting keepDetectors to a list of their names.

7.6.3.1. Source distribution model

A source distribution model can be defined using the Source model parameters indicated in Figure 7.52. SECT has three options for the input model type. These are a top hat function (top hat), a Gaussian profile (gaussian), or a Sersic profile (sersic). The model type can be selected from the Shape name pulldown menu. Parameters to define the source shape can be entered in the Diameter, x-axis offset, y-axis offset, Eccentricity, Rotation angle and Sersic index boxes. The Sersic profile used is defined by

\[ I(r) = A \exp\left(-\left(r/r_0\right)^n\right), \]
where \( A \) is a normalising factor, such that the area of the profile is 1.0, \( r \) is the distance from the centre of the profile, \( r_0 \) is the scale radius, and \( n \) is the Sersic index. For a Sersic profile, 2 times \( r_0 \) should be entered in the Diameter box. A point source model can be set using a Gaussian profile with a diameter of 0.01".

The defined profile appears as an image under the array footprint at the bottom of the GUI, where the detector names can be turned off or on using Show detector labels. A right mouse click inside the image brings up a menu of items, including the option to zoom.

Note that errors increase significantly for source models larger than the SSW beam size and the tool cannot accurately reproduce the extreme case of fully-extended emission, see Section 7.6.6 for more details.

The model parameters specified are recorded in the metadata of the correctedSpectrum returned by SECT.

### 7.6.3.2. Correction options

The SECT GUI has three tick-box options located between the variable inputs and source model definition boxes. These options are:

- **optimiseDiameter**, which allows the task to optimise the diameter of the source distribution model. Optimisation is performed by iterating around the input value until the difference between SSW and SLW spectra in the overlap region are minimised. Note that only the diameter parameter is changed, while the other parameters are fixed at their input values.

- **doPlots**, which if True produces two plots. One of the data before and after correction, and the other of the applied correction factor (see Figure 7.53). If optimiseDiameter is also set to True, a third plot is produced, which shows the spectra corresponding to the source diameters checked during the optimisation process. For an SPSS input, the centre detectors are used for all three plots.

- **applyCorrection**, which determines the output of SECT. If set to True then the output correctedSpectrum contains the corrected input data, while if False, the output will contain the correction curve itself. It may be useful to obtain the correction in order to check the effect of changes to the source model parameters.

![Figure 7.53. The plots produced by SECT when doPlots is set to True (and optimiseDiameter is False) for the observation of Saturn with observation ID 1342198279.](image)

### 7.6.3.3. Reference beam

There are two further options for SECT (gaussRefBeamDiam and couplingThreshold), which control the reference beam for normalising the corrected spectrum and the coupling threshold, below which no output is produced.
The reference beam is used to normalise the output of SECT. The aim is to correct the input data to that which would have been observed if the SPIRE FTS had a frequency independent Gaussian beam. By default, the FWHM of the reference beam is set to 40′. The effect of the reference beam is illustrated in Figure 7.54. From this figure it can be seen that as the size of the reference beam is increased, so the flux density encompassed within the beam increases, until the whole source distribution is included. The FWHM can be modified to match that of another telescope at a particular frequency, to facilitate a direct comparison between the respective data.

![Figure 7.54. Diagram of an example source model distribution and the reference beam, with the source distribution assumed independent of frequency. The final spectrum is normalised to include only emission inside the reference beam. If the FWHM of the reference beam is increased, more flux density is encompassed within the beam. And if set large enough, the entire source distribution will effectively be included inside the reference beam.](image)

The correction derived and applied by SECT is normalised by the reference beam, such that no correction is made at the frequency where the actual beam size and shape equals that of the reference beam. With the default FWHM of 40′, the normalised correction applied is 1.0 at around 950 GHz in SLW. At frequencies where the reference beam is smaller than the true beam, the correction will reduce the flux density of the input spectrum, and for frequencies where the reference beam is larger than the true beam, the flux density will increase.

The normalisation due to the reference beam can be removed by setting it a large FWHM, which is equivalent to a flat beam with a value of 1.0 everywhere in the source model image. The output of the tool is then be the total flux density of the source model distribution. See Wu et al. (2013) for more details of how the algorithm inside the tool utilises the reference beam.

The source-beam coupling is defined as the double integral of the source distribution multiplied by the beam shape (see Wu et al. (2013)). When the coupling tends to zero, the corrected spectrum tends to infinity because SECT tries to correct emission which was never observed. In order to prevent the task calculating an infinite correction factor for detectors in the array that do not overlap with the source model, a coupling threshold is defined with `couplingThreshold`. The value of the final correction factor is the normalised source-beam coupling, which is compared to the threshold limit. This comparison is made at 700 GHz for SLW and 958 GHz for SSW. If this comparison finds the the normalised source-beam coupling is below the limit for a particular detector, then that detector is removed from the output. The default value for the `couplingThreshold` is 0.2. This value can be adjusted between 0 and 1, however the default is generally effective at removing potentially confusing spectra from the output. The threshold value does not change the algorithm used by SECT, only those detectors included in the output product.
7.6.4. Pointing considerations

Before SECT corrects for any semi-extended nature or a user specified pointing offset, it firstly carries out two preparatory "de-corrections" of the input data. One is applied to all data, to put back the non-perfect alignment between the centre detectors, as SLWC3 is offset from SSWD4 by 2.2". The other is applied only for observations taken before OD 1011, in order to re-introduce the BSM offset of 1.7" that was corrected for observations made this OD (see Section 7.4). The observations of Uranus used to derive the point-source calibration were taken under the same conditions as the data being calibrated. Which means the calibration inherently accounts for the misalignment between the centre detector and the BSM offset for observations from before OD 1011 and that the final processed data is properly. However SECT needs to reset these known (and fixed) offsets, to ensure that no further correction will be applied to observations of a point-source located at the BSM offset position of [-0.91, 1.46] and observed before OD 11011, or for a point source observed after OD 1011 and located at [0, 0].

7.6.5. Scripting with SECT

7.6.5.1. Source distribution model

More options are available for SECT when running the task using a script. For instance, a more complicated source model can be specified. The simplest method of defining a source model is to use SECT's internal options of a top hat, Gaussian or Sersic. With the source shape defined by diameter, x-axis offset, y-axis offset, eccentricity, rotation angle and Sersic index, e.g.,

```python
# Create an elliptical top hat model image:
# (Model, Diameter, x-off, y-off, eccentricity, rot. angle, Sersic index)
shape = SemiExtendedSourceModelShape("top hat", 18.0, 0.0, 0.0, 0.5, 90.0, 1.0)
```

The resulting shape is an object with a SemiExtendedModelShape class, from which the actual model profile can be extracted as a Double2d array, following:

```python
shapeArray = shape.sourceModelImage
```

shapeArray can be visualised in HIPE by right-clicking on the variable and selecting Open With > 2d/3d Image Viewer.

In order to match the beam profile shapes that are inside the Calibration Context, any source model supplied to SECT must be a 257x257 image, with a 1" pixel scale. These constraints are necessary because rather than using a WCS, SECT uses a pixel-by-pixel comparison to calculate the coupling of the model with the beam profile. So the input image cannot have arbitrary sizing, although see Section 7.6.5.2.

In order to change the model image profile, the extracted 2D array can be modified and then entered directly into SECT, which will accept an input of a SemiExtendedModelShape, a Double2d or a SimpleImage object. However if the image profile is modified, the 1" pixel size and dimensions of 257x257 must be preserved. An image created outside of SECT can also be used for the model, as long as it adheres to the necessary pixel grid constraints. Any input for the model source profile should be normalised to an area of unity.

With the source model image prepared, the next step is to use SECT to correct the observed point-source calibrated spectra, for example:

```python
# Load the Observation Context (obs)
obs = getObservation(1342198279)
# Extract the Level-2 point-source calibrated product (spss)
spss = obs.level2.getProduct("HR_spectrum_point")
# Run the tool to obtain the corrected spectrum
```
correctedSpectrum = semiExtendedCorrector(spectrum=spss, calibration=cal, \n   doPlots=1, sourceModel=shape)

If applyCorrection is set to False, the output correctedSpectrum will not contain the corrected data, but the correction calculated for the input shape model itself (e.g. the left hand plot in Figure 7.53):

correction = semiExtendedCorrector(calibration=cal, sourceModel=shape, \n   applyCorrection=False)

7.6.5.2. User provided beam maps and/or source models: expert use

Generally the beam maps from the Calibration Context are the best choice to be used with SECT, with a source model image that matches the dimensions and scale of the beam maps (as described in the previous section). But it is possible to input a custom beam map and source model image pair, which allows a different pixel scale to used. Up-sampling the beam map, by rebinning and smoothing, can reduce the effects of pixellisation on the calculations for small source models. There are several limitations to this approach:

1. Each dimension of the beam map must equal an odd number of pixels.

2. The dimensions and pixel scale of the beam maps must be identical to the source model image.

3. Regardless of the pixel scale used for the input beam map and source model image, SECT uses a fixed pixel scale of 1", therefore:

   • the Gaussian reference beam diameter is assumed to match the internal tool pixel scale. If the input dimensions are adjusted to a different pixel scale, then the reference beam diameter must also be adjusted.

   • the relative positions of different detectors are constants in the tool pixel scale, so for a different input pixel scale there will be errors in positioning for all detectors except SSWD4.

7.6.6. How much can I believe the SECT results?

The output of SECT is a spectrum with flux density units of Jy, referenced to the supplied Gaussian beam size, which is 40" by default. The error on the final spectrum is likely to be dominated by the uncertainty on the source distribution model used. One way to estimate the associated error is to run the tool with a range of source models and examine the difference between the resulting spectra. Another way would be to run a Monte Carlo simulation with simulated spectra, as described in Wu et al. (2013).

The correction applied attempts to replicate the results equivalent to observing a fully sampled spectral map and convolving every frequency layer to match the largest beam. Therefore, in order to gain some confidence that the tool is achieving a good result, one of the SPIRE fully sampled mapping observations could be examined. Extracting a point-source spectrum from the map and applying SECT should give the same result as convolving the map to the same beam size and then extracting a point-source spectrum. Details of how to extract a point-source spectrum from a cube and to convolve a cube to a different beam size are given in Section 7.12, along with reasons as to why these two may not always agree.

SECT only applies a correction for the forward coupling efficiency. Wu et al. (2013) describe other effects that are not included - the efficiency with which the reconstructed beam shape couples to the source; the deviation of the model from true source distribution; and the response far from the optical axis. The magnitude of these effects was measured by Wu et al. (2013) for sources up to the size of Saturn (17") and was found to be close to 1.0. However for larger sources, the effect is more significant and for fully-extended sources, it is roughly a factor of two. This means that SECT cannot be used to reproduce the extended-source calibrated spectrum, which assumes a flat infinitely extended source.
Considering these results, SECT is best restricted to sources that are smaller than the SSW beam size of around 20''.

SECT assumes the source distribution is constant in frequency. However, this may not be true because

- Spectral lines may have a different distribution to the continuum
- Different parts of the source may have different temperatures
- The distribution may be different for lines of different energies

In any of these cases, it may be necessary to work with SECT in a script and build up a correction by looping over frequencies/spectral lines and using a different source model for each one. In particular, care should be taken with the option to optimise the source model diameter, as this only takes account of the continuum in the overlap region between the two bands. Therefore the optimised diameter may not be representative of the distribution of the spectral line emission.

The way SECT is implemented in HIPE (see Wu et al. (2013) and the SPIRE Handbook) means that pixelisation issues can fold some uncertainty into the corrected spectrum. The algorithm relies on multiplying two images with 1'' pixels, which limits the accuracy that can be achieved for small source model diameters or for small spatial offsets. The option for top hat or Gaussian source distributions include some corrections to reduce pixelisation issues, but the beam shape images in the Calibration Context are still only defined on a 1'' grid. For this reason, the minimum diameter that can be used for top hat source models is limited to 3'', as errors increase significantly for smaller diameters.

### 7.7. Recipes for bright sources (above 500 Jy)

The FTS observed sources significantly brighter than 500 Jy, and up to 25,000 Jy, using bright source mode. By using different detector bias settings in order to reduce the responsivity of the instrument, this mode allowed observations of sources that would otherwise have saturated the detectors, such as Orion, Sgr B2 and Mars. Details of the calibration of bright mode data are given in Lu et al. 2014, Experimental Astronomy, while the mode used for an observation can be determined from the metadata item `biasMode`, which can be found in the Observation Context and individual products. The entry will have a value of `nominal` or `bright`:

\[
\text{biasMode} = \text{obs.meta["biasMode"].value}
\]

As bright mode works by reducing the responsivity of the bolometers, there is increased noise in the final spectrum. The noise in bright mode observations should be expected to be approximately a factor of 2 worse than the nominal mode for the SLW array and a factor of 4 worse for the SSW array. Considerations for bright sources are therefore slightly different to those covered in Section 7.5, as uncertainty in the background is not a significant issue.

From HIPE version 11, the pipeline processing of bright mode observations was revised to use the same tasks as the nominal pipeline, including the nominal mode calibration products for flux conversion. However the reduced responsivity of the detectors is taken into account with a dedicated bright mode non-linearity correction calibration product, with a bright mode gain factor applied near the end of the pipeline. Therefore it is recommended to reprocess bright mode observations using HIPE version 11 (or later), as the non-linearity of the detectors will be better corrected and the continuum levels more trustworthy. **However bright data reduced with HIPE versions 13 and 14.0 (i.e. using the calibration spire_cal_13_0 to spire_cal_14_2) need to be reprocessed with HIPE version 14.1. This is because spire_cal_13_0 to spire_cal_14_2 contained out-of-date bright gain corrections. The bright gain factors were rederived for the release of HIPE 14.1 and can be found in spire_cal_14_3.**

Note that the point-source and extended calibration considerations that are described for faint and medium strength sources in Section 7.5.1, also apply to bright mode data.
7.8. Recipes for mapping observations

An FTS observation of the Orion Bar is used throughout this section to illustrate an extended source observed in fully sampled mapping mode. The observation used is from operational day (OD) 495, with observation ID (obsid) 1342204919, and can be obtained from the Herschel Science Archive.

The FTS pipeline for mapping observations produces two sets of SpectralSimpleCubes for each detector array. Each set is created using a different projection algorithm, although both project the measured detector data onto an equivalent rectangular, equidistant grid. The following sections describe aspects of data reduction to consider for FTS observations taken in intermediate and fully sampled mapping modes:

- Understanding the SPIRE beam Section 7.8.1
- Checking for clipping in individual interferograms Section 7.8.2
- Examining the actual positions observed on the sky Section 7.8.3
- Restricting or expanding the data for gridding Section 7.8.4
- The gridding algorithms Section 7.8.5
- Examining coverage and redundancy in the cube Section 7.8.6
- Maps with faint continuum Section 7.8.7

A notable aspect of FTS mapping is the two unusable detectors in SSW (SSWD5 and SSWF4), which marginally reduce coverage. Another coverage related issue to consider is the outer ring of detectors in both arrays, as these are partially vignetted (see the SPIRE Handbook). In order to provide the largest possible gridded area, these detectors are included in the pipeline cube creation step. However, some of these detectors have lower than average sensitivity and higher than average systematic noise, as discussed in Hopwood et al. 2015. If the map in question is of a relatively faint source, it may be worth regridding the spectral cubes in order to exclude the partially vignetted detectors. This can be achieved by setting unvignetted = True when calling spirePreprocessCube, which creates the 2D list of spectra for projecting. From HIPE version 14.1, a correction for artefacts in LR SLW data is propagated to the extended-source calibrated spectra. This is an empirical correction derived using LR point-source calibrated data (see Marchili et al. 2016 for details). As there is no point-source conversion for the vignetted detectors, the vignetted detectors are uncorrected and therefore not included for LR SLW cubes processed with HIPE version 14.1 (or later). This is true for the LR SLW cubes from H+LR observations, so there is a difference in area with the corresponding HR SLW cubes.

The Spectrometer Thumbnail Mosaic Plot script is useful for plotting many detectors from data taken during a single pointing measurement. It can be edited for use with mapping observations, in order to display the spectra taken at each jiggle position. The script allows the spectra from all detectors in the array to be plotted as they appear on the array footprint. See Section 7.2.3.2 for more details.

7.8.1. Understanding the SPIRE beam and how it relates to mapping observations

The FTS beam is frequency dependant, as shown in Section 7.6.1, and described by two calibration products that have a frequency scale corresponding to the frequency axis of the spectrum. SCalSpecBeamProf contains the 2D-beam shapes and SCalSpecBeamParam contains the beam FWHM. SCalSpecBeamParam also contains the point-source calibration factors, which are dependent on spectral resolution and OD. The dependency on OD is not because of any change in the beam FWHM, but due to a difference in the point-source calibration for observations taken before OD 1011 and those taken after, as explained in Section 7.4.1. The appropriate beam parameter calibration product for a particular observation can be obtained from the calibration Context using any Level-2
product, for example, if using a spectral cube, `beamParam` can be obtained from an Observation Context (`obs`) as:

```python
beamParam = obs.calibration.spec.beamParamList.getProduct(cube)
```

This calibration product can also be extracted by specifying the resolution and date of the observation, for example,

```python
beamParam = obs.calibration.spec.beamParamList.getProduct("HR", obs.startDate)
```

For a particular detector (e.g. SSWD4) the beam diameter column can be accessed from `beamParam` as

```python
beamDiam = beamParam["0000"]["SSWD4"].beamDiam
```

The beam diameter corresponds to the 2D-beam shapes that are stored in the `SCalSpecBeamProf` calibration product, which were derived by moving the telescope over the planet Neptune. The beam shapes can be obtained from the calibration tree, e.g. for SSW:

```python
beamProfSsw = obs.calibration.spec.beamProfList.getProduct("SSW")
```

The resulting product can be viewed with the Spectrum Explorer in the same way as for Spectral Cube products. The beam shapes are Gaussian for SSW, but not for SLW above 700 GHz. Full details of the beam profiles are given in Makiwa et al. (2013), Applied Optics, 52, 3864.

Note that from HIPE version 12 onwards, FTS spectra are corrected to the Local Standard of Rest (LSR) by the pipeline. This correction is carried out towards the end of the pipeline, which means that although the frequency axis of the calibration product and the final spectra will match, there may be a small linear shift compared to the actual values applied.

### 7.8.2. Check clipping

Clipping due to saturation of the electronics may provide an explanation for more holes in a spectral cube than expected. Detectors with severely or totally clipped interferograms are removed from the final data products by the pipeline, after which nothing can be done to recover the data. This section should allow you to determine if an observation has suffered clipping in this way.

Clipping occurs when the signal received by a detector is outside the dynamic range of the electronics, which causes truncation in the detected interferograms. Clipped samples are flagged in the mask table attached to the data as TRUNCATED. Details of the SPIRE masks and mask handling are given in Section 8.4. If the truncation is not too severe, the clipping correction task generally works well to reconstruct the missing parts of an interferogram, using an 8th order polynomial fit. In tests where the interferogram was clipped, such that the 4th lobe away from zero path difference was affected, the spectrum was corrected to better than 3%. This means that clipping should not be an issue for most observations.

Cases of more severe clipping may occur for maps of relatively bright sources, if large flux density gradients were observed with nominal detector bias settings. For maps taken in nominal mode, the dynamic range of the detectors is set using the observed signal at the beginning of the observation (see Section 7.1.2), which means the detector offset level is determined only at the first jiggle position. If the source brightness changes significantly for subsequent jiggle positions, some detectors may become unusable and will be removed from the data products by the pipeline. The `createIfgm` task removes samples that are clipped at the high and low OPD (optical path difference) ends of the interferogram. If all samples are removed, then so is the entire interferogram (i.e. one scan) for that detector. If all scans are removed, the detector will no longer appears in the output product. The `baselineCorrection` task will remove any scan that has fewer than 100 samples.

The final spectrum may be affected by scans where some samples were clipped at high OPD and removed, but not enough to see the whole scan removed. The example observation of the Orion Bar
(1342192173) suffered clipping in SSWD1, as illustrated by Figure 7.55. This figure shows the Level-0.5 data from the detector in question, in jiggle position 13. This detector is in the outer ring, where partial vignetting introduced a slope to the interferogram baseline (see Section 7.3.15). Some samples at the high OPD end are clipped, which reduces the OPD range of the interferogram and therefore results in a decrease of the spectral resolution. The final spectrum for this position shows broader lines than for the surrounding detectors and should be removed before the cube is created and any line fitting is performed. The spectral resolution can be checked in the pre-processed cube product (which collates the individual extended-source calibrated spectra), where the resolution is recorded for each spectrum in the “resolution” column. In the individual extended-source calibrated spectra (SDS products), the spectral resolution is recorded in the actualResolution metadata item, which is attached to each detector dataset.

Figure 7.55. On the left, the Level-0.5 data for a clipped detector (SSWD1) in jiggle position 13 of obsid 1342192173 are plotted in the Detector Timeline Explorer. On the right, the final extended-source calibrated SSWD1 spectrum is compared to the unclipped SSWC1 spectrum, plotted using the SDS Explorer. The interferogram data shows truncation both near to zero path difference, which has been corrected by the pipeline, and at high OPD, which has been removed by pipeline. The reduction in OPD range causes the final spectrum to be reduced in spectral resolution (as if it had been apodized).

In order to verify that clipping is the underlying cause for data missing from the final data products, it is necessary to examine the Level-0.5 timelines, for example using the Detector Timeline Viewer (see Section 8.2), or to look at the Level-1 interferogram product using the Spire Mask Editor Tool (see Section 8.4). It should be obvious if an interferogram has been terminally truncated, because there will be a completely flat baseline with little or no signal at zero path difference. There is no way to recover the data for such cases.

### 7.8.3. Pointing Information

For a mapping observation, information on pointing can be examined by extracting the Spectrum2d dataset contained inside the preCube product, which can be achieved either using the Product Viewer or by running the following command:

```python
mySpectrum2d = preCube.spectrum2d
```

The RA and Dec for each spectrum are contained in the longitude and latitude columns of `mySpectrum2d`. 
mySpectrum2d can be visually inspected by right clicking on it in the variables tab and opening with the Spectrum Explorer (see Figure 7.56).

Figure 7.56. Displaying the Spectrum2d dataset extracted from a preCube, in the Spectrum Explorer. The RA and Dec for each spectrum is shown, and individual spectra can be plotted by clicking in the second column labeled "0". Double-clicking on the "0" column title will deselect or select all spectra.

For SSW, the two dead detectors (SSWD5 and SSWF4) lead to less even coverage of observed sky positions across the field of view. Figure 7.57 shows a plot of the observed coordinate positions for the fully sampled Orion Bar observation for SSW and SLW. This plot can be generated with the following:

```python
from herschel.ia.gui.plot.renderer.axtype import AxisType
mySpectrum2d = preCube.spectrum2d
p=PlotXY(mySpectrum2d["longitude"].data, mySpectrum2d["latitude"].data)
p.setLine(0)
p.firstLayerXY.setXAxisType(AxisType.RIGHT_ASCENSION)
p.firstLayerXY.setYAxisType(AxisType.DECLINATION)
p.setXtitle("RA [hh:mm:ss]")
p.setYtitle("DEC [dd:mm:ss]")
```

The positions observed on the sky can also be examined from within the Spectrum Explorer, using the Mosaic (or Raster) Tab, as explained in the Spectrum Explorer section of the Herschel Data Analysis Guide.

Figure 7.57. The actual positions observed on the sky for the fully sampled example observation of the Orion Bar. SSW is on the left and SLW on the right.
7.8.4. Restricting or expanding the data for gridding

Prior to gridding, the pipeline collates the individual extended-source calibrated spectra (SDS) into a SpirePreprocessedCube product, which is the pre-processed cube and labelled as the preCube in the pipeline. The preCube is a listing of all spectra and positions in a Spectrum2d format. All detectors are included in the preCube, but there are some basic options (methods) that allow data to be removed. Individual or selected groups of spectra can be removed using the deleteSpectra method and specifying the detector name(s), jiggle ID and raster ID. For example:

```python
# Remove all SSWD1 spectra from all jiggle/raster positions:
preCube.deleteSpectra("SSWD1")
# Remove all spectra from jiggle position 13, raster position 0:
preCube.deleteSpectra(13, 0)
# Remove SSWD1 from jiggle position 13, raster position 0:
preCube.deleteSpectra("SSWD1", 13, 0)
# Multiple detector name can also be specified in a list:
preCube.deleteSpectra(["SSWD1", "SSWC2"], 13, 0)
```

Note that jiggle and raster ID start at 0, and if the specified spectrum doesn't exist (or the jiggle or raster ID are specified incorrectly), nothing will be removed.

The spectra in a preCube can be extracted as an array of SpireSpectrum1d objects using the extractSpectra method. The syntax for specifying which spectrum (or spectra) to extract is similar to that for deleteSpectra:

```python
# Extract all SSWD1 spectra from all jiggle/raster positions:
spectraArray = preCube.extractSpectra("SSWD1")
# Extract all spectra from jiggle position 13, raster position 0:
spectraArray = preCube.extractSpectra(13, 0)
# Extract SSWD1 from jiggle position 13, raster position 0:
spectraArray = preCube.extractSpectra("SSWD1", 13, 0)
# Multiple detector names can also be specified in a list:
spectraArray = preCube.extractSpectra(["SSWD1", "SSWC2"], 13, 0)
# Get the first SpireSpectrum1d from the array:
spectrum1d = spectraArray[0]
```

Several observations can be combined into a single cube by combining two or more preCubes, following:

```python
preCube = preCube1.append(preCube2)
```

Once all of the required spectra are contained in a single preCube product, they can be gridded into a cube using SpecWcsCreator and spireProjection (see Section 7.8.5). For example, if using the Naive projection algorithm:

```python
wcs = SpecWcsCreator.createWcs(preCube, gridSpacingX, gridSpacingY)
combinedCube = spireProjection(spc=preCube, wcs=wcs, projectionType="naive")
```

Observations processed with a pipeline version prior to HIPE 13, will not have the preCube saved inside the Observation Context, with the individual SDS products stored in the Level-1 Context instead. This change in format, combined with a correction for the extended calibration (as explained in Valtchanov et al. 2016), which was introduced for HIPE 14, means that it is essential for all mapping observations to be updated to HIPE 14 processed data. If the observation in the Herschel Science Archive has been reduced with HIPE version 14, it can be downloaded directly. If the version is HIPE 13, then use the on-demand reprocessing option or run the HIPE 14 User Pipeline mapping script. For the latter option, it does not matter if the data is not in the HIPE version 13 and later Observation Context format, as the script will automatically update the Context to the new format.

7.8.5. The gridding algorithms

There are currently four SPIRE compatible gridding algorithms available in HIPE, which are listed in Table 7.6. Each gridding algorithm (also called projection algorithm) requires an RA and Dec grid
specified as a WCS object. The WCS is set up using a preCube and desired grid spacing in degrees, e.g., for a grid with 17.5" pixel size:

```python
wcs = SpecWcsCreator.createWcs(preCube, 17.5/3600.0, 17.5/3600.0)
```

The way the samples are mapped onto the WCS grid can be examined by overplotting the grid lines on top of the observed data points. The following script was used to produce Figure 7.58, where grid lines have been added to the SSW plot from Figure 7.57:

```python
start = wcs.getWorldCoordinates(0,0)
end   = wcs.getWorldCoordinates(wcs.naxis2, wcs.naxis1)
ystep = wcs.cdelt2
for i in range(wcs.naxis1+1):
    xGridLine = start[0] + xstep*i - xstep/2
    p.addLayer(LayerXY(Double1d([xGridLine, xGridLine]), [start[1]], color=java.awt.Color.GRAY))
    yGridLine = start[1] + ystep*i - ystep/2
    p.addLayer(LayerXY(Double1d([start[0]], [yGridLine, yGridLine]), [start[1]])-ystep/2, color=java.awt.Color.GRAY))
```

![Figure 7.58](image.png)

**Figure 7.58.** The observed SSW sky positions for the example Orion Bar observation (Figure 7.57), overlaid with a WCS grid to illustrate how the individual spectra are gridded to create a spectral cube, where each grid square corresponds to a cube spectral pixel.

The WCS can be set up by forcing the reference pixel to be at a specified RA and Dec. For example, the following command will put the reference pixel at the `crval1` and `crval2` RA and Dec of [83.849707, -5.397227].

```python
wcs = SpecWcsCreator.createWcs(preCube, 83.849707, -5.397227, 
                                          17.5/3600.0, 17.5/3600.0)
```

The WCS can also be constructed in instrument coordinates rather than RA and Dec, which rotates the image to match the satellite Y and Z axes. One example where this might be useful, is to check the position of missing pixels in the map against the locations of the two dead detectors in the SSW array. The gaps from these detectors (SSWD5 and SSWF4) are always in the same part of the map when working in instrument coordinates. A WCS in instrument coordinates can be constructed by setting `useDetCoords` to True.
wcs = SpecWcsCreator.createWcs(preCube, 17.5/3600.0, 17.5/3600.0, True)

<table>
<thead>
<tr>
<th>Task name</th>
<th>Algorithm details</th>
<th>SPIRE Pipeline Specification Manual entry</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>nearestNeighbourProjection</td>
<td>Nearest spectrum to grid square centre is used. No averaging or interpolation.</td>
<td>Nearest Neighbour Spatial Projection in SPIRE Pipeline Specification Manual</td>
<td>The spectrum closest to the centre of a grid square is used, even if its position is not inside the grid square. Not generally recommended.</td>
</tr>
<tr>
<td>convolutionProjection</td>
<td>Each grid square is assigned the Gaussian weighted contribution of all spectra falling within the kernel area.</td>
<td>Convolution Spatial Projection in SPIRE Pipeline Specification Manual</td>
<td>Grid squares assigned a Gaussian weighted contribution from spectra within the kernel. Used in the standard FTS pipeline.</td>
</tr>
<tr>
<td>griddingProjection</td>
<td>Per grid square, averaging and interpolation of all spectra using a convolution kernel to model the beam profile</td>
<td>Gridding Spatial Projection in SPIRE Pipeline Specification Manual (also see the HIFI User's Manual)</td>
<td>A SPIRE wrapper to use the HIFI task for On-The-Fly maps. A HIFI build is required. This algorithm should not be used for intermediate sampled FTS mapping observations.</td>
</tr>
</tbody>
</table>

These algorithms can be applied using `spireProjection` or with the tasks shown in Table 7.6. When using `spireProjection`, the `projectionType` can be `nearest`, `naive`, `convolution` or `gridding` (see the SPIRE Pipeline Specification Manual - SPIRE Projection in SPIRE Pipeline Specification Manual). For example:

cube = spireProjection(spc=preCube, wcs=wcs, projectionType="naive")
Two sets of spectral cubes are created by the standard pipeline, with each set containing one cube per detector array. One set of cubes is created with the Naive projection algorithm, which is the same map making algorithm used for the SPIRE photometer maps (the NaiveMapper). This projection calculates the mean sum for all spectra located within one grid square (a spectral pixel or "spaxel"), as illustrated by Figure 7.59. Empty grid squares are left blank, with the corresponding spaxel set to NaN. The other set of cubes is generated using the Convolution projection algorithm, which results in more complete coverage, i.e. there are no NaN "holes" within the main body of the grid for a Convolution projected (CP) cube. For each spaxel, this projection sums the Gaussian weighted contribution from all spectra in the relevant detector array that fall within the kernel. The Gaussian kernel used is set to a FWHM of the beam diameter and limited to a width of twice this. Although there are no holes spatially (i.e. per layer) in the resulting cubes, because the beam width is frequency dependent, so is the coverage. Therefore, as there is a minimum allowed coverage of 0.1, there can be holes spectrally at the map edges. The FTS beam is smaller towards the middle of each band, so these spectral gaps occur towards the centre of the frequency bands. However, such partial spectra are few in number and confined to the edges of the cube grid.

Figure 7.60 shows a comparison of the naive SSW and the CP SSW cubes for the example Orion Bar observation. The cube map at 1200 GHz is overlaid by the FTS footprint for the central Jiggle position and the position of the individual spectra that were projected into each grid square. For the naive cube, the "holes" correspond to zero spectra within a spaxel, whereas for the CP cube there are no holes and the map has a smoother appearance.
Figure 7.60. SSW cube maps at 1200 GHz. Both cubes are overlaid with the FTS footprint for the central Jiggle position (pink circles) and all the positions of the ungridded spectra (green pluses). On the left is the naive projected cube and on the right is the convolution projected cube, which is smoother with no "holes".

The Convolution projection task has a default kernel size of 35" for SLW or 19" for SSW but the size can be set explicitly. As in the FTS mapping pipeline, a frequency dependent convolution kernel width can be used, e.g., calculated from the beam diameters in the SCalSpecBeamParam calibration product (see Section 7.12.5):

```python
cube = spireProjection(spc=preCube, wcs=wcs, projectionType="convolution",
beamWidthArray=kernelWidth)
```

The Gridding and Convolution projection tasks use a similar algorithm, and will produce the same results for a fully sampled mapping observation, if the beam width when using the Convolution task is set to 0.3 times the value used for the Gridding task. For intermediate sampled maps, the Convolution task can be set to have a relatively large kernel radius, so the resulting cubes do not contain any "holes". In contrast, the Gridding task has more stringent limits on the maximum size of the convolution kernel, which is not sufficiently large enough to populate all spaxels with at least one spectrum and results in "holes" in the cube. Therefore the Gridding task should not be used for intermediate sampled maps. The Gridding projection task is a wrapper for the task used for HIFI spectral maps, and so running the Gridding projection requires a HIPE installation that contains the HIFI software (e.g., selecting "all instruments" when installing HIPE). The task can then be run using the spireProjection task as:

```python
cube = spireProjection(spc=preCube, wcs=wcs, projectionType="gridding")
```

A comparison of the results obtained for the example Orion Bar observation, using the Nearest Neighbour, Naive, Convolution and Gridding projection algorithms, is shown in Figure 7.61.
Figure 7.61. SSW cubes of the example Orion Bar observation. From left-to-right, these were created using the Nearest Neighbour, Naive, Convolution and Gridding Projection tasks. The corners are filled with adjacent values when using the Nearest Neighbour task, but the noise levels in the spaxels are also higher as there is no averaging. There are NaN spaxels within the map for the Naive projection, but no holes when a convolution is applied.

The four projection algorithms provide different ancillary fields in the resulting spectral cubes. Nearest Neighbour projection returns a cube with flux, error and flag. Naive and Convolution projections return a cube with flux, error, coverage and flag. Gridding projection returns a cube with flux, error and flag. The coverage for Naive and Convolution projections is described in Section 7.8.6.

7.8.6. Holes in the map: examining the coverage

The Naive projection and Convolution projection tasks provide the coverage. For any given detector, the standard pipeline product is the average of all forwards and reverse scans, with the number of scans recorded in the "numScan" column. For the Naive projection, the coverage is the total number of scans falling within a spaxel, i.e. the sum of the "numScan" column for all spectra within a spaxel. The minimum number of repetitions an FTS observation was allowed was two, and there are one forwards and one reverse scan per repetition, so unless a scan has been removed during processing, the minimum coverage for Naive cubes is four. In the case of the Convolution projection task, the coverage is convolved by the kernel in the same way as for the signal, and so the weighted contribution made by all scans within the kernel area are combined. The coverage is conserved so that the sum of the coverage cube should be the same for both the Naive and Convolution projection results. The default minimum coverage for CP cubes is 0.1. However this limit is not per spaxel, so spectra containing gaps can be present towards the grid edges of CP cubes.

The coverage can be visualised by opening a cube with the Spectrum Explorer and selecting the "Cov" field in the drop-down menu to the bottom right, as shown in Figure 7.62. It is likely the cut levels will need to be adjusted, which is provided as an option when right-clicking on the cube image. The coverage cubes in Figure 7.62 are both cut at 90%. The coverage information could be used for statistical analysis, or to adjust the map pixel (grid) size such that the coverage goes up or down.
Figure 7.62. The coverage of the Naive projected (naive) and Convolution projected (cp) SSW cube of the example Orion Bar observation. Each spaxel contains data from 0 (black), or 4 (orange) to 12 (white) spectral scans.

Increasing the map pixel size can avoid creating holes in the Naive map (i.e. map pixels which do not contain any data). The pixel size can be increased by specifying a different grid spacing in the mapping user pipeline script. The default grid size for fully sampled maps is 9.5'' for SSW and 17.5'' for SLW. Figure 7.62 of the coverage map for the example Orion Bar observation, shows those pixels not containing any data points (as black) for SSW with this pixel size. If the standard grid was modified to have 12.5'' pixels, for example, the holes in the SSW map would disappear. Similarly the SLW pixel size can be reduced slightly without introducing holes.

7.8.7. Maps with faint continuum levels

When a faint source has been observed in mapping mode, uncertainties from the telescope model can affect the continuum shape in the same way as for a single pointed observation of a faint source (see Section 7.5). In such cases, it may be possible to reduce the residual emission by subtracting an appropriate dark sky spectrum from each spaxel in the cube. There are several things to consider if attempting this:

• The dark observation used should be at least as long as the mapping observation to minimise added noise. It should also be close in time (and/or taken under similar observing conditions).

• Each spaxel may already contain spectra from several different detectors, so subtracting a single dark spectrum is unlikely to improve small-scale noise, as this changes for different detectors.

• It may be best to smooth the dark spectrum before subtraction, or to combine several detectors in the dark sky observation to smear out detector dependent effects.

• If there are problems with small-scale noise, it may be better to subtract the dark detector-by-detector before gridding.

The following script provides one way to subtract a single dark sky spectrum from a cube. Note that the dark and cube should be processed with the same version of HIPE:

```python
# Load the Observation Context
obs = getObservation(obsid, poolName)
# Extract the SSW cube
sswCube = obs.level2.getProduct("HR_SSW_cube")
# Extract the dark sky extended-source calibrated product
# From a sparse observation
darkObs = getObservation(darkObsids, poolName)
darkSpec = darkObs.level2.getProduct("HR_spectrum_ext")
# Subtract the dark SSWD4 spectrum from the SSW cube
subtractedCube = subtract(ds1=sswCube, ds2=darkSpec['0000']['SSWD4'])
```
7.9. Observations with few repetitions

2nd level deglitching can struggle to remove glitches from observations with fewer than 10 repetitions, due to low number statistics. One way to identify insufficient deglitching is by inspecting the QC report of an observation, to see if one or more of the relevant flags (outOfBandHighRatioSLW, outOfBandHighRatioSSW, outOfBandLowRatioSLW, outOfBandLowRatioSSW) have been raised. A second way, is to look for outlying scans in the spectral products, but this is only possible by re-running the Spectrometer User Pipeline (found in the HIPE Pipelines), while omitting the line that averages the scans:

```python
#extended = averageSpectra(extended)
```

Suppressing the averaging of all the scans per detector, means that for an extended calibrated spectral product, the SDS Explorer can be used to view all scans from all detectors to check for data consistency, as shown in Figure 7.63.

![Figure 7.63. All scans from all SSW unvignetted detectors for an example observation. The green scan in the centre detector (SSWD4) is an outlier.](image)

Due to the random nature of glitches, it is likely that not all detectors will be adversely affected. Depending on the science goals, it may not be necessary to fix the data in all detectors where a problem is identified, but note that off-axis detectors may still be relevant even for point-like sources, if a background subtraction is necessary (see Section 7.5). If you find that at least one relevant detector shows an outlier scan, then this scan can be removed or a correction attempted. A simple removal relies less on human judgement, however correcting the afflicted scan(s) retains the maximum number of independent data points.

7.9.1. Removing data

If choosing to remove outlying data, the first step is to manually identify the problem scan and its detector in the extended calibrated spectral product. An outlying scan can be identified by plotting all scans from the detector in question, which can be achieved using the SDS Explorer, as shown in
Figure 7.64. In the case of mapping observations, it is necessary to perform this visual inspection for each building block (jiggle position).

Figure 7.64 shows a plot from the SDS Explorer, which contains all of the information required to identify an outlying scan, i.e. Building block ID (the hexadecimal number in the centre of the title), the scan number (in the legend), and the detector name (also in the legend). In this case, the green scan is an outlier, i.e. scan number 3 for SSWD4 in building block 0xA1060003. For mapping observations, the scan has to be removed right after interferogram creation. For point source observations, the scan should be dealt with prior to phase-correction and after 2nd level deglitching. For example, to remove the identified data from the SpectrometerDetectorInterferogram (sdi) insert the following code into the User Pipeline scripts:

```python
# Identify the building block ID
glitchBbid = [0xA1060003]
# Identify the scan ID
glitchScan = [3]
# Identify the detector name
glitchDetector = ["SSWD4"]
# Remove the outlying scan
if bbid == glitchBbid :
    sdi[glitchScan].removeChannel(glitchDetector)
```

After executing the adjusted user script for the example observation, the resulting ssds contains only three scans for the detector SSWD4, which all agree within the noise.

### 7.9.2. Correcting data

Rather than removing the outlying scan from the building block completely, it may possible to fix the data by manually removing glitches that were missed by 2nd level deglitching. It is easiest to accomplish this by stopping the User Pipeline script (either single pointing or mapping) after the call to deglitchIfgm, e.g. by inserting a stop directly after this task is used. When viewing all scans of the detector in question, you will likely see outlying data in the central burst region of the interferogram (between -0.6 and 0.6 cm OPD) for the identified scan. It is easiest to first identify the residual glitch visually with the SDI Explorer. In the example discussed here, and shown in Figure 7.65, the outlying scan (3) is deviating from the other scans at an OPD of 0.025 cm.
Figure 7.65. All interferograms from detector SSWD4 in an area of the central burst region. The green line deviates from the other data.

Once the outlier has been visually identified, the exact indices of the relevant samples need to be determined. This is best done by inspecting the scan with the Dataset Viewer: Select the sdi variable in the Variables view; in the Outline view, navigate to the relevant scan ("0003" in this case) and click the plus symbol to drop down all detectors for this scan. Select the relevant detector (SSWD4 in this case) with a right-click and Open With > Dataset Viewer. Now scroll the data tables to the relevant OPD range (0.0225 - 0.0250 cm in this case) and the relevant index can then be read from the index column on the left hand side:

Figure 7.66. Data samples from detector SSWD4.

Make a note of the indices of the outlier data samples, in this case there are two relevant indices: 258 and 259, see Figure 7.66. Now that the precise locations of problematic samples are known, the masks of these data samples can be used to remove the offending data samples and replace them by the average of non-affected scans. Add the following lines after the createIfgm call to fix the affected data samples:
from herschel.spire.ia.util.SpireMask import GLITCH_SECOND_LEVEL
from herschel.spire.ia.util.SpireMask import GLITCH_SECOND_LEVEL_UNCORR

# Identify the building block ID (0xA1060003 in this case)
glitchBbid = [0xA1060003]
# Identify the scan id (3 in this case)
glitchScan = ["0003"]
# Identify the detector name ("SSWD4" in this case)
glitchDetector = ["SSWD4"]
# Identify the identified indices in an array ([258,259] in this case)
glitchIndices = [[258,259]]
# Fix the outlying samples
if bbid == glitchBbid :
    for idx in glitchIndices:
        sdi[glitchScan][glitchDetector].mask[idx] = GLITCH_SECOND_LEVEL.getValue()+GLITCH_SECOND_LEVEL_UNCORR.getValue()
        sdi = deglitchIfgm(sdi, identifyGlitches=False)

Figure 7.67. All interferograms from detector SSWD4 in an area of the central burst region after fixing scan 3.

Always make sure to inspect the resulting interferogram product, see Figure 7.67, and on running the remainder of the User Pipeline script, also inspect the resulting extended calibrated spectral product (extended), for example, see Figure 7.68.
7.10. Comparison with the SPIRE photometer

For any FTS spectrum, the HIPE task spireSynthPhotometry can generate synthetic photometry at the SPIRE photometer PSW (250 µm/1200 GHz), PMW (350 µm/857 GHz) and PLW (500 µm/600 GHz) wavebands. If there is a complementary SPIRE photometer observation, i.e. one overlapping with the FTS footprint, photometry can be extracted from the maps and input into the same task for easy comparison to the FTS synthetic photometry. The observation ID(s) for any SPIRE photometer map that overlaps with an FTS observation, can be found in the FTS Observation Context metadata and the metadata of the Level-2 products. The relevant keyword(s) take the form of photObsid###, where ### is a counter starting from 000 and ordered by the date of the photometer observations (from earliest to latest). If no overlapping photometer maps exist, then there will be no photObsid keywords in the metadata.

7.10.1. The FTS Synthetic Photometry task

The task spireSynthPhotometry generates synthetic FTS photometry at the three SPIRE photometer wavebands, with the PLW and PMW photometry points (at 600 and 857 GHz) taken from SLW spectra and the PSW point (at 1200 GHz) from SSW spectra.

spireSynthPhotometry requires the photometer relative spectral response functions (RSRFs), so a SPIRE Calibration Context must be provided. The photometer RSRFs are reasonably wide and although the PMW RSRF falls mainly in the SLW frequency range, a minor fraction does extend into the SSW band. If only the SLW frequency range is used to calculate the PMW synthetic photometry, there is a systematic loss estimated at ~2% (Hopwood et al. 2015). If there is good agreement between the SLW and SSW spectra in the overlap region, more accurate PMW synthetic photometry can be derived, by stitching the two spectra together and inputting the result into spireSynthPhotometry, as shown by the following example:

```bash
# Load the SPIRE Calibration Context
cal = spireCal()
```
# Extract a Simple Spectrum from an SDS, which will have two segments, # one for SLWC3 and one for SSWD4
spec = spireProduct2SimpleSpectrum(spss, detector=['SLWC3','SSWD4'])

# Join the SLWC3 and SSWD4 spectra together by averaging the overlap region
stitched = stitch(ds=spec, variant='average', unit='GHz',
                   avg_variant='wave-flag-weight')

# Take the synthetic photometry on the stitched results
syntheticPhotometry2 = spireSynthPhotometry(spectra=stitched, calibration=cal)

spireSynthPhotometry can be run from the command line (as shown above), or from the GUI, which can be opened from the HIPE Tasks menu. The task requires a minimum input of an FTS spectrum or a Level-2 point-source or extended-source calibrated product, and a SPIRE Calibration Context. The output is a TableDataset containing the synthetic photometry, with each detector in a separate row.

If doPlots is set to True, then the input spectrum (or the centre detectors for a Level-2 SDS or SSPS) are plotted in comparison with the synthetic photometry and the SPIRE photometer RSRFs. If photometer values from PSW, PMW and PLW maps are provided, these will be added to the plot and the output table.

## 7.10.2. Point source observations

The point-source flux density in the three bands of the SPIRE photometer can provide a useful piece of information on the accuracy of the spectrometer flux. Figure 7.69 shows the spireSynthPhotometry output plot for the input of a point-source calibrated SPSS. The example given is for a sparse HR observation of the point-like source CRL618 (observation ID 1342268302), with the associated PSW, PMW and PLW photometry provided to the task. See Section 6.9.1 for a description of extracting point-source fluxes from photometer maps.

![Synthetic Photometry](image)

**Figure 7.69.** Example plot generated by spireSynthPhotometry for a sparse HR observation of CRL618. The input spectra, calculated synthetic photometry and photometer values provided are all included. The SPIRE photometer RSRFs are also plotted by the task for reference.
The following commands run the task from the command line for the example observation of CRL618.

```python
# Load the SPIRE Calibration Context
cal = spireCal()
# Download the Observation Context for the sparse HR observation of CRL618
obs = getObservation(1342268302, useHsa=True)
# Extract the point-source calibrated product
spss = obs.refs['level2'].product.refs['HR_spectrum_point'].product
# Take the synthetic photometry and generate the plot
# comparing the synthetic and photometer photometry
syntheticPhotometry = spireSynthPhotometry(spectra=spss, calibration=cal, 
    psw=55.9, pmw=24.4, plw=11.8, doplots=True)
```

The derived synthetic photometry values follow the photometer convention of monochromatic flux densities, assuming a point-source with a power law spectrum of spectral index -1 (see the SPIRE Handbook, section 5.2.4 for more details). If the actual RSRF-weighted total in-band flux density is needed, then the synthetic photometry values must be divided by the K4P conversion factor (see Table 6.12 in the photometer section).

The following should be considered when performing such a photometry comparison:

- The calculation assumes a perfect point-source. For sources that are partially extended within the FTS beam, the synthetic photometry should not be taken until after the spectra are corrected for the source size using SECT (see Section 7.6). Note that for partially-extended sources a correction for the source shape is also necessary for the photometer pipeline photometry, see Section 6.9.1.9
- If the point-source is embedded in extended background (or foreground) emission, this emission must be removed before taking the synthetic photometry or comparing to photometry from a corresponding photometer map. Background subtraction is discussed in Section 7.5.2.
- To generate synthetic photometry for a point source located in an intermediate or fully sampled mapping observation, a spectrum must first be extracted at the position of the source using specPointSourceExtractor (see Section 7.12.3). This task provides a point-source calibrated spectrum in Jy, which can be input into spireSynthPhotometry.

### 7.10.3. Mapping observations

Note that all FTS mapping mode pipeline products are extended-source calibrated. Due to a correction for the extended-source calibration, which was introduced in HIPE 14, only FTS data processed with HIPE 14 should be used for scientific analysis. The following assumes HIPE 14 processed FTS data.

If a target has been observed in FTS spectral mapping mode and also mapped with the SPIRE photometer, the FTS cubes and photometer maps can be compared using spireSynthPhotometry. The task can accept spectra with extended-source calibration, i.e. spectra in units of \( \text{W m}^{-2} \text{Hz}^{-1} \text{sr}^{-1} \), which can either be extracted from a spectral cube pixel (spaxel) or directly from the pre-processed cube product (also known as the preCube), which collates the individual spectra ready for projecting into a cube. For extended-source calibrated data, spireSynthPhotometry calculates synthetic photometry following the method described in the SPIRE Handbook. The observed spectrum \( I_S(v) \) is integrated over the photometer passbands (PSW, PMW or PLW) as in Eq. 5.18 from Section 5.2.5. of the Handbook, using the empirical photometer beam variation with frequency (see Eq. 5.38) and the photometer RSRFs. The resulting RSRF weighted flux is then multiplied by the conversion factor KMonE (see Table 6.9 in the photometer Section 6.9.1.2) to give the monochromatic intensity for a source with a power law spectrum of spectral index -1. The output synthetic photometry has units of MJy/sr.

The following example illustrates the use of spireSynthPhotometry for spectral maps, by showing a direct comparison of the synthetic photometry from a single spectrum (extracted from the pre-Cube before the spectra are projected into a cube) and the value of the photometer map in the pixel where the spectrum falls. A fully sampled HR spectrometer map of NGC7023 (the Iris nebula) is used for the example, with observation ID 1342198923. There are five Level-2 photometer maps where
NGC7023 is within the FTS field of view, including Level-2.5 and Level-3 maps. As the observations were performed in SPIRE-PACS parallel mode, we use the Level-2.5 map from observation 1342188653.

The first step is to load the SPIRE Calibration Context and download the extended-source calibrated photometer Level-2.5 map from the Herschel Science Archive. From HIPE version 13, the Planck zero-point correction is already applied to photometer data in the archive.

```
# Load the SPIRE Calibration Context
cal = spireCal()
# Download the NGC7023 Photometer observation
obsP = getObservation(1342188653, useHsa=True, instrument='SPIRE')
# Extract the PSW and PLW Level-2.5 photometer maps and WCS
map250 = obsP.level2_5.getProduct("extdPSW")
wcs250 = map250.wcs
map500 = obsP.level2_5.getProduct("extdPLW")
wcs500 = map500.wcs
```

The next step is to load the spectrometer preCubes, which are listings (in the form of a Spectrum2D) of all extended-source calibrated spectra and their sky positions. There is one pre-Cube per detector array.

```
# Download NGC7023 fully sampled HR mapping observation
obsS = getObservation(1342198923, useHsa=True)
# Extract the preCube for each detector array
preCubeS = obsS.level2.getProduct("HR_SSW_spectrum2d")
preCubeL = obsS.level2.getProduct("HR_SLW_spectrum2d")
```

Now select one detector from SSW and one from SLW, and check the number of extended-source calibrated SpectrometerDetectorSpectrum products (SDS). The associated spectra will fall on different pixels on the photometer maps.

```
# All SDS for detector SSWD4
specS = preCubeS.extractSpectra("SSWD4")
# All SDS for SLWC3
specL = preCubeL.extractSpectra("SLWC3")
# Get the number of SDS. There should be 16 for full sampled map,
# 4 for intermediate.
spec = len(specS)
print spec
```

The final step is to take and compare the photometry at each jiggle position.

```
# Set up arrays to store the results
synth250 = Double1d(nspec)
synth500 = Double1d(nspec)
phot250 = Double1d(nspec)
phot500 = Double1d(nspec)
# Take the PLW and PSW photometry
for i in range(nspec):
    # PLW
    # Obtain the synthetic photometry
    result = spireSynthPhotometry(specL[i], calibration=cal)
synth500[i] = result["spec500"].data[0]
    # Extract (rad,dec) for the spectrum
    ra = specL[i].meta["ra"].double
dec = specL[i].meta["dec"].double
    # now take the nearest pixel from the photometer
    pix = wcs500.getPixelCoordinates(ra,dec)
    row = int(ROUND(pix[0]))
column = int(ROUND(pix[1]))
    phot500[i] = map500.image[row,column]
print "Jiggle: %i, SLWC3, phot = %f, spec = %f (MJy/sr), ratio = %5.1f %%",(i,phot500[i],synth500[i], 100.0*(1.0 - phot500[i]/synth500[i]))
    #PSW
    ra = specS[i].meta["ra"].double
dec = specS[i].meta["dec"].double
    result = spireSynthPhotometry(specS[i],calibration=cal)
```

260
```python
# now from the photometer, the nearest pixel
row = int(ROUND(pix[0]))
column = int(ROUND(pix[1]))
# print "Jiggle: %i, SSWD4, phot = %f, spec = %f (MJy/sr), ratio = %5.1f %%"
#(i,phot250[i],synth250[i], 100.0*(1.0 - phot250[i]/synth250[i]))
```

And we can plot the results, to produce Figure 7.70:

```python
plt = PlotXY()
l1 = LayerXY(Double1d.range(phot250.length()),phot250/synth250)
l1.setLine(Style.NONE)
l1.setSymbol(Style.FCIRCLE)
l1.setColor(java.awt.Color.BLUE)
l1.setName('SSWD4 250 µm')
l2 = LayerXY(Double1d.range(phot500.length()),phot500/synth500)
l2.setLine(Style.NONE)
l2.setSymbol(Style.FSQUARE)
l2.setColor(java.awt.Color.RED)
l2.setName('SLWC3 500 µm')
plt.addLayer(l1)
plt.addLayer(l2)
plt.xaxis.title.text = 'Jiggle position'
plt.xrange = [-1.0,16.0]
plt.yaxis.title.text = 'Ratio Phot/Spec'
plt.yaxis.getTick().setGridLines(True)
plt.getLegend().setVisible(True)
```

The results of this example, shown in Figure 7.70 show some discrepancies: +20%, -10%. This is expected because NGC7023 has significant structure with strong surface density gradients. The beam coupling with the source must be properly modelled on both sides, the spectrometer and the photometer, and taken into account for a proper comparison (see Section 6.9.1.9, the SPIRE Handbook, Griffin et al. 2013, section 6.2.4 and Fig. 16).
There are other possibilities for comparing spectrometer and photometer data. For example, performing an aperture extraction, centred on a region of interest, in both the photometer and spectrometer maps. Another possibility is to make the photometer maps with the spectral cube WCS and then perform the comparison. If any of these alternative methods are of interest then please contact the Herschel Science Centre Helpdesk for assistance.

7.11. Spectral Analysis

To build on the the generic Herschel spectral analysis described in Chapter 5 of the Herschel Data Analysis Guide, this section focuses on analysing FTS data using tools available in HIPE. Many of these tools can be applied to a full Spectrometer Point Source Spectrum (SPSS) or Spectrometer Detector Spectrum (SDS), but all will accept an individual spectrum as input. Individual FTS spectra are the data from a single detector, and are contained in SpireSpectrum1d datasets (see Chapter 3 of the Herschel Scripting Guide for general details of Herschel spectral datasets). Each FTS spectrum contains columns for the frequency scale (wave), the calibrated signal (flux) and associated error (error), masks (mask) and number of scans that have been averaged (numScans). They may also contain columns for weights and flags, however these are not populated by the standard FTS pipelines.

7.11.1. Local Standard of Rest

The FTS pipeline corrects the frequency scale from the satellite frame to the Local Standard of Rest (LSR) using the velocity of the Herschel spacecraft, which is given with respect to the LSR in the metadata (see Table 7.7 and Section 7.3.29 for details on the correction). For high resolution (HR) mode, the spectral resolution is larger than the correction to the LSR. However, it is still important to correct HR data, as the central frequency of significant spectral lines can be determined with an accuracy much smaller than one spectral resolution element.

Prior to HIPE version 12, the FTS pipeline did not convert the frequency scale to the LSR frame. It is possible to check whether the LSR correction has been applied to a pipeline product by examining the metadata item freqFrame. For corrected data, this will be set to a value of LSRk. In data processed with HIPE version 11 and before, the metadata item may not exist at all, or will be set to a value of TOPOCENT to indicate the correction has not been applied.

### Table 7.7. Information about the spacecraft velocity along the line of sight.

<table>
<thead>
<tr>
<th>HIPE product metadata name</th>
<th>FITS header variable</th>
<th>Example value</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>velocityDefinition</td>
<td>VELDEF</td>
<td>'RADI-LSR'</td>
<td>The velocity definition and frame</td>
</tr>
<tr>
<td>radialVelocity</td>
<td>VFRAME</td>
<td>21.08429538498 [km/s]</td>
<td>Spacecraft velocity along the line of sight of the telescope wrt the local standard of rest: ( v = (\lambda_{\text{rest}} - \lambda) / \lambda_{\text{rest}} )</td>
</tr>
<tr>
<td>freqFrame</td>
<td>SPECSYS</td>
<td>'TOPOCENT' or 'LSRk'</td>
<td>Frame of reference the frequency scale refers to</td>
</tr>
</tbody>
</table>

The HIPE task to convert the frequency of an FTS spectrum to the LSR frame or back to the satellite frame is called applyRadialVelocity, which uses the satellite radial velocity, stored in the radialVelocity metadata item. The task is applicable to SDS or SPSS products, e.g. for an SDS with variable name sds:

```python
# To convert to the LSR frame
sdsLsr = applyRadialVelocity(sds, targetFrame="lsr")
# To convert back to the satellite frame
sds = applyRadialVelocity(sdsLsr, targetFrame="satellite")
```

The task applies the following equation to convert from satellite to LSR frame:

\[
\text{frequency}_{\text{Lsr}} = \text{frequency} \times \left(1.0 - \frac{\text{radialVelocity}}{c}\right)
\]
7.11.2. Joining SLW and SSW data

The SPIRE spectrometer has two detector arrays, SLW and SSW, which consist of 19 and 37 detectors. With the SLW and SSW data overlapping in observed frequency from 944 to 1018 GHz.

When considering joining a pair of SLW and SSW spectra (either point-like or fully extended) only well-calibrated data from the two centre detectors should be considered. That is, only when there is not significant jump between the bands. The centre detectors are the best aligned pair, whereas for the remaining coaligned detector pairs, the SLW detector is slightly offset from the centre of the corresponding SSW detector.

The SLW and SSW from off-axis detector pairs should be analysed separately. However, even for the centre detector pair, it is important to visually examine the SSW and SLW data within the overlap region to ascertain if there is a good enough agreement between the two bands to join the spectra together. If a significant step between the spectra is present, it may be possible to correct the data to reduce this issue with further processing. See Section 7.6 for details of the possible causes and fixes for flux discrepancies between the bands.

A single contiguous Spectrum1d can be created using the Stitch tool from the Spectrum Toolbox. The following commands will join the point-source calibrated data from the centre detectors from an Observation Context obs, by taking the mean in the SLW/SSW overlap region:

```python
pointSpec = obs.level2.getProduct("HR_spectrum_point")
# note that the syntax for data processed with HIPE version 12.1 and before is:
#           pointSpec = obs.level2.getProduct("HR_unapodized_spectrum")
simpleSpec = spireProduct2SimpleSpectrum(pointSpec, detector=['SLWC3','SSWD4'])
stitchedSpectrum1d = stitch(ds=simpleSpec, variant='average', unit='GHz',
                          avg_variant='wave-flag-weight')
```

7.11.3. Visualising SPIRE spectra

To examine the spectra from one or more detector arrays, a full FTS SDS or SPSS can be opened with the SDS Explorer. Figure 7.71 shows the SDS Explorer view after opening a point-source calibrate pipeline product and the plot generated when detectors are selected in the GUI. See Section 8.1 for full details on the options available for the SDS Explorer.

Figure 7.71. A full SPSS product opened in the SDS Explorer (left) and the plot produced (right) after selecting several detectors from the "honeycomb" display of the FTS detector arrays. If additional products are opened with the SDS Explorer, any detectors selected will be added to the same plot. There are options to manage the plot or plot thumbnails in the lower panels. And the metadata can be examined by selecting the "Meta Data" tab."
The "Spectrum Explorer" is a generic HIPE tool for viewing any Herschel spectrometer data. The Spectrum Explorer can be used to view the spectra within an FTS SDS or SPSS product by right-clicking on the necessary product in the variables tab and choosing Open With > Spectrum Explorer, as shown in Figure 7.72. Individual detectors can then be plotted by selecting them in the lower left panel of the Spectrum Explorer view. Alternatively, the spectra from an individual (a SpireSpectrum1d) can be extracted from the FTS product and opened on its own. Extracting a single spectrum can be achieved using the SDS Explorer, by dragging a detector from the "honeycomb" display to the variables panel (as explained in Section 8.1), or from the Product Viewer window. In the Product Viewer, extend the "0000" scan number folder right-click on a detector name and select Open With > Spectrum Explorer. If the data is unaveraged, there will be a folder for each scan, numbering from "0001". Additional detectors can be added to the plot, by dragging and dropping them from the folder into the plot window.

Opening a spectrum or FTS product with the Spectrum Explorer results in two panels, the plot panel at the top of the view and a data selection panel to the bottom. A preview panel will open to the bottom right once data is selected to be added or removed from the plot. A range of options are available via the icons along the toolbar above the top panel, including the Spectrum Toolbox (crossed hammer and spanner icon) and the Spectrum Fitter GUI (line profile icon over a crossed hammer and spanner). Chapter 5 of the Herschel Data Analysis Guide details the set of toolbox operations that are provided for any spectrum measured by the spectrometers onboard Herschel.

Figure 7.72. A full SDS product open in the Spectrum Explorer (left) and the right-click route to open a single SpireSpectrum1d with the Spectrum Explorer (right).

7.11.4. Spectral fitting

Spectral fitting can be performed in HIPE with the Spectrum Fitter GUI (see Chapter 7 of the Herschel Data Analysis Guide and Section 7.11.8) and with the Spectrometer Line Fitting SPIRE Useul script, which is accessible via the HIPE Scripts > SPIRE Useful scripts menu (see Section 7.11.9). Better results are usually obtained when fitting to the continuum and spectral lines simultaneously, rather than fitting lines individually. The instrument line shape is well described by a sinc profile, while a low order polynomial provides a good fit to the continuum in most cases (as described in Hopwood et al. (2015)). A modified black body function may provide a better fit to the continuum, and although there is no such function available in HIPE, one could be written following Section 5.8 of the Herschel Scripting Guide.

7.11.5. Sinc model

The natural FTS instrument line shape can be well approximated by a cardinal sine function (i.e. the sinc function), as illustrated by Figure 7.75. The sinc model used by HIPE is:
where $p_0$ is the amplitude at the central frequency of the line, $p_1$ is the central frequency, and $p_2$ is the distance between the peak and the first zero crossing, i.e. the sinc width. The width is defined as

$$p_2 = \frac{\Delta \sigma}{\pi}$$

where $\Delta \sigma$ is the width of one independent resolution element in the spectrum, which depends on the actual maximum OPD (optical path difference) visited by the FTS mirror (see Section 7.1.1). For HR mode, the spectral resolution of the reduced data is usually the optimum of 1.18 GHz, but can be larger if an interferogram for a particular detector has been truncated in the pipeline, e.g. due to clipping at high OPD. The full width at half maximum (FWHM) of a sinc function can be calculated as

$$\text{FWHM} = 1.20671 \times \Delta \sigma$$

When fitting a sinc to an unresolved HR line, $p_2$ can be fixed at $\Delta \sigma/\pi$, which in the best spectral resolution case is 0.38 GHz.

To calculate the integrated line flux (in W m$^{-2}$) from the sinc fit to a point-source calibrated spectrum, with flux density units of Jy and line width in GHz, the following equation can be used:

$$I = \int f(x : p) = 10^{-26} p_0 10^9 \pi p_2$$

The FTS instrument line profile is not a perfect sinc shape, see Swinyard et al. (2014), Hopwood et al. (2015), and the SPIRE Handbook. Asymmetry in the line profile is most evident in the first side lobe on the high-frequency side, which can be seen in the example line provided by Figure 7.75. The presence of this asymmetry translates to the integrated line flux when calculated from fitted sinc parameters, with the line flux suffering a systematic shortfall of 2.6%. This systematic uncertainty could be folded in with the fitting error, or used to correct the measured line flux.

### 7.11.6. Gaussian model for apodized data

If the extended Norton-Beer 1.5 apodization function has been applied to the data, spectral lines are reasonably well approximated by a Gaussian (see Figure 7.76). The integrated line flux from such a fit has an additional systematic error, due to the apodization of the asymmetry seen in the instrumental line shape. The recovered line flux is around 5% higher than the equivalent line flux recovered from a sinc fit to the standard data, which translates to a 2.4% systematic overestimate to that expected without any asymmetry present (see Hopwood et al. 2015). Although apodized data can be used to identify lines, the apodization folds spectral noise into the lines. To ensure accurate results, line measurements should be taken from data that has not been apodized.

The Gaussian model used by HIPE is:

$$f(x : p) = p_0 \exp \left( -\frac{(x - p_1)^2}{2 p_2^2} \right)$$
where $\rho_0$ is the amplitude at the central frequency of the line, $\rho_1$ is the central frequency, and $\rho_2$ is the width (i.e. FWHM/$2 \sqrt{2 \ln 2}$). For the standard apodization function applied in the pipeline, the extended Norton-Beer 1.5, the FTS spectral resolution is degraded by a factor of 1.5 (see [Naylor Tahic 2007]), and so for an unresolved line, $\rho_2$ could be fixed to

$$\rho_2 = 1.5 \times 1.20671 \times \frac{\Delta \sigma}{2 \sqrt{2 \ln 2}}$$

where $\Delta \sigma$ is the width of one independent resolution element, defined in the previous section. The FWHM of an unresolved apodized line is at best 2.15 GHz.

When working with apodized data, the approximate integrated flux of the emission line can be calculated from the parameters derived from the Gaussian fit. Assuming flux density units of Jy, and line width units of GHz, the integrated flux in W m$^{-2}$ is:

$$I = \int f(x : p) = 10^{-26} \rho_0 10^9 \sqrt{2\pi} \rho_2$$

### 7.11.7. Partially resolved lines

Extragalactic sources can have broad intrinsic line profiles and therefore the resulting spectral features may be partially resolved by the FTS. In such cases a convolution of a sinc function and Gaussian function may better describe the line shape. If a sinc fit to a line is clearly too narrow, the line may be partially resolved, and the sincGauss model in HIPE can be used to fit instead of a sinc. But note that noise and blending can also lead to line broadening.

The sincGauss model used by HIPE is:

$$f(x : p) = \rho_0 \exp\left(-b^2\right) \left(\frac{\text{erf}(a - ib) + \text{erf}(a + ib)}{2 \text{erf}(a)}\right)$$

where

$$a = \frac{\rho_3}{\sqrt{2} \rho_2}$$

and

$$b = \frac{x - \rho_1}{\sqrt{2} \rho_3}$$

and $\rho_0$ is the amplitude at the central frequency of the line, $\rho_1$ is the central frequency, $\rho_2$ is the width of the sinc function ($\Delta \sigma / \pi$; see Section 7.11.5) and $\rho_3$ is the width of the Gaussian function (FWHM/$2 \sqrt{2 \ln 2}$; see Section 7.11.6). The analytical expression for the sincGauss function above
was derived by multiplying the Fourier transforms of the sinc and Gaussian functions and taking the inverse Fourier transform. See the URM SincGaussModel entry for more details.

Assuming flux density units of Jy and line width in units of GHz, the integrated line flux, in units of W m\(^{-2}\), can be calculated from the fitted sinc-Gauss parameters as

\[
I = \int f(x : p) = 10^{-26} p_0 10^9 \frac{\sqrt{2\pi} p_3}{\text{erf} (p_3/\sqrt{2} p_2)}
\]

Figure 7.73 can be used in order to determine whether it is worthwhile (or appropriate) to use a sinc-Gauss function rather than a sinc function. The ratio of the integrated line flux for the two functions (with their amplitudes set equal) is plotted for lines of different intrinsic widths. The intrinsic width was set to 100, 200, 300 and 400 km/s and used as the width of the Gaussian part of the sinc-Gauss function. The sinc width was fixed to the instrument resolution, as described in Section 7.11.5. The plot shows that a sinc-Gauss profile is not needed for lines that are 100 km/s wide, but makes a significant difference to the integrated line flux when the width is 400 km/s.

Figure 7.73. Ratio of the integrated line flux in sinc-Gauss to sinc function profiles for various intrinsic source widths.

The HIPE implementation of the sinc-Gauss function cuts off abruptly in the wings, with the cutoff moving closer to the central peak as the line width decreases. However Figure 7.74 shows this effect only becomes significant for lines with widths of 200 km/s and narrower. The integrated line flux recovered from the fitted parameters is not affected by this issue, but if fitting multiple spectral features in the same spectrum, then the wing sections beyond the cutoff will not be well fitted, which could lead to a poorer overall fit, and hamper the fit and identification of faint features close to a significant partially resolved line. Taking this issue into consideration, along with Figure 7.73, it is better to only use the sincGauss function for lines with a width greater than 200 km/s.
Figure 7.74. Normalised sinc$\text{Gauss}$ profiles with various widths. There is a cutoff in the wings of the function that gets closer to the central peak as the line width decreases.

### 7.11.8. Interactive line fitting using the Spectrum Fitter GUI

The functionality and use of the Spectrum Fitter GUI (SFG) is described in detail in Chapter 7 of the Herschel Data Analysis Guide. The SFG is intended as a user-friendly interface that allows immediate viewing of fitting results. For FTS products, line fitting from a script gives greater control over the models and fitting parameters, allows easier examination and manipulation of the fitting results, and may be less time consuming. However using the SFG with a single spectrum can be useful for a quick test, or to gain a feel for HIPE line fitting.

Before opening the SFG, there are two FTS appropriate options that can be considered in the HIPE preferences, which can be found following the menus: *Edit > Preferences > Editors & Viewers > SpectrumFitterGUI*. The "Default Model" can be set to `sinc` for standard data and to `gauss` for apodized data. The default initial width of the model can also be set.
Figure 7.75. The Spectrum Fitter GUI with a polynomial and multiple sinc profile model fitted to a point-source calibrated spectrum. One line, the corresponding best fit model and the model subtracted residual are shown in the left panel to illustrate the line asymmetry present in FTS lines and how this compares to a fitted sinc profile.

Figure 7.76. The Spectrum Fitter GUI with a polynomial and multiple Gaussian profiles fitted to an apodized point-source calibrated spectrum. The close-up of one fitted line is shown in comparison to the fitted model and model subtracted residual. The apodized line asymmetry is highlighted by the fitted Gaussian profile.

As shown by Figure 7.75 and Figure 7.76, initiating the Spectrum Fitter GUI for a SpireSpectrum1d opens four panels in the HIPE window: a plot of the input spectrum to the top left; a data and fit results information panel to the bottom left; a preview panel to the bottom right; and the main options panel, where the model is set up and the fitter run, which can be found at the top right with the title Spectrum Fitter GUI.

The Spectrum Fitter GUI panel can be used to add multiple model profiles, so the continuum and spectral lines can be fitted simultaneously. A new model can be added in the Models tab, by clicking addModel and then selecting the desire one from the pulldown list in the new parameter panel. A polynomial can be selected to fit to the continuum, but will be generated with the default order of zero. This order should be changed to second or third order, which should be sufficient to fit the continuum of an FTS spectrum. Note that Update must be pressed to action the change. Sinc models can be added for each spectral line present, again using the addModel button, and then selecting Sinc. Each new model is added to the plot and its information to the FitResult table in the bottom left panel. The initial peak amplitude and position for each sinc model can be set by clicking on either the Amplitude or center parameters, both of which will turn yellow. Once yellow the cursor can be used to click the
plot on the position of the line to be fitted, which will set the initial amplitude and position parameters of the sinc. To the right of each parameter and associated standard deviation boxes, are two boxes that allow parameter limits to be set, and a final box to the right of these, which allows the respective value to be fixed. The sinc width should be fixed, as discussed in Section 7.11.5. Once all the desired models have been added, clicking on the Accept button will initiate the fitting iterations. The resulting total model and residual are added to the plot window once the fit has converged, as shown in Figure 7.75 and Figure 7.76).

To achieve a good fit for an extended-source calibrated spectrum, the fitting tolerance must be changed to a smaller number than the default of 0.01, as the data is of the order ~10^{-17}. The Fitter Tolerance can be found in the Engine tab, and should be set to around 10^{-40}, to ensure the fit will converge on the expected solution, otherwise the fit will likely be deemed satisfactory by the fitting engine, while the model is still significantly offset from the data. The fitting engine can also be selected in the same tab, and the maximum number of iterations changed.

There are options for exporting the fitting results from the Export tab panel. One option is to save a script that will recreate the fitting result when run on the command line. The residual data and the total or individual components of the model can be exported in ASCII format.

The MultiFit tab is discussed further in Section 7.12 for use with cubes, but can be applied to an SDS or SPSS. If the Spectrum Explorer is opened with one of these products and a single detector selected, the SFG can be used to fit the spectrum in that detector, before using the best fit results as a starting point to fit to all detectors, via the MultiFit tab. This could be useful for subtracting a baseline from all detectors present.

7.11.9. SPIRE Spectrometer Line Fitting script

The "Spectrometer Line Fitting" script is one of the SPIRE Useful scripts provided in HIPE, and can be accessed from the Scripts > SPIRE Useful scripts menu on the toolbar. This script takes an extended-source or point-source calibrated spectrum as input and simultaneously fits the continuum and the specified spectral lines using the SpectrumFitter. The script uses a default input of the 12CO line frequencies, but for optimum results a catalogue of all strong lines present in the spectrum should be used. If a source velocity, or redshift for extragalactic sources, are provided, these are used to improve the initial line position guess for the fit.

The script adds a sinc profile for each line being fitted, i.e. the lines are assumed to be unresolved. With the sinc width fixed to the actual resolution of the detector spectrum (as recorded in the actualResolution metadata item) divided by π (see Section 7.11.5). Fitting is performed on each of the centre detectors, with a plot generated per detector to compare the input data, total model, and residual, see Figure 7.77. The fitted line parameters and associated errors are printed to the HIPE console window, but can also be written to a text file, if an output path is provided.

The script is intended as a demonstration of simultaneous fitting for SPIRE FTS data, and can be edited to:

- alter the spectral line list for fitting – The script uses the frequencies of the 12CO lines, but more lines can be added or alternate lines used. Rest-frame line frequencies, and their identifying names, can be added to the lineFreq and lineNames lists directly. Redshifted lines (included those identified in the data or residual) should be added to the lineFreqsCorr and lineNames lists following the commented out code given on lines 157 and 158 of the script. The line names and frequencies can also be read from an ASCII catalogue file. There is an example given in the script that shows how to use a space separated ASCII file to input the lines, for a file with two columns containing the name lines and rest-frame frequencies. The columns will be read in as lineNames and lineFreqs.

- change the model being fitted to the continuum – The default model is a 3rd order polynomial. This can be changed before the loop over detectors.

- change the detector selection – The script only fits the centre detectors, SLWC3 and SSWD4, but other detectors could be added, or a single detector specified.
• fix the model parameters – by default only the sinc width is fixed, but see Section 7.9 of the Herschel Data Analysis Guide for a description of how to fix other parameters in the fit.

• limit the model parameters – Section 7.22 of the Herschel Data Analysis Guide describes how to set limits on fit parameters, such as the line frequency.

• change the fitting engine – The Levenberg Marquardt fitting algorithm is used in the script. Section 7.25 of the Herschel Data Analysis Guide describes other available algorithms.

• alter the fitting tolerance – The script sets a tolerance for the minimisation to match the typical dynamic range of extended-source or point-source calibrated data. This is needed in order to obtain a good fit, especially for extended-source calibrated spectra. Setting tolerances for the fitter is described in Section 5.8 of the Herschel Scripting Guide.

• select a different model to fit to the spectral lines – The standard function fitted to each line position is a sinc profile, using the following command:

\[
m = \text{sf.addModel('sinc', sincParamGuess)}\]

The model can be changed to a Gaussian (\texttt{gauss}) for apodized data, or a sinc-Gauss model (\texttt{sinc-Gauss}) for partially resolved lines (see Section 7.11.7). For the \texttt{sincGauss} function the sinc width should be fixed, with the initial parameter for the Gaussian width added to the initial guess, as shown in the following example.

\[
\# \text{Initial sincGauss parameters: [peak, line frequency, sinc width, Gaussian width]} \\
sincParamGuess = [\text{initialAmplitude}, \text{line}, \text{spectralResolution/pi, 1.0}] \\
m = \text{sf.addModel('sincGauss', sincParamGuess)}
\]

Note that if the model is changed to \texttt{gauss} or \texttt{sincGauss}, then the equation to calculate the integrated line flux and associated error must also be changed (towards the end of the script). For example, for a \texttt{sincGauss} profile, the equation in Section 7.11.7 should be used:

\[
\text{lineFlux} = \text{sdsUnit.toSI}^\ast\text{parameters[0]}\ast1e9\ast\text{SQRT(2*pi)}\ast\text{parameters[3]}/\text{ERF(\text{parameters[3]}/\text{SQRT(2)}/\text{parameters[2]})}
\]

Figure 7.77. Example of the plots produced by the Spectrometer Line Fitting script.

There is an IDL line fitting tool available (the \texttt{FT_Fitter}), which has been written to work with FTS data. More details on this tool can be found via the Herschel User Contributed Software page.

### 7.11.10. Calculating line fitting errors

The HIPE spectrum fitter provides the standard deviations on the fit parameters. In the line fitting script these are used to calculate the line flux error. However, this is likely to be an underestimate. A
more realistic way to calculate the line flux error is to use the residual spectrum around the position of the line and measure the bin-to-bin spectral noise (see Section 7.5.3). All strong spectral features should be fitted and subtracted before using the residual for this estimate. For more information on the fitting mechanics and how it calculates errors, see the Fitter Reference Document.

7.11.11. Line identification

When identifying lines in SPIRE FTS spectra, it is important not to wrongly identify the side lobes of the sinc function as spectral lines. A good approach to determine if a particular feature is real or not is to cross-reference with the corresponding apodized spectrum. Although the amplitude of the lines are reduced and the spectral resolution degraded, the side-lobes are less prominent in apodized spectra, making it less likely for lines to be wrongly identify, and potentially making it easier to separate blends and weak features in the standard data. In future versions of HIPE, tools may become available to help with line identification and labelling.

7.11.12. Joining SPIRE and PACS point spectra

SPIRE and PACS were developed as complementary instruments. PACS covers the shorter wavelength range up to 200 µm, with SPIRE covering the longer wavelength range starting from 200 µm.

The SPIRE Useful script “Combine PACS and SPIRE spectra”, gives an example of how to combine PACS and SPIRE spectra of point sources in order to create a single SED. The script creates a combined PACS spectrum and a combined SPIRE spectrum, and then combines these into a single Spectrum1d output. The combined output can be viewed in the Spectrum Explorer, as shown in Figure 7.76 for the example of CRL618 provided in the script. See Section 6.8 of the Herschel Data Analysis Guide for further details.

![Figure 7.78. The combined SPIRE/PACS Spectrum1d produced by the Combine PACS and SPIRE spectra Useful script, for the example source CRL618. From left-to-right, the segments are SPIRE SLW, SPIRE SSW, PACS R1, PACS B2B, PACS B2A. The log-log scale highlights the good agreement between the instrument calibrations.](image)

7.11.13. Comparing SPIRE and HIFI spectra

HIFI spectra have a much higher spectral resolution than SPIRE, so before comparing spectra from these two instruments, the HIFI data could be convolved with the SPIRE spectral response function. There is a simple example script of how to approach this provided in HIPE, which can be found in the Scripts > SPIRE Useful scripts menu on the toolbar, and is called “Spectrometer Convolve Spectrum”.

272
The script starts with high resolution data in the format of a `SimpleSpectrum` or a `Spectrum1d`. It then convolves with a sinc function that matches the high resolution data from SPIRE. Note that the kernel has a restricted range, so it doesn't reproduce the wings far from a spectral line.

### 7.12. Cube Analysis

The final product of the standard FTS pipeline for intermediate or fully sampled mapping mode observations are spectral cubes. These take the form of `SpectralSimpleCube` objects, which are described in Section 3.5 of the Scripting Guide. Optimising the pipeline steps involved in creating an FTS spectral cube is covered in Section 7.8, while this section describes FTS cube analysis. For general cube analysis see Chapter 6 of the *Herschel* Data Analysis Guide.

A correction for the extended-source calibration (as explained in Valtchanov et al. 2016), was introduced in HIPE version 14.0. It is essential for all mapping observations to be updated to HIPE version 14 processed data. If the observation in the *Herschel* Science Archive has been reduced with HIPE version 14, it can be downloaded directly. If the version is HIPE 13, then use the on-demand reprocessing option or run the HIPE 14 *User Pipeline mapping script* to update the data.

Another notable improvement to the calibration, which affects spectral cubes, is a correction for the SLW spectra of low resolution (LR) observations. This was introduced for point-source calibrated data in HIPE version 14.0, and then updated to correct LR extended-source calibrated spectra and cubes in HIPE version 14.1. As the correction was derived using point-source calibrated data, it does not extend to the vignetted detectors. These detectors are included in all LR pre-processed cubes (described below), but omitted when gridded into the SLW spectral cube. Therefore, as of HIPE 14.1, the SLW cubes of LR observations contain lower systematic noise, but also have reduced area. This includes the LR SLW cubes in a H+LR observation, and so these do not match the area of the associated HR SLW cubes.

Spectral cubes can be inspected in HIPE with the *Spectrum Explorer* and analysed using the *Cube Toolbox* (as explained in Sections 6.6 and 6.7 of the *Herschel* Data Analysis Guide). FTS spectral cubes can be opened with the *Spectrum Explorer* by right-clicking on the product in the Observation Context Viewer and selecting *Open With > Spectrum Explorer*. An example of an FTS spectral cube opened with the *Spectrum Explorer* is given by *Figure 7.79*, where the *naive* projected SSW cube from the Orion Bar observation 1342204919 is displayed. Options to modify the display colours and cut levels are provided when right-clicking on the cube image. An individual spectrum will be added to the plot in the top panel when a spectral pixel (spaxel) is selected with a click, making sure the "+" button shown in *Figure 7.79* is highlighted green. The *Cube Toolbox* is opened by clicking the hammer, spanner and cube icon located on the top bar of the *Spectrum Explorer* window.
Figure 7.79. Viewing the naive projected SSW cube from the Orion Bar observation 1342204919 in the Spectrum Explorer. One frequency layer is displayed, which corresponds to the red line in the top plot and the bar to the bottom of the display panel, where the layer selected can be changed. There are several options for spaxel selection. Pressing the plus box will allow one or more single spaxels to be selected. Rectangular, circular and line regions can also be selected. The corresponding error or coverage layer can be displayed via the pulldown menu to the bottom right of the cube display window. Additional cubes can be added for comparison by dragging them onto the top plot panel. The Cube Toolbox can be opened by clicking the cube, hammer and spanner icon, which provides a number of tasks that can be applied to an FTS cube.

The aim of the Cube Toolbox is to provide tasks in HIPE to extract 1D, 2D or 3D data from a Herschel cube product. The most relevant tasks for FTS cubes are:

- extractRegionSpectrum – for extracting a 1D spectrum over a selected or specified region.
- cropCube – for extracting a sub-cube in any of the three dimensions.
- subtractBaselineFromCube – for fitting and subtracting a baseline from each cube spaxel.
- integrateSpectralMap – for extracting a SimpleCube containing the image(s) for the range(s) specified. The resulting cube contains an integrated map for each range defined, which are detailed in integratedMap['LayerInfo']. The map in layer i can be accessed as a SimpleImage using map_i = integratedMap.getSimpleImage(i)

There are other Cube Toolbox tasks that are less relevant for FTS cubes, such as computePVMap, which can be used for creating position-velocity diagrams. The Herschel Data Analysis Guide should be the first port of call for all basic information on Toolbox tasks.

FTS spectral cubes and ways to manipulate and analyse them are discussed over the next few sections. Section 7.12.2 explains how to extract a single spectrum from a cube. Section 7.12.3 explains how to extract a point source from a mapping observation. Section 7.12.4 explores matching spectra from SLW and SSW cubes. Section 7.12.5 details convolving a cube to a different beam size and discusses considerations when comparing different cubes or the layers within a single cube. Section 7.12.6 and Section 7.12.7 look at fitting spectral lines and the continuum in FTS spectral cubes, which includes
details on the Spectrometer Cube Fitting script available in HIPE. Plotting the data from cubes is discussed in Section 7.12.8. And comparison to photometer maps covered by Section 7.12.9

7.12.1. SPIRE FTS cubes

Two sets of spectral cubes are produced by the FTS pipeline and stored in Level-2 of the Observation Context. These sets are divided by the projection algorithm used to create them, either naive or convolution. See Section 7.8.5 for information on all projection algorithms available for FTS mapping observations. The pipeline generates separate cubes for each of the FTS detector arrays, with pixel size set according to the average beam size for each array, i.e. with an array dependent WCS. Attempting combined analysis of SLW and SSW cubes is therefore complicated and discussed further in Section 7.12.5. Inside each cube, there are several 3D datasets, which are also mentioned in Section 7.8.5 and Section 7.8.6. These datasets are:

- **flux**: the mean of spectra within a grid square for the naive projected cubes or for the CP (convolution projected) cubes, the Gaussian weighted mean of all spectra falling within the width of the weight kernel.

- **error**: the standard error on the mean of spectra within a grid square for the naive cubes or the weighted error for the CP cubes

- **coverage**: the number of spectra (scans) within a grid square, or for the CP cubes, the sum of the weighted contributions

- **flag**: available for flagging, although not currently set by the pipeline

These products and the datasets they contain, are also described in Section 7.8.5 and Section 7.8.6. For cubes produced with HIPE version 12 or later, the frequency information is stored as part of the WCS third axis, which is defined by the keywords CRVAL3, CRPIX3 and CDELT3. For previous versions of HIPE, the frequency information was contained within the cubes as a table dataset called ImageIndex, which provided the mapping of the third dimension index to frequency, in GHz. The frequency of any FTS cube can be extracted using

```
frequency = cube.wave
```

From HIPE version 12, the frequency scale of FTS spectral cubes is also corrected to the Local Standard of Rest (LSR). See Section 7.11.1 for more details on this correction.

An important consideration for FTS spectral mapping is the type of source that was observed in the context of the calibration applied. FTS cubes contain extended-source calibrated data with flux in surface brightness units of W m\(^{-2}\) Hz\(^{-1}\) sr\(^{-1}\). Generally speaking, FTS spectral mapping is of extended regions and so the extended-source calibration is appropriate, but some observations may contain a point-source or a partially-extended region. In such cases, a conversion or correction of the calibration for the associated spectrum or spectra is required, as discussed in Section 7.12.3 and Section 7.6.

The following examples use the Level-2 naive-gridded data for a high resolution, fully sampled observation of the Orion Bar (observation ID 1342228734), obtained directly from the Herschel Science Archive. The SLW and SSW spectral cubes (cubeL and cubeS) can be loaded from the Observation Context (obs) as SpectralSimpleCubes using:

```
obsid = 1342228734
# Obtain the Observation Context from the HSA
obs = getObservation(obsid, useHsa=True)
# Extract the naive projected cubes for both detector arrays
cubeL = obs.level2.getProduct("HR_SLW_cube")
cubeS = obs.level2.getProduct("HR_SSW_cube")
```

To obtain the convolution projected (CP) cubes, append ".convol" to the product names, e.g. "HR_SLW_cube_convol". For apodized cubes append ".apod" to either type of cube name, e.g. "HR_SLW_cube_convol_apod". And for LR observations, change "HR" to "LR".
7.12.2. Extracting spectra from a spectral cube

A spectrum can be extracted from a cube spaxel using pixel coordinates. For example, to extract the spectrum with (column, row) pixel coordinates (5, 4), use:

```python
column = 5
row = 4
spec = cubeL.getSpectrum1d(row, column)
```

The output is in the form of a `Spectrum1d`. To extract the spectrum as a `SimpleSpectrum`, `spireProduct2SimpleSpectrum` can be used as:

```python
spec = spireProduct2SimpleSpectrum(cubeL, spaxelX=column, spaxelY=row)
```

`spireProduct2SimpleSpectrum` also accepts RA and Dec coordinates (see Section 7.12.4).

The output in all cases is a spectrum with flux units of $\text{W m}^{-2} \text{Hz}^{-1} \text{sr}^{-1}$, for which most of the `Spectrum Explorer Spectrum Toolbox` tools and the `Spectrum Fitter GUI` can be applied.

Unless the nearest neighbour projection algorithm was used to grid the cube (which is not generally recommended), each cube spaxel can contain spectra averaged from different spectrometer detectors (see Section 7.8). Each detectors has a unique flux calibration and fringe pattern, and therefore different systematic noise properties. Moreover, the outer rings of both detector arrays are partially vignetted by the instrument, with some suffering lower than average sensitivity. Therefore combining the spectra from different detectors may increase the noise per spaxel to greater than the expectation from HSpot. The spectrum in each detector has an associated uncertainty from averaging over a number of spectral scans (where two scans constitute one repetition). This is stored in the `error` dataset inside the cube, and is the standard error on the mean. For spaxels containing the average of two or more detectors, the combined systematic error of these detectors is folded into the statistical uncertainty.

7.12.3. Extracting a point source from a spectral cube

FTS spectral cubes are extended-source calibrated, with flux units of $\text{W m}^{-2} \text{Hz}^{-1} \text{sr}^{-1}$. If a spectral map contains a point source, the associated spectrum should be converted into point-source calibrated data with flux density units of Jy. This conversion should also be applied to the spectrum or spectra of a partially extended region, before using the Semi-Extended Correction Tool (SECT) to correct for source extent, as described in Section 7.6.

The HIPE `specPointSourceExtractor` task can extract a point-source or semi-extended source spectrum from an FTS spectral cube and apply the appropriate extended to point-source conversion. The task requires an input of the RA and Dec source coordinates (in decimal degrees), a SPIRE Calibration Context (or the beam parameter product) and the pre-processed cube (`preCube`) for one or both of the detector arrays. The `preCube` contains a `Spectrum2d` dataset, which the pipeline uses to collate all the extended-source calibrated spectra for a given array, before these are projected into the cube. `specPointSourceExtractor` assumes a point-like source is to be extracted from the `preCube(s)`, and uses a Gaussian assumption for the beam to correct the spectrum to the flux density expected for a source centred at the supplied coordinates.

`specPointSourceExtractor` can either be run from the GUI, which is listed in the HIPE Tasks window, or from the command line, e.g.:

```python
# Extract the preCubes
slwSpectrum2d = obs.level2.getProduct('HR_SLW_spectrum2d')
sswSpectrum2d = obs.level2.getProduct('HR_SSW_spectrum2d')
# Use specPointSourceExtractor to extract point-source calibrated
# spectra at RA/Dec [83.854, -5.411]
pointSpectrum = specPointSourceExtractor(slwSpectra=slwSpectrum2d,
                                         sswSpectra=sswSpectrum2d, coords=[83.854, -5.411],
                                         calibration=obs.calibration, maxDistArcsec=10.0,
                                         averageSpectra=False)
```
The output pointSpectrum, is a Spectrometer Point Source Spectrum product (SPSS), which is the form of the final products from the Spectrometer Single Pointing user pipeline. The centre detector(s) of pointSpectrum contain the point-source calibrated and position corrected data, extracted from the preCube(s). This pair of detectors are used for the output regardless of where the input coordinates fall on the spectrometer array footprint, which is an arbitrary choice for convenience, as due to gridding, the actual spectra can be a combination of the data from several detectors.

By default and for each input preCube, specPointSourceExtractor selects the spectrum closest to the supplied coordinates. For each array, picking a spectrum is constrained by maxDistArcsec, which sets a maximum selection distance. The task performs a weighted correction on the spectrum, based on the distance from the source location, assuming the source is Gaussian with a FWHM of the beam diameter. If the parameter averageSpectra is set to True, the task performs a weighted average of all the spectra within the maximum distance.

If the extracted point source is embedded in extended background (or foreground) emission, this should be removed prior to running specPointSourceExtractor. One possible way to remove a background is to choose a suitable background region in the same cube. To ensure compatible calibration, the background spectrum can also be extracted using specPointSourceExtractor. In order to avoid the subtraction increasing the noise in pointSpectrum, the background spectrum should first be smoothed or fitted, to obtain the wide-scale shape, which can be achieved with the Spectrum Toolbox smooth task, e.g.:

```plaintext
backgroundSpectrum = specPointSourceExtractor(
    slwSpectra=slwSpectrum2d,
    sswSpectra=sswSpectrum2d, coords=[83.849, -5.393],
    calibration=obs.calibration, maxDistArcsec=20.0, 
    averageSpectra=False)
backgroundSpectrum = smooth(ds=backgroundSpectrum, filter='Gaussian', 
    width=21, unit='GHz', edge='SHRINK_WIDTH', 
    variant='wave-flag-weight', overwrite=1)
```

The resulting backgroundSpectrum can be subtracted from pointSpectrum using the Spectrum Toolbox subtract task:

```plaintext
result = subtract(ds1=pointSpectrum, ds2=backgroundSpectrum, overwrite=True)
```

or more simply with:

```plaintext
result = pointSpectrum - backgroundSpectrum
```

For partially-extended sources, the output of specPointSourceExtractor can be used directly as input to SECT (see Section 7.6).

### 7.12.4. Matching spectra from SSW and SLW cubes

As discussed in Section 7.12.1, the final result of the standard pipeline for a SPIRE FTS mapping observations are two sets of spectral cubes. Each set contains separate cubes for SLW and SSW, as each cube has an array dependent WCS. So extracting spectra from a pair of array-cubes, which correspond to the same sky position is not generally achievable by simply specifying pixel coordinates. As long as the chosen position is within the region of both cubes, spireProduct2SimpleSpectrum can be used to extract the spectrum that is closest to the inputted RA and Dec sky coordinates, from each array:

```plaintext
# RA (decimal degrees) = 15 * (HH + MM/60 + SS/3600)
ra = 15.0*(5.0 + 35.0/60.0 + 25.2/3600.0)
# Dec (decimal degrees) = + or - (DD + MM/60 + SS/3600)
dec = -5.0 - 24.0/60.0 - 29.0/3600.0
#
slw = spireProduct2SimpleSpectrum(cubeL, ra=ra, dec=dec)
ssw = spireProduct2SimpleSpectrum(cubeS, ra=ra, dec=dec)
```

Using this method will extract slw from the spaxel located nearest to the input coordinates in the SLW cube, and likewise for the extraction of ssw from the SSW cube. However, the actual RA and Dec of the two selected spaxels are likely to be different, unless the two cubes have matched WCS.
It is much easier to compare data from an SLW and SSW cube, if the spatial grid is aligned between the two. In the standard pipeline, the centre of the grid for both cubes is located at the same RA and Dec, but the default grid spacings are different for each array. Therefore the two grids become more discrepant with distance from the centre of the map.

The standard grid spacing for a fully sampled map is 17.5″ for SLW and 9.5″ for SSW. In order to match the array grids, the SSW pixel size should be changed so there is an integer number of SSW pixels for each SLW pixel, i.e., one-to-one, or three-to-one, and so on. It might also be advantageous to convolve the cubes so that all frequencies have the same spatial resolution (see Section 7.12.5). Both of these methods involve regridding the cubes from the preCubes and even if the grid size is one-to-one, a separate WCS object must be used for each array, because the WCS also contains information on the frequency grid. Matching the array grids array can be achieved following:

```python
# If observation not already loaded:
#obsid = 1342228734
#obs = getObservation(obsid, useHsa=True)
###
# Set the resolution of the observation (HR or LR)
res = "HR"
# Obtain the preCubes from the Observation Context (obs)
preCubeL = obs.level2.getProduct('%s_SLW_spectrum2d'%(res))
preCubeS = obs.level2.getProduct('%s_SSW_spectrum2d'%(res))

# Create the WCS, specifying the same grid spacing for SLW and SSW.
# Note: this must be done separately for SLW and SSW because the WCS also
# contains information on the frequency axis.
wcsL = SpecWcsCreator.createWcs(preCubeL, 17.5/3600.0, 17.5/3600.0)
wcsS = SpecWcsCreator.createWcs(preCubeS, 17.5/3600.0, 17.5/3600.0)

# Ensure the WCS reference pixel is the same for both arrays
# (the reference RA and Dec are already the same and equal to the
# observation target coordinates)
wcsL.crpix1 = wcsS.crpix1
wcsL.crpix2 = wcsS.crpix2

# Ensure the size of the grid is the same for both arrays
wcsS.naxis1 = wcsL.naxis1
wcsS.naxis2 = wcsL.naxis2

# Recreate the spectral cubes with the naive mapper
cubeSlwMatch = spireProjection(spc=preCubeL, wcs=wcsL, projectionType="naive")
cubeSswMatch = spireProjection(spc=preCubeS, wcs=wcsS, projectionType="naive")

# To regrid the standard pipeline convolution projected cubes:
beamParam = obs.calibration.spec.beamParamList.getProduct(preCubeL.\n  meta["processResolution"].value, preCubeL.startDate)
beamDiamSlw = beamParam[0]["SLWC3"]["beamDiam"].data
beamDiamSsw = beamParam[0]["SSWD4"]["beamDiam"].data

cubeSlwMatchConvol = spireProjection(spc=preCubeL, wcs=wcsL, \n  projectionType="convolution", beamWidthArray=beamDiamSlw)
cubeSswMatchConvol = spireProjection(spc=preCubeS, wcs=wcsS, \n  projectionType="convolution", beamWidthArray=beamDiamSsw)
```

The spaxel numbers between each pair of grid-matched cubes should exactly correspond, making it significantly more accurate to extract spectra using the same sky coordinates.

The SLW and SSW spectra of the matched cubes could be combined into single products per pair, using spireProduct2SimpleSpectrum and following the instructions in Section 7.11.2. However, combining data from the two arrays must be carried out with caution, because each spaxel is an average of spectra coming from different detectors and the actual separation of SLW and SSW detectors on sky is different.

### 7.12.5. Convolving a cube to a different beam size

The FTS beam size and shape are frequency dependent, with the SSW beam FWHM roughly a factor of two smaller than for SLW, as shown by Figure 7.48. The SSW profile is well approximated by a
Gaussian over the full frequency band, whereas the SLW beam is approximately Gaussian below ~700 GHz, but above that the profile has a more complicated shape, as discussed in Makiwa et al. (2013). The beam diameters and profiles can be found in the Calibration Context products SCalSpecBeamParam and SCalSpecBeamProf, which are detailed in Section 7.8.1. These differences between the two array beams must be considered when comparing SLW and SSW cubes, but also the frequency dependent size should be accounted for before layers within a single cube can be compared.

The convolution projection algorithm can be used to obtain a cube convolved to a constant or different beam size. The convolution gridding approach is useful because it uses a frequency dependent kernel, although one that is limited to a Gaussian. There is a brief description of the convolution projection algorithm in Section 7.8.5 and also in the SPIRE Pipeline Specification Manual. As the SSW array beam is approximately Gaussian, so the quadratic difference of Gaussian widths can be used to calculate an appropriate kernel.

The Orion Bar, which is used as an example source in part of this section, has extended and uniform emission on the scale of the FTS beam, i.e. it is classed as a fully extended source. For such a source, the join between SSW and SLW is usually good both before and after convolution, with the convolution purely adjusting the absolute level of the spectrum. L1521F is an example of a source with more complicated structure within the FTS beam, so this source is used in the following example, which convolves the SSW cube to a beam size of 80''. Note that if running the following example code, the code in the previous section (Section 7.12.4), must be re-run using the observation ID 1342240020, setting res to "LR" and using the same WCS as below, which is created using a reference beam of 80''.

```python
# Load the observation of L1521F:
obsid = 1342240020
obs = getObservation(obsid, useHsa=True)
#
# Set the resolution of the observation (LR)
res = "LR"
# Get the beam sizes from the SPIRE Calibration Context,
# which is inside the Observation Context (obs)
cal = obs.calibration
beamFwhmSsw = cal.spec.beamParamList.getProduct(res, obs.startDate)[0]['SSWD4']['beamDiam'].data
# Set a large reference beam size
refB = 80
# Get the SSW preCube and set up the WCS using the reference beam
preCube = obs.level2.getProduct('%s_SSW_spectrum2d'%(res))
wcs = SpecWcsCreator.createWcs(preCube, refB/2./3600., refB/2./3600.)

# Calculate the quadratic difference between true beam size and reference beam
diffBeamSsw = SQRT(refB**2 - beamFwhmSsw**2)

# Create the cube by convolving every slice to the reference beam, assuming a
# Gaussian beam and using the difference between the Gaussian and reference beams.
# Make sure not to restrict the maximum radius of the kernel (which is small)
cubeSswConv = spireProjection(spc=preCube, wcs=wcs, 
    projectionType="convolution", 
    beamWidthArray=diffBeamSsw, 
    kernelMaxRadius=Double.NaN)
```

Convolution can also be used to match the SLW and SSW cubes, although care must be taken when interpreting the results. The SLW beam has its largest deviation from Gaussianity at the high frequency end of the band, in the overlap region with SSW. This means that when there is structure in the SSW map on a scale that is smaller than the SLW beam, the convolution mapper may not produce spectra that are well matched in the SLW-SSW overlap region. It is difficult to calculate a more accurate kernel because only the core of the SLW beam has been characterised and the beam profile calibration product only contains an 128'' by 128'' map of the beam. For this reason, the non-Gaussian spectral region may need to be omitted when SED fitting to the continuum shape.

The following is an example of using a large reference beam size when convolving both the SLW and SSW cubes:
Set the resolution of the observation (HR or LR)
res = "LR"

Get the beam sizes from the SPIRE Calibration Context,
which is inside the Observation Context (obs)
cal = obs.calibration
beamFwhmSlw = cal.spec.beamParamList.getProduct(res, obs.startDate)[0]["SLWC3"]
beamFwhmSsw = cal.spec.beamParamList.getProduct(res, obs.startDate)[0]["SSWD4"]

Set a large reference beam size
refB = 80

Obtain the SLW and SSW preCubes
preCubeL = obs.level2.getProduct('%s_SLW_spectrum2d'%(res))
preCubeS = obs.level2.getProduct('%s_SSW_spectrum2d'%(res))

Create the WCS, specifying the same grid spacing for SLW and SSW.
This must be done separately for SLW and SSW because the WCS also
contains information on the frequency axis.
wcsL = SpecWcsCreator.createWcs(preCubeL, refB/2./3600.0, refB/2./3600.0)
wcsS = SpecWcsCreator.createWcs(preCubeS, refB/2./3600.0, refB/2./3600.0)

Ensure the WCS reference pixel is the same for both arrays
(the reference RA and Dec are already the same and equal to the
observation target coordinates)
wcsS.crpix1 = wcsL.crpix1
wcsS.crpix2 = wcsL.crpix2

Ensure the size of the grid is the same for both arrays
wcsS.naxis1 = wcsL.naxis1
wcsS.naxis2 = wcsL.naxis2

Calculate the quadratic difference between true beam size and reference beam
diffBeamSlw = SQRT(refB**2 - beamFwhmSlw**2)
diffBeamSsw = SQRT(refB**2 - beamFwhmSsw**2)

Create the cubes by convolving every slice to the reference beam
assuming a Gaussian beam and using the difference between the
Gaussian and reference beams. For SLW, make sure not to restrict
the maximum radius of the kernel (which is small)
*The convolution mapper is quicker if the reference beam is set to a larger value*
cubeSlwConv = spireProjection(spc=preCubeL, wcs=wcsL, 
projectionType="convolution", 
beamWidthArray=diffBeamSlw, 
kernelMaxRadius=Double.NaN)
cubeSswConv = spireProjection(spc=preCubeS, wcs=wcsS, 
projectionType="convolution", 
beamWidthArray=diffBeamSsw)

Opening one of the resulting cubes in the Spectrum Explorer and then dragging the second to the plot
in the top panel, allows an easier comparison between the two. Following the "Cube comparison" label
in Figure 7.79, click on "NONE" and select "WORLD" from the pulldown menu opened. Make sure
this is selected for both cubes and that "Show Comparison Preview" is ticked. Now matching spectra
from the cubes will be shown in the plot and fixed in the plot when a spaxel is selected. More details
on comparing cubes with the Spectrum Explorer can be found in the Herschel Data Analysis Guide.
Alternatively, the cubes could be compared by in HIPE using the following script:

set up a multi-panel plot, with a panel for each spaxel
# in the convolved matched cubes
gridPlot = PlotXY()
gridPlot.gridLayout.setGap(0, 0)
gridPlot.plotSize=(0.7,0.7)
firstRow = 0
lastRow  = 6
firstCol = 0
lastCol  = 6

Loop over the spaxels and plot a comparison between
the matched Naive cubes and the matched convolved cubes

Opening one of the resulting cubes in the Spectrum Explorer and then dragging the second to the plot
for i in range(firstCol, lastCol-firstCol+1):
    for j in range(firstRow, lastRow-firstRow+1):
        thisPlot = SubPlot(SubPlotGridConstraints(i,firstRow-j,1,1))
        # Extract the spectra from the matched Naive cubes and plot in green
        spectrumSsw = cubeSswMatch.getSpectrum1d(j, i)
        spectrumSlw = cubeSlwMatch.getSpectrum1d(j, i)
        thisPlot.addLayer(LayerXY(spectrumSsw.wave, spectrumSsw.flux,\
            color=java.awt.Color.GREEN, stroke=1.7))
        thisPlot.addLayer(LayerXY(spectrumSlw.wave, spectrumSlw.flux,\
            color=java.awt.Color.GREEN, stroke=1.7))
        # Extract the spectra from the matched and convolved cubes
        # and plot SLW in red and SSW in blue
        spectrumSlwConv = cubeSlwConv.getSpectrum1d(j, i)
        spectrumSswConv = cubeSswConv.getSpectrum1d(j, i)
        thisPlot.addLayer(LayerXY(spectrumSlwConv.wave, spectrumSlwConv.flux,\
            color=java.awt.Color.RED, stroke=1.5))
        thisPlot.addLayer(LayerXY(spectrumSswConv.wave, spectrumSswConv.flux,\
            color=java.awt.Color.BLUE, stroke=1.5))
        # Add the spaxel number as an annotation
        thisPlot.layers[0].setAnnotation(1, Annotation(600, 1.1e-18,\
            "(%i, %i)"%(i,j), fontSize=10))
        # Adjust the ticks and make tick labels invisible
        thisPlot.layers[0].xaxis.tick.height=0.05
        thisPlot.layers[0].xaxis.tick.minorHeight=0.01
        thisPlot.layers[0].yaxis.tick.height=0.05
        thisPlot.layers[0].yaxis.tick.minorHeight=0.01
        thisPlot.layers[0].xaxis.title.visible=0
        thisPlot.layers[0].xaxis.tick.labelVisible=0
        thisPlot.layers[0].yaxis.title.visible=0
        thisPlot.layers[0].yaxis.tick.labelVisible=0
        # The yrange may need adjusting,
        # depending on the brightness of the source
        thisPlot.layers[0].yrange = [0, 2.2e-18]
        gridPlot.addSubPlot(thisPlot)

Figure 7.80 compares the results of running the above examples on the LR fully sampled mapping observation of L1521F (ID 1342240020) with a SPIRE photometer map of the same source (ID 1342202254). The WCS used to grid the cubes is overplotted on the photometer map in pink. The same WCS should be used to create both the matched cubes and the matched and convolved cubes (in this case, using a reference beam of 80''). It is clear that in the upper right part of the cubes, where structure is evident in the photometer map, the join in the overlap between the SSW and SLW spectra is not good, even after convolution. It is likely that the non-Gaussian beam at the high frequency end of SLW is causing the problem in these spaxels.
Figure 7.80. A SPIRE photometer observation of L1521F (left) with the matched WCS grid from the spectral cubes marked in pink. The right hand plot shows the spectra from the SLW and SSW cubes after matching the WCS (green) and after matching the WCS and convolving to an 80″ beam (SLW in red, SSW in blue).

Code to recreate the SPIRE photometer map plot in Figure 7.80:

```python
# Download the photometer observation from the archive
obsP = getObservation(1342202254, useHsa=True, instrument="SPIRE")
photMap = obsP.level2.getProduct("extdPSW")
# Take the source RA and Dec coordinates from the FTS metadata
raNominal = cubeSswMatch.wcs.crval1
decNominal = cubeSswMatch.wcs.crval2
#
# Convert source RA and Dec to pixel coordinates in the map.
xNominal = photMap.wcs.getPixelCoordinates(raNominal, decNominal)[1]
yNominal = photMap.wcs.getPixelCoordinates(raNominal, decNominal)[0]
#
# Set up the plot and add the photometer image
pl = PlotXY()
pl.plotSize = (4.0, 4.0)
pl.addLayer(LayerImage(photMap.image))
#
# Set the origin and the scale of the axes so that they coincide with
# the WCS of the photometer image, amending the centre with the new
# reference coordinates.
# (convert CDELT1 as the x-axis will be displayed in RA, hh:mm:ss)
pl[0].xcdelt = photMap.wcs.cde1t/COS(decNominal*Math.PI/180.)
pl[0].ycdelt = photMap.wcs.cde2t
pl[0].xcrpix = xNominal+1
pl[0].ycrpix = yNominal+1
pl[0].xcrval = raNominal
pl[0].ycrval = decNominal
#
# Change the axis type to have ticks in degrees/hours, min, sec
# and the RA increasing towards the left.
#
from herschel.ia.gui.plot.renderer.axtype import AxisType
pl.xaxis.setAxisType(AxisType.RIGHT_ASCENSION)
pl.yaxis.setAxisType(AxisType.DECLINATION)
pl.xaxis.titleText = 'Right Ascension (J2000)'
pl.yaxis.titleText = 'Declination (J2000)'
#
# Adjust ticks for visual appearance
pl.xaxis.tick.autoAdjustNumber = 0
pl.xaxis.tick.number = 5
pl.xaxis.tick.minorNumber = 3
pl.xaxis.getAuxAxis(0).tick.autoAdjustNumber = 0
```
7.12.6. Cube fitting with the Spectrum Fitter GUI

Spectral cubes can be fitted in HIPE using the Spectrum Fitter GUI (SFG), by fitting to the spectrum from a single spaxel and then fitting to all spaxels in the cube using the MultiFit tab. Firstly, open the cube for fitting in the Spectrum Explorer and select a single spaxel to display the corresponding spectrum. Then initiate the SFG using the icon of a hammer and spanner overlaid with a Gaussian, which can be found on the top bar of the Spectrum Explorer window (see Figure 7.79 and the Herschel Data Analysis Guide). The combined continuum and line model for this reference spaxel should be set up in the same way as for any single spectrum (Section 7.11.4). Once a satisfactory fit has been obtained, switch to the MultiFit tab, and press Accept. The fitter will then loop over all spaxels in the cube and perform a fit to each one, using the best fit parameters from the initially selected spectrum as the first guess for each combined model. A text file of the results, or a script to recreate the fit, can be written to disk by ticking the options inside the tab. The full MultiFit functionality of the SFG is described in Section 7.19 of the Herschel Data Analysis Guide. The next section (Section 7.12.7) describes a script to fit SPIRE cubes, which follows the same basic method used by the SFG, but has been focused on multi-fitting to and extracting results relevant to spectral mapping.

7.12.7. Cube fitting and line intensity/velocity maps

The "Spectrometer Cube Fitting" script is available in HIPE for fitting FTS cubes with the SpectrumFitter, and can be found in the Scripts > SPIRE Useful scripts menu on the toolbar. The SpectrumFitter Line Fitting script (which is described in Section 7.11.9) is used as a base, but has been modified to fit all spaxels in the cube and extended to plot the integrated intensity map for each line fitted. In order to obtain initial guess parameters to be applied during the fitting of all other spaxels, the script initially fits the spectrum from a reference spaxel. The reference spaxel can be changed from
the default of the map centre, in the User Selectable Options section at the beginning of the script. The script can also be changed to fit to a convolution projected cube, rather than the naive projected cube, by appending "_convol" to the product name. The script generates a plot of the reference spaxel spectrum, the total model fitted and the residual, while the best-fit parameters are written to an xml file, to be used when setting up the fit to the rest of the spaxels. The MultiFit function of the SpectrumFitter is then used to perform a fit to the rest of the cube.

The global fitting results for the cube are sorted into four Double3d arrays, one for integrated line flux, one for velocity, and two for their associated errors. Other arrays could be added by modifying the script in the loop starting on line 362. The script creates a list of SimpleImages for integrated line flux and velocity (line 375). There is one SimpleImage created for each line fitted, which are the $^{12}$CO lines by default. As an example of visually comparing the results, the integrated intensity maps are plotted for each line along with the integrated map from the entire cube (see Figure 7.81). The velocity images can also be visualised, either by double clicking on velocityImages in the Variables tab, and then double clicking on the relevant spectral line, or by editing the script to plot velocityImages instead of lineFluxImages.

The residuals from every spaxel are also plotted (see Figure 7.82). In the case of the Orion Bar observation used for this example, the $^{12}$CO lines are well fitted. The residual plot may reveal that for other observations, the fit is poor for one or more lines, for some spaxels, and the associated line intensity map looks may look odd compared to the respective plots of other lines. In the case of poor fitting results, the script can be adjusted to try and improve the results. Optimisation of the script may be possible by: changing the reference spaxel on lines 187 and 188, to give a different initial guess for the MultiFit; or changing the fixed width of each line (line 300); or fixing the line centre frequencies. See Section 7.11.9 for other suggestions on optimising the fitting carried out by the script.

The fit parameters from the script can be visualised by extracting a parameter cube, which will contain one layer for each fit parameter. It can be used to show a map of each parameter and can be obtained from the multi-fit object (mf) using:

```python
parCube = mf.parameterCube
```

The parameter cube is described in more detail in the Herschel Data Analysis Guide.

In order to compare the cube residual and fitted model with the input cube, residual and totalModel cubes are created by the script:

```python
residual = mf.getResidual()
totalModel = mf.getTotalModel()
```

The model can be overplotted on the data by opening the input data cube in the Spectrum Explorer and dragging and dropping the totalModel variable onto the plot in the top panel of the Spectrum Explorer window, which will add totalModel as a second tab in the bottom left panel. In order to overplot pixel-by-pixel, change the drop down menu at the bottom of both tabs from NONE to PIXEL. When a spaxel is selected, a spectrum from both the data and the model cube should appear in the plot. See Herschel Data Analysis Guide for more details of this feature. Individual model cubes can also be obtained, e.g., the polynomial baseline cube can be extracted from mf using:

```python
polyBaseline = mf.getModel(0)
```

This would then allow the baseline to be subtracted from the data cube using the Spectrum Toolbox task, subtract:

```python
subtractedCube = subtract(ds1=cube, ds2=polyBaseline)
```
**Figure 7.81.** The integrated intensity plot produced by the Spectrometer Cube Fitting script after running it on the SSW CP cube of an observation of the Orion Bar, 1342228734.

**Figure 7.82.** Plots produced by the Spectrometer Cube Fitting script after being run on the SSW CP cube of the Orion Bar observation 1342228734. The initial fit to the central SSW spaxel is show to the left, along with the model fitted and the residual. To the right is a plot of the residuals from all spaxels in the cube. The $^{12}$CO lines are well fitted for the central spaxel, whereas some spaxels show more significant residual.

### 7.12.8. Publication quality plots for mapping data

The "Spectrometer Cube Fitting" script produces a plot of fitted flux, as shown in Figure 7.81, with the associated commands found on lines 417-467 in the script. This plot can easily be customised to produce a figure suitable for a paper. The following example shows how to adjust the spacing between subplots and add contours.

In order to remove gaps between the sub-plots, the grid spacing needs to be set to zero. The axis labels and titles at the joins also need to be removed, and can be suppressed inside the loop over sub-plots before adding them to the main plot using:
subP.baseLayer.xaxis.tick.labelVisible = 0
subP.baseLayer.xaxis.title.visible = 0
subP.baseLayer.yaxis.tick.labelVisible = 0
subP.baseLayer.yaxis.title.visible = 0

These commands remove all axis labels, but the loop could be adjusted so that, for instance, all y-axis to the left edge and x-axis to the bottom edge of the subpanels are visible. See also the example in Section 3.32 of the Herschel Data Analysis Guide.

In order to add contours to the plot, the following changes to the loop over sub-plots can be made:

- Change the colour table to be in black and white so that contours stand out:

  ```python
  ll.colorTable = 'Ramp'
  ```

- Add contours as a new layer in each sub-plot at the end of the loop after adding the sub-plot to the main plot (p.addSubPlot(subP)):

  ```python
  contours = automaticContour(image=map, levels=10, distribution=1,
                               min=levs[0], max=levs[1])
  keys=contours.keySet()
  for key in keys:
      if key.startswith('Contour'):
          cont=contours[key]
          keys=cont.keySet()
          for keyc in keys:
              x=(cont[keyc].data[:,1]-map.wcs.crpix1+1.0)*3600.*
                 map.wcs.cdelt1/COS(map.wcs.crval2*pi/180.)
              y=(cont[keyc].data[:,0]-map.wcs.crpix2+1.0)*3600.*
                 map.wcs.cdelt2
              subP.addLayer(LayerXY(x,y,color=java.awt.Color.RED,stroke=0.8))
  ```

**Figure 7.83** shows the results of applying the above code changes to the cube line fitting script plot. Further refinements can be made by adjusting the pixel scaling of each image or adjusting the x and y ranges.

**Figure 7.83.** The results of overplotting contours on the output of the cube line fitting script.

For a quick look at contours, it is also possible to use the `automaticContour` task as a GUI, which can be accessed from the `Tasks` tab in HIPE. The resulting contour object can be directly dragged into an image displayed with the `Image Viewer`. In order to quickly extract an image from a cube, right
click on the cube inside the Spectrum Explorer and select Extract current layer. The extracted layer can then be opened with the Image Viewer.

For more information about automatic Contour and plotting contours in general, see the Herschel Data Analysis Guide Section 4.17 and Section 3.30.

7.12.9. Comparison with photometer maps

The comparison of a spectral cube with a SPIRE photometer map must take into account the filter transmission profiles of the Photometer and also the difference in beam size between the spectrometer and photometer. Further details and an example comparison are given in Section 7.10.
Chapter 8. SPIRE Visualisation Tools

8.1. SDI/SDS Explorer

The SDI/SDS Explorer is a GUI-based visualization tool that allows the contents of SPIRE interferogram and spectral products to be visualised. The SDI Explorer works with Spectrometer Detector Interferogram (SDI) products, and the SDS Explorer works with Spectrometer Detector Spectrum (SDS) and Spectrometer Point Source Spectrum (SPSS) products. The SDS Explorer will also work with several of the calibration products.

8.1.1. Starting SDI/SDS Explorer

The SDI/SDS Explorer can be used to view interferogram and spectral products. Right-click on the product in the Variables tab, and “Open with” SDI or SDS Explorer. The SDI/SDS Explorer is the default viewer for several SPIRE products, so in many cases a simple double-click on the spectrum variable will be sufficient to launch it.

![Figure 8.1. Starting the SDI/SDS Explorer via HIPE.](image)

8.1.2. SDI/SDS Explorer Layout

The Graphical User Interface of the SDI/SDS Explorer is divided into four sections: the SLW array footprint on the top to the left; the SSW array footprint on the top to the right; the Control Panel on the bottom left; and the Preferences Panel on the bottom right (see the Figure 8.2).
8.1.2.1. Detector Array Display

The detector array panel at the top of the GUI allows the selection of any of the SPIRE spectrometer detectors. The layout reflects the arrangement of the detectors in spacecraft coordinates, and the N-E arrows on the lower right hand side of each array show how this maps onto sky coordinates. The detectors that are coaligned between the two arrays are highlighted with a red border. Detectors that contain data (and are therefore "active" in the display) are coloured according to the strength of the signal (see later for a description of the colourscheme). Detectors without any data are coloured red (e.g. the two dead detectors in SSW in Figure 8.2).

The Detector Array Panel is "active" for detectors that contain valid data. A single click with the left mouse button on any detector will launch a plot window showing the spectrum from that detector. A single click on a different detector will overplot its spectrum in the existing plot window. Only one plot window is present at a time, so clicking on further detectors adds them to the same plot. A second mouse click on a detector which has already been plotted will remove its spectrum from the plot. In order to start a fresh plot window, simply close the current plot window - this will mean the next mouse click on a detector will launch a new plot. The names of the detectors that are displayed in the plot are shown in the legend.

Right clicking with the mouse on a detector in the Detector Array Panel will bring up a menu, giving options to open the detector data in another viewer ("Open With") or to "Send To" another application. There is also an option to "Create Variable" which will create a variable of class `SpireSpectrum1d` in the session. In addition, a detector can be dragged with the mouse and dropped into the HIPE Variable Tab to create a variable, or dragged into the Console, or into a script to paste the full reference to the detector dataset.

8.1.2.2. Control Panel

The Control Panel (see bottom left of Figure 8.2) gives control of the display:

- Select a subset of the scans within the data product (Scan Selection).
- Create plots showing spectra from all detectors in a mosaic plot ( Thumbnails).
- Define the fill colours for the detectors (Colour Scheme Range).
This **Scan Selection** section of the Control Panel gives control over which scans are plotted when clicking on a detector in the Detector Array Panel. By default, only the first scan is plotted.

- The **Forward** and **Reverse** buttons allow the plotting of scans based on the direction of those scans. Since all scans should be either forward or reverse, checking both options will plot all the scans in the product.

- The **Single** option allows one scan at a time to be plotted (selecting which scan using the scan number box, or the slider).

- The **All Scans** option plots all the scans of a given detector.

- The **free text** box can be used to specify a range of scans to be plotted. For example, to plot scans 1 through 4 and scans 7 and 8, type: 1–4 7,8. The wildcard * will select all scans.

The **Thumbnails** section of the Control Panel allows a mosaic plot of the spectra from all detectors in the array to be made, with each "thumbnail" spectrum placed at the relevant position in the array. The scaling option chosen in the Preferences panel (**Section 8.1.2.3**) is applied to each thumbnail image - either individually for each detector, or as a whole across the array depending on whether the "Thumbnail plots scale matched" box is ticked. If this box is ticked, then the x and y scale for all detectors is set to the same value. If not, and the "Fit" or "Passband/Fit" option is chosen, all detector plots will be optimised for their own data only. If the "User (fixed scale)" option is chosen, it is best to set up the plot scaling for a single detector first, and then use the thumbnail option to create similar plots for the other detectors. Depending on the selection in the Preferences Panel, data are shown for all, or only the unvignetted detectors. Three options are available under the Thumbnail drop-down menu (see **Figure 8.3**).

1. **SLW** to plot data which were recorded by the detectors in the long wavelength detector array.
2. **SSW** to plot data which were recorded by the detectors in the short wavelength detector array.
3. **Co-Aligned** to plot data which were recorded by the co-aligned detectors in SLW and SSW.

![Figure 8.3. Control Panel Thumbnails selection menu.](image)

The **Colour Scheme Range** section controls the colour scheme used for detectors in Detector Array Panel. Two colour schemes are available which both go from white (high values) to black (low values): Grey Scale and Heat. The values used to determine the colour are the average of the detector data within a user-specified range. The range slider, and the indices and values displayed next to it, specify the abscissa range in the interferograms or spectra used to compute the average signal value and subsequently set the colours. Note that the slider and text boxes work only for the abscissa indices, not the values (which are shown below the slider).

### 8.1.2.3. Preferences Panel

The Preferences Panel (see **Figure 8.4**) gives control of:

- Which detectors are to be displayed in the Detector Array Display and the Thumbnail plots. By default, all of the nominal detectors are shown. This can be restricted so that only unvignetted detectors are shown.

- The initial scale of the plots.

- The title, subtitle, and legend entries for the main plot area.

- Whether the plot legend is shown (default) or hidden.

- Whether error values are displayed.
• The units of the spectral phase (if Phase is selected as the plot type)

• The quantity plotted when complex data are loaded - either the Real or Imaginary part, the Phase, or Absolute. The Plot Type selection is only available if the product contains complex data with real and imaginary components.

Figure 8.4. SDI/SDS Explorer - Preferences Panel.

The first two check boxes allow selection of the displayed detectors. By default, the "Nominal detectors only" button is ticked - this means that only detectors that actually see the sky are displayed. If no boxes are ticked, all detectors including the resistors and thermistors are displayed. The outer ring of detectors in each array are partially vignetted inside the instrument and when the "Unvignetted only" button is ticked, these vignetted detectors are not displayed.

The three selection check boxes for the Initial scale are used to select whether the initial scale of a plot is fixed to the last user choice ("User (fixed scale)"), whether the scale of the plot adjusts to fit to the current data ("Fit"), or whether the plot presents the full frequency range defined by the instrument in the x-axis but fits the y-axis to the data ("Passband/Fit").

The Edit buttons allow the user to customize the title, subtitle, and legend of the main plot area. These can be constructed using metadata items from the product, which are selectable from a scrollable list (see Figure 8.5). By default, the title is set to the OBSID - BBID - STARTDATE.

Figure 8.5. Preference panel to edit plot titles.
The **Legend** check box shows or hides the legend (by default this box is ticked so that the legend is shown).

The **Errors** check box specifies whether to display the error column of the data in the plot (see Figure 8.6). If a dataset contains an “error” column, ticking the check box will cause the errors, multiplied by the sigma factor, to be rendered as a transparent envelope which surrounds the data. The text field beside the error check box is used to set the sigma factor for the displayed error envelope.

The SDS Explorer can cope with spectra that contain **complex numbers**. The lower two options for Phase and Plot Type become active if the spectrum contains complex values. In this case the Plot Type option specifies whether to display the real part, imaginary part, phase or the absolute value.

The absolute value of a complex number is given by $\sqrt{\text{imaginary}(s)^2 + \text{real}(s)^2}$. This is useful if the Fourier transform has been applied to an interferogram that is not symmetrical about zero path difference, for example to check the residual phase. However, in the standard pipeline, the phase correction task works very well to correct the phase of the interferogram, and a single sided Fourier transform is carried out (i.e. the pipeline spectra do not contain complex values). If “Phase” is selected as the plot type, the displayed units can be set to radians (from $-\pi$ to $+\pi$) or degrees (from $-180^\circ$ to $180^\circ$). This selection only applies to the first time when phase data are plotted. Changing this selection subsequently does not have any effect on the plot until a new plot is created. Note that the spectral phase $\phi(S)$ is defined as $\tan(\phi(S)) = \frac{\text{Imaginary}(S)}{\text{Real}(S)}$.

![Figure 8.6](image.png)

**Figure 8.6.** If a dataset has non-zero error values, both data and error are plotted when the error checkbox is ticked.

### 8.1.3. Example 1: Plotting and Overplotting

In order to inspect data from a specific scan and detector from a specific product, perform the following steps:

1. Start the SDI/SDS Explorer for the product in question.
2. In the Scan Selection section of the Control Panel, specify which scan(s) should be plotted, e.g. all reverse scans.
3. In the Preferences Panel, edit the plot titles, subtitles, and legends if desired.
4. In the Preferences Panel, select the Initial Scaling needed, e.g. Passband.
5. In the Detector Array Display shown in Figure 8.2, single left mouse click the detector to plot its data, e.g. SLWD3. A new PlotXY window will open with the SLWD3 detector plotted as shown in Figure 8.7.

6. For an overplot, click the left mouse button on an additional detector, e.g. SSWC2 (see Figure 8.8). An additional single-click would remove the spectrum from this detector again.

![Data taken on 2010/Jun/01 16:58:33 UTC](image)

Figure 8.7. Single plot of the reverse scans 1 and 3.
8.1.4. Example 2: Making a Thumbnail Image

In order to compare data from different detectors on the same page perform the following steps:

1. Start the SDI/SDS Explorer for the product in question.

2. In the Scan Selection section of the Control Panel specify which scan(s) should be plotted, e.g. scan number 1.

3. Open the main plot for a single detector by performing the steps in Section 8.1.3, and highlight the desired range with the mouse. This is the range that will be plotted for all detectors in the thumbnail images. For example, the overlap region between the two bands between 950 and 1000 GHz.

4. In the Preferences Panel, select whether to plot thumbnail images from all, only the nominal, or only the unvignetted detectors, e.g. "Unvignetted only".

5. From the Thumbnails drop-down menu on the Control Panel, select SLW, SSW, or Co-Aligned (see Figure 8.9 for the result when selecting SSW).

Note that in order customize the thumbnail plot further, an example script is provided in HIPE and described in Section 7.2.3.2.
8.1.5. Example 3: Plotting complex data: real, imaginary, absolute, and phase

Applying the Fourier transform to a double-sided interferogram results in a complex spectrum. The SDS Explorer gives specific consideration to visualize complex spectra. Perform the steps below on a complex spectrum to visualize the data in different ways:

1. Consult the SPIRE Pipeline Specification Manual section on the Fourier transform task in SPIRE Pipeline Specification Manual for instructions to create an SDS using the double sided Fourier transform (with ftType "prePhaseCorr"). The resulting SDS product should contain complex numbers in the flux column.

2. Start the SDS Explorer for the product in question.

3. In order to see the real and imaginary values overplotted, go to the Preferences Panel, select the PlotType to be "Real" and "Imaginary", and then click on the relevant detector to produce the equivalent of Figure 8.11.

4. In order to see the phase, go to the Preferences Panel, select the PlotType to be "Phase". Also, select in the Phase section whether data should be present in units of "Radians" or "Degrees". Then click on the relevant detector to produce the equivalent of Figure 8.10.
Figure 8.10. Viewing the phase of a complex spectrum.

Figure 8.11. Overplotting the real and the imaginary part of a complex spectrum.
8.2. Detector Timeline Explorer (DTE)

The Detector Timeline Explorer (DTE) is a Graphical User Interface allowing a user-friendly view of SPIRE (both photometer and spectrometer) Timeline-type Products.

8.2.1. Starting DTE

The DTE can be started from the Herschel Interactive Processing Environment (HIPE). At least one instance of a Timeline-type product must already be available in memory to launch the DTE (e.g. an output from a pipeline script or product loaded from Local Store).

In HIPE: identify the timeline-type product for visualization from the Variables list window and right-click on the product. From the Open With menu entry, select Detector Timeline Viewer from the drop-down menu (Figure 8.12).

The DTE can also be called from the command line as follows:

```python
from herschel.spire.ia.gui import DetectorTimelineExplorer
DetectorTimelineExplorer.show(pdt)
```

![Figure 8.12. Starting the DTE via HIPE.](image)

8.2.2. DTE Layout

The Display Panel is divided in three sections (see Figure 8.13):

- **LEFT - Array Display**: In this area we have a region to display the bolometer arrays (three arrays for the photometer, two for the spectrometer).
- **RIGHT - Quick View**: This area is used to display the plots.
- **BOTTOM - Control Panel**: This area contains menus to select the array and the type of sub-products that you want to view. In this area there is also the slider and play-stop button for browsing through time across the timeline with a selectable color scheme, time scale, etc.
8.2.2.1. Array Display

The Array Display area shows the bolometer arrays (Figure 8.14). A Control Panel contains drop-down menus to select whether to visualize all arrays or only one (Figure 8.17).

There are two color schemes for the bolometer arrays (heat and grey) and the preferred one can be selected from the Colour&Mask Preferences Panel (Section 8.2.2.6). The values for both color schemes are based upon the intensity at the current index of the channel.

Coaligned detectors have a red border.

We can visualize the nominal bolometers only by selecting the Nominal checkbox in the Control Panel.

A timeline can be plotted with a single click on the desired detector and when the mouse is over a bolometer, the instantaneous value can be seen within the tooltip.

A mosaic plot can be created with a right-click on the desired array and selecting Create Mosaic".
8.2.2.2. Quick View Area

The "Quick View" area is used to visualize the plot (Figure 8.15).

When the DTE is started, a plot is created within the "Quick View" area. An array's central detector is plotted. In the case where any central detector is not good, the first valid detector is plotted.

The plot can be visualized in the Quick View area or in a new floating window. When floating windows are selected to display the plot, each plot creates a new window (Figure 8.15) and erases the "Quick View" area.

An over-plot can also be created (see Section 8.2.2.4).

The data can be visualized with the Table Plotter (see Section 8.2.2.5).

Access to all plot properties (axis scale, title, label, etc.) is obtained with a simple right-click over the plot.

8.2.2.3. Control Panel

The Control Panel contains (Figure 8.16):
Figure 8.16. Control Panel.

- Data of the observation;
- "Current Index" label that shows row/readout #;
- "Select Index" writable field allowing selection of row/readout #. A warning message appears if the selected index is out of range;
- Drop-down menu to select the array and the dataset to view (Figure 8.17);

Figure 8.17. Drop-down menu to select the array (on the left) and the dataset (on the right).

- "Nominal" checkbox: to visualize only the nominal bolometers;
- "Colour&Mask preferences” button to open the preferences panel for colours and masks (Section 8.2.2.6);
- "DTE Help" button to open this section of SDRG;
- "Select Plot Window" radio buttons to select where to create a plot (in the Quick View area or in a new floating window). When the Over Plot checkbox is selected, the plot is always created in the QuickView area and the "Floating" button is disabled;
- A slider and play/stop button for browsing through time across the timeline (Figure 8.18). The frame per second (fps) can also be selected;

Figure 8.18. Slider for browsing through time across the timeline.

- Three radiobuttons ("Plot", "Over Plot" and "Table Plotter") that allow creation of plots, overplots (see Figure 8.19 and Section 8.2.2.4) or data visualization with Table Plotter (see Figure 8.19 and Section 8.2.2.5);

Figure 8.19. Plot Visualization: Overplot (Left) and Table Plotter (Right).
8.2.2.4. Overplot

To enable overplotting, check the Over Plot radio button within the Control Panel area. Overplotting is allowed only within the "Quick View" area. When floating plot window is selected and the "Over Plot" checkbox is checked, an info-window shows this message: Overplotting not allowed in floating windows and the last plot is redrawn within the "Quick View" area ready for overplotting. For this reason, when the Over Plot radio button is selected, the Select Plot Windows radio buttons are disabled.

During overplotting it is not possible to change the X-axis scale.

8.2.2.5. Table Plotter

To visualize the data with the Table Plotter the Table Plotter radio button should be checked within the Control Panel area.

When the Table Plotter radio button is selected, the desired detector inside Array Display area can be clicked on to visualize it.

Table Plotter objects can be created in both the "Quick View" area and in a floating window.

8.2.2.6. Color&Mask Preferences

The Color&Mask Preferences panel allows:

![Color&Mask Preferences panel](image)

Figure 8.20. Color&Mask Preferences panel.

1. selection of the scale color for the bolometer array visualization: heat or grey scheme are available;
2. selection of the Max and Min values for the colour scale;
3. work with the mask.
Colour Preferences

Using the Select color scale radio buttons it is possible to choose the color scheme between grey (default) and heat. The color scheme selected here is set as default for all the DTE instances (see also Section 8.2.3).

To select a min/max value for the color scale, either the writable fields or the sliders can be used. When the min value is changed, all the bolometers with a registered value below this limit are colored white. When the max value is changed, all the bolometers with a registered value over this limit are colored black. The new colours for the others bolometers are calculated using these new limits (Figure 8.21).

![Figure 8.21. Color scale Min/Max value example.](image)

Mask Preferences

It's possible that some detectors in the dataset are flagged because they have an error or a problem. This information is contained in the "Mask" dataset and with the DTE this information can be used to optimize the visualization of the data. The mask preferences show a list of mask values as defined by the SpireMask product (see Figure 8.20) and for each of these values there is a checkbox with a coloured button. Via this checkbox it's possible to select a specific mask value and via the colour button it is possible to assign a colour to the specific mask value.

The selection of a specific mask value causes the update of:

- **the display array**: the bolometers flagged (at the selected readout) by this mask are colored with the associated color (Figure 8.22). It is possible to use the slider (Figure 8.18) for browsing through time across the timeline.

- **the displayed plot**: the mask is overplotted on the current plotted data (Figure 8.23).

![Figure 8.22. Mask preferences: the selection of a specific mask value cause an update of the Array Display.](image)
The "mask oveplot" style is linked to the mask preferences: if the colour associated at the mask is changed, the layer's colour is also changed. Different mask values also have different plot symbols.

The Y axis on the right shows the bit number of the mask.

Figure 8.24. Mask preferences: displaying and plotting of a bolometer when multiple mask bits are set at the same time.
Using the buttons "Select all" and "Deselect all", it is possible to select or deselect all masks. This also changes the default behaviour for all instances of the DTE, see Section 8.2.3.

### 8.2.3. DTE Preferences

The HIPE preferences (see the HIPE Owner's Guide) contain a specific DTE section (see Figure 8.25) where it is possible to set the default behavior for the flag visualization and color scheme to use in the array display.

The defaults are also set using the Color&Mask Preferences panel in the main DTE GUI (see Section 8.2.2.6): i.e. if a user changes the color scheme in Color&Mask Preferences panel, the new value is set as default for all the DTE instances and the changes are reflected also in the DTE HIPE preferences. For mask visualization, click on "Select all" or "deselect all" in Color&Mask Preferences (see Section 8.2.2.6) to switch on/off the flag visualization in all DTE instances.

![Figure 8.25. Mask preferences: displaying and plotting of a bolometer when multiple mask bits are set at the same time.](image)

### 8.2.4. Example 1: Plotting functions

In order to inspect detectors from a specific product using the DTE, plotting functions perform the following steps:

1. Start the DTE for the product in question.
2. Click on the desired detector to create the plot.
3. Click on another detector to change the displayed plot.
4. Select the Over Plot radio button within the "Control Panel" area to allow overplotting (see Figure 8.26).
5. Click on another detector to overplot the data (see Figure 8.19).

6. To see the last plotted data within TablePlotter (Figure 8.19) select the Table Plotter radio button within the "Control Panel" area (see Figure 8.26).

7. To change the detector displayed with the Table Plotter, simply click on the desired detector in the Array Display.

Figure 8.26. Radio buttons to select the plotting options.

### 8.2.5. Example 2: Browse through time across the timeline.

In order to browse through time across the timeline, perform the following steps:

1. Start the DTE for the product in question.

2. In the Control Panel click on the ">>" button to go on to the next step of the timeline readout. (Figure 8.18)

3. To automize the browsing, click on the "PLAY" button. (Figure 8.18)

4. Use the "PAUSE" button to block the automatic browsing and the "STOP" button to come back to the beginning of the timeline. (Figure 8.18)

5. To change the speed of the automatic browsing, change the value in the "FPS" (frame per second) text field (default value = 30 fps, Figure 8.18).

### 8.2.6. Example 3: Visualize the flagged detectors in the Array Display

In order to Visualize the flagged detectors, perform the following steps:

1. Start the DTE for the product in question.

2. In the Control Panel click on the "Color&Mask Preferences" button to open the preferences panel (Figure 8.16).

3. Select the mask value of interest using the checkboxes: if a detector is flagged by this mask (at the selected timeline readout) its colour in the Array Display changes according to the associated color (Figure 8.27). All the mask values may be selected by simply clicking on the Select all button (and deselect all the mask values by clicking on the Deselect all button, Figure 8.20).

4. The colour associated with the mask can be changed by clicking on the coloured button and selecting the new colour from colour selector (Figure 8.28).

Figure 8.27. Mask preferences: two mask values with associated checkboxes and colour buttons.
8.3. SPIRE Bolometer Finder

8.3.1. Purpose

The Bolometer Finder allows us to view the timelines of individual bolometers that cross a selected map pixel. A Bolometer Finder Tool is available as GUI interface from the tasks list in HIPE as a front end to the BoloFinder task itself. The Bolometer Finder Tool can take as input either an Observation Context or a Time Ordered Data, TOD buffer, and display a map. Clicking on any map pixel will produce a graph of signal timelines for individual bolometers within the selected pixel. Individual data samples can then be flagged/masked and the Level 2, Level 1, Level 0.5 data updated. In this way it is possible to carry out interactive deglitching if necessary.

The Bolometer Finder Tool is the recommended method for utilizing the BoloFinder task and is described in Section 8.3.2. The BoloFinder task can also be accessed directly (see Section 8.3.2) or by using a SPIRE Useful Script (see Section 8.3.5).

8.3.2. The BoloFinderTool

The BoloFinderTool requires as input either an Observation Context (recommended) or a TOD buffer. Either one of these must be present in the Variables panel for the BoloFinderTool to be used. As an example, the Parallel Mode observation with obsid 1342227651 (0x5000C4C3) is read into HIPE into the variable obs using the getObservation command. Selecting the variable obs in the Variables panel and right clicking to access the sub-menu, we can select Open With -- Bolometer Finder Tool as in Figure 8.29:
Figure 8.29. Selecting the Bolometer Finder Tool from an Observation variable.

Selecting the Bolometer Finder Tool for the observation opens the new window for the BoloFinder shown in Figure 8.30. This is the Bolometer Finder Set Up Tab and from this window, the desired map for a given array can be selected for this observation. The map type can correspond to the point source (prefix psrc), extended (prefix ext), moving object frame SSO (prefix sso) maps from HIPE version 10 and later or for the pre-HIPE 10 maps with no map prefix, the NONE option can be selected. Clicking on Display Image will display the selected map type in the Image Tab of the Bolometer Finder window (Figure 8.31).

Figure 8.30. Bolometer Finder Set Up Tab.
Figure 8.31. Bolometer Finder Image Tab.

For this example, we will use the Bolometer Finder to correct residual glitch tails in the PSW map shown in Figure 8.31. From the image, the faint glitch tail can be seen extending diagonally over at least 3 pixels \((x, y = (175, 273), (176, 274), (177, 275))\). The location of the glitch is zoomed in the figure for clarity. The Bolometer Finder can be used to both identify and mask the offending bolometer timelines. To inspect the bolometer timeline samples within any given pixel in the map, simply click on a map pixel to select it. Clicking on a map pixel opens a timeline sample viewer in the Bolometer Data Tab of the Bolometer Finder window as shown in Figure 8.32. All bolometers are arbitrarily colour coded and the sample points are shown as symbols, joined by lines in time sequence. Individual bolometers can be isolated using the All Bolometers pull down menu at the bottom of the window in Figure 8.32. Isolating the outlying bolometers, allows the individual sample points to be masked (using the 2nd level glitch mask) and unmasked accordingly. In Figure 8.33 the process of selecting each pixel, masking the appropriate timelines and updating the map are shown for each of the 3 pixels containing the residual glitch tail. Note the signal for the outlying bolometers traces the glitch tail from 0.5 - 0.175 Jy/Beam in the timelines over the 3 timeline plots. The procedure, outlined in the figure would be to:

- Select a pixel from the Bolometer Finder Image Tab.
- Inspect the Bolometer Data Tab for outlying bolometers.
- Select the outlying bolometer from the pull down list.
- Mask the bolometer samples by clicking on each point (the joining line will disappear).
- Click on the Update Map button to remake the map from the updated timeline data.
- Repeat above steps for all affected pixels. Note that you will be asked whether you want to save the changes to the mask you have made.

From the final image panel in Figure 8.33, it can be seen that the glitch tail has been removed from the final map by masking of the outlying bolometers using the Bolometer Finder. Note that although the map is updated at each iteration, the observation still has to be saved back to disk to make the changes permanent.
Figure 8.32. Bolometer Finder Timeline Sample Tab.

Figure 8.33. Interactive Deglitching using the Bolometer Finder.
8.3.3. Interacting with the BoloFinder via Scripting

Using the BoloFinder task directly requires the Todbuffer corresponding to the map as its primary input - this will require the use of createTodBuffer task (and probably the baselineRemovalMedian) and naïveScanMapper to create the required inputs from the SPIRE Level 1 data.

There are various ways of defining the map pixel of interest: clicking on a map, specifying a map pixel in celestial coordinates, or inputting the map x and y coordinates. Additionally, from the HIPE Scripts top-level menu, the Photometer Bolometer Finder script (under SPIRE Useful Scripts) will allow clicking on as many points on a map as desired, with plotting of the signals that contribute to the clicked-on map pixel.

The output from the task is of type BoloFinder which is an Instance of the task API class. This means that the output can either be simply printed to get a summary OR analysed further within HIPE.

The following recipe shows the typical way of running the Bolometer Finder at the present time. The commands create the TodBuffer and NaiveMapper map for an image. BoloFinderTask (invoked by the HIPE command out=boloFinder(…) line) then displays the map image, waits for a mouse-click at any point on the map and returns the details in the object out.

Simply “printing” the output lists the contributions to the selected map pixel, as shown in the example below.

```python
# Retrieve sample observation
obs = getObservation(1342211401, useHsa=True)

# Baseline correct
scans = baselineRemovalMedian(obs.level1)

# Make TodBuffer and Map
tb = createTodBuffer(scans)
mymap = naiveScanMapper(tb)

# Run BoloFinder with display of map
# Left click on any point in the map to proceed
outBolo = boloFinder(tb, map=mymap, displayMap=True)

# Print default results summary
print outBolo
```

```
#Data samples contributing to map pixel 10974 Map (row,col) = (64,94)
#Start data sample #22732 - PSHW14  Scan - 0 #data samples - 2
#Start data sample #26656 - PSHW11  Scan - 0 #data samples - 1
#Start data sample #44191 - PSHW8  Scan - 0 #data samples - 4
#Start data sample #74684 - PSHJ7  Scan - 1 #data samples - 1
#Start data sample #86205 - PSHW8  Scan - 1 #data samples - 1
#Start data sample #92664 - PSHW10 Scan - 1 #data samples - 4
#Start data sample #133491 - PSHW14 Scan - 2 #data samples - 2
#Start data sample #137415 - PSHW11 Scan - 2 #data samples - 2
#Start data sample #154950 - PSHW8  Scan - 2 #data samples - 4
#Start data sample #185443 - PSHJ7  Scan - 3 #data samples - 1
#Start data sample #196964 - PSHW8  Scan - 3 #data samples - 1
#Start data sample #203423 - PSHW10 Scan - 3 #data samples - 4
#Start data sample #244250 - PSHW14 Scan - 4 #data samples - 2
#Start data sample #248174 - PSHW11 Scan - 4 #data samples - 2
#Start data sample #265709 - PSHW8  Scan - 4 #data samples - 2
#Start data sample #296202 - PSHJ7  Scan - 5 #data samples - 1
#Start data sample #307723 - PSHW8  Scan - 5 #data samples - 1
#Start data sample #314182 - PSHW10 Scan - 5 #data samples - 4
#Start data sample #355333 - PSHW14 Scan - 6 #data samples - 2
#Start data sample #359311 - PSHW11 Scan - 6 #data samples - 2
#Start data sample #377092 - PSHW8  Scan - 6 #data samples - 4
#Start data sample #408017 - PSHJ7  Scan - 7 #data samples - 1
#Start data sample #419700 - PSHW8  Scan - 7 #data samples - 1
#Start data sample #426249 - PSHW10 Scan - 7 #data samples - 4
```

Figure 8.34 shows the interactive display just before clicking on the map pixel of interest.
Figure 8.34. Running the BoloFinder with displayMap=True. Left-click on the desired map pixel to extract all the timelines that contributed flux to that pixel.

Using the the BoloFinder API, the data can be extracted from the results object ("outBolo") obtained in the example above.

```python
# Get names of the contributing bolometers into a HIPE array "names"
names = outBolo.boloNames
print names[1]  # Output: PSWD11
# Get detailed data on PSWD11 into data array
pswd11 = outBolo.getBoloData("PSWD11")
# Data returned in an array - show this
print len(pswd11)  # Output: 7
# Data summary for one data sample
print pswd11[0]
# Output:
# PSWD11 - 10974/26656/244 Group=1 map(row,col) = (64,94) Scan=0 Signal=0.04 Weight=1.0
# Extract individual data values
print pswd11[0].boloName  # Output: PSWD11
print pswd11[0].mapPixel
print pswd11[0].mapRow
print pswd11[0].mapCol
print pswd11[0].weight
print pswd11[0].sampleIndex
print pswd11[0].scanNumber
print pswd11[0].llDataIndex
# Output: 26656
# Compute statistics for the clicked-upon pixel
mapvals = Double1d()
for name in names:
dvals = outBolo.getBoloData(name)
for i in range(len(dvals)):
    if dvals[i].weight > 0.0:
        mapvals.append(dvals[i].signal)

print 'mean = %.4f, median = %.4f, sigma = %.4f Jy' % 
    (MEAN(mapvals), MEDIAN(mapvals), STDDEV(mapvals)/SQRT(len(names)))
# Output: mean = 0.0333, median = 0.0363, sigma = 0.0171 Jy

8.3.4. Specifying the map pixel via the command line

The map pixel may also be specified in row and column map pixel coordinates, or by Right Ascension and Declination, as shown in the following code block.

# Specify by map pixel
outBolo = boloFinder(tb, map=mymap)
outBolo.locateXY(94,64)
print outBolo
# outputs the following to the HIPE console
# outputs the following to the HIPE console
# Data samples contributing to map pixel 10974 Map = (94,64)
# Start data sample #22732 - PSWH14 Scan - 0 #data samples - 2
# Start data sample #26656 - PSWD11 Scan - 0 #data samples - 1
# Start data sample #41910 - PSWA8 Scan - 0 #data samples - 5
# Start data sample #47684 - PSWJ7 Scan - 1 #data samples - 1
# Start data sample #66205 - PSWH8 Scan - 1 #data samples - 1
# Start data sample #92664 - PSWE10 Scan - 1 #data samples - 4
# Start data sample #133491 - PSWH14 Scan - 2 #data samples - 2
# Start data sample #137415 - PSWD11 Scan - 2 #data samples - 2
# Start data sample #154949 - PSWA8 Scan - 2 #data samples - 5
# Start data sample #185443 - PSWJ7 Scan - 3 #data samples - 1
# Start data sample #196964 - PSWH8 Scan - 3 #data samples - 1
# Start data sample #203423 - PSWE10 Scan - 3 #data samples - 4
# Start data sample #248174 - PSWD11 Scan - 4 #data samples - 1
# Start data sample #265708 - PSWA8 Scan - 4 #data samples - 5
# Start data sample #296202 - PSWJ7 Scan - 5 #data samples - 1
# Start data sample #307723 - PSWH8 Scan - 5 #data samples - 1
# Start data sample #314182 - PSWE10 Scan - 5 #data samples - 4
# Start data sample #355333 - PSWH14 Scan - 6 #data samples - 2
# Start data sample #359311 - PSWD11 Scan - 6 #data samples - 2
# Start data sample #377091 - PSWA8 Scan - 6 #data samples - 5
# Start data sample #408017 - PSWJ7 Scan - 7 #data samples - 1
# Start data sample #419700 - PSWH8 Scan - 7 #data samples - 1
# Start data sample #426249 - PSWE10 Scan - 7 #data samples - 4
# Specify by RA and Dec
raStr = '17:39:55.5'
dercStr = '+69:00:12.2'
from herschel.calsdb.util import Coordinate
ra = Coordinate.ra2Double(raStr)
derc = Coordinate.dec2Double(decStr)
outBolo.locateRaDec(ra,dec)
print outBolo
# Data samples contributing to map pixel 10974 Map (row,col) = (64,94)
# Start data sample #22732 - PSWH14 Scan - 0 #data samples - 2
# Start data sample #26656 - PSWD11 Scan - 0 #data samples - 1
# Start data sample #41910 - PSWA8 Scan - 0 #data samples - 4
# Start data sample #47684 - PSWJ7 Scan - 1 #data samples - 1
# Start data sample #66205 - PSWH8 Scan - 1 #data samples - 1
# Start data sample #92664 - PSWE10 Scan - 1 #data samples - 4
# Start data sample #133491 - PSWH14 Scan - 2 #data samples - 2
# Start data sample #137415 - PSWD11 Scan - 2 #data samples - 2
# Start data sample #154949 - PSWA8 Scan - 2 #data samples - 5
# Start data sample #185443 - PSWJ7 Scan - 3 #data samples - 1
# Start data sample #196964 - PSWH8 Scan - 3 #data samples - 1
# Start data sample #203423 - PSWE10 Scan - 3 #data samples - 4
# Start data sample #248174 - PSWD11 Scan - 4 #data samples - 1
# Start data sample #265708 - PSWA8 Scan - 4 #data samples - 5
# Start data sample #296202 - PSWJ7 Scan - 5 #data samples - 1
# Start data sample #307723 - PSWH8 Scan - 5 #data samples - 1
# Start data sample #314182 - PSWE10 Scan - 5 #data samples - 4
# Start data sample #355333 - PSWH14 Scan - 6 #data samples - 2
# Start data sample #359311 - PSWD11 Scan - 6 #data samples - 2
# Start data sample #377091 - PSWA8 Scan - 6 #data samples - 5
# Start data sample #408017 - PSWJ7 Scan - 7 #data samples - 1
# Start data sample #419700 - PSWH8 Scan - 7 #data samples - 1
# Start data sample #426249 - PSWE10 Scan - 7 #data samples - 4
8.3.5. Visualising timelines with the plotAllData() method

As of HIPE 10.0, the Bolometer Finder output includes the plotAllData() method to display a plot of all timelines that contribute data to the specified map pixel.

Continuing the example in the preceding section, the plot can be displayed with this command:

```python
outBolo.plotAllData()
```

Then the timeline plot is displayed, as shown in Figure 8.35. The legend gives information on each detector-scanline pair and the number of samples from that timeline that contribute to the specified map pixel. The zooming and editing features of PlotXY plots are available as described in the plotting chapter of the Data Analysis Guide.

![Timeline Plot](image_url)

**Figure 8.35. Example of displaying a timeline plot with the plotAllData method**
8.3.6. Identifying outliers with the Bolometer Finder Useful Script and the SPIRE Mask Editor

As a complement to the plotting capabilities provided by the Bolometer Finder class, a script is available for plotting only the timeline samples contributing to a map pixel. The map and plots are displayed in a persistent GUI window. The map may be clicked upon as many times as desired, with the timelines crossing that map pixel displayed on the right-hand side as shown in Figure 8.36.

The script is available from the top level menu Scripts → SPIRE Useful Scripts → Photometer Bolometer Finder in HIPE. To run the script, it is necessary to define variables "scans", "array" and "mymap" like so:

```python
# example of setting up the variables
obs = getObservation(1342211401, useHsa=True)
scans = baselineRemovalMedian(obs.level1)
array = 'PSW'
```

Then the entire script can be executed, to bring up the GUI shown in Figure 8.36. Click on any map pixel to display the signal plot. The x-axis of the timeline plot is the radial distance of the sample from the center of the selected pixel, in units of arcseconds. If multiple samples from a timeline contribute to the pixel, the points are connected by lines. The legend lists the detector name, the scan number (indexed from 0), and the starting index into the corresponding Level 1 product (PointedPhotTimeline). The example in shows the error map in the left panel (selected from the drop-down menu at upper right of the GUI) and the signal plot on the right side after clicking on the start of a bright trail in the error map.

![Figure 8.36. Example of running the Photometer Bolometer Finder Useful Script, for the error plane of the PSW map of obsid 1342211401](image)

The information given in the legend can be used together with the SPIRE Mask Editor to view and mask the residual glitch. In the example, the artifact is in PSWC8, 7th scan line, 431st sample. The Mask Editor is described in Section 8.4.2. The following procedure may be used to select and mask the artifact.

- In the HIPE Variables view, right-click on your observation context variable to bring up the Level 1 Mask Editor. Select Bit 11, Glitch L1Detected. Enter the scan number at the top and the detector name at the bottom. Optionally, enter the starting index at the bottom. Click on Show/Edit to display the specified timeline.

- A TablePlotter window will appear. Zoom in on the glitch and then select the “Hide” button, then draw a box around the samples you want to mask. When you are done masking, close the TablePlotter window.
In the Mask Editor window, press "Save" to save the updated masks. Repeat the process with other detectors and scans as needed.

When finished with the Mask Editor, press "Exit". Your observation context can be reprocessed to make an improved map.

8.4. SPIRE Mask Handling

8.4.1. SPIRE Mask Formalization

SPIRE data contains various kinds of (Boolean) masks, used to indicate problems or caveats associated with the data. For consistency, SPIRE Masks follow the rule that the Boolean false state represents that a mask condition is not applicable.

The SPIRE data processing supports three general categories of masks.

- **Channel Masks**: are applied to all data from a given channel (bolometer, thermistor, resistor, etc.) and thus affects all time samples for that channel. The Channel Masks are contained within the Channel Mask Table calibration product (see RD1 for definition). They indicate whether a signal channel is useful or not, such as whether that particular channel is dead.

- **Instrument Mode Masks**: are effectively channel masks that are applied only to data from specific observing modes. The observing mode masks are contained within the Instrument Mode Mask Table calibration product (see RD1 for definition). Following the same procedure as for Channel Masks, they indicate whether a signal channel is useful or not for that specific observing mode.

- **Sample Masks**: All data products contain a sample mask table added at the Reformat Level 0 Product stage. In this table, a 32-bit integer is reserved for each sample for each detector, referred to as a Sample Mask. Mask information is represented by bits in a Sample Mask, with different bits representing different mask conditions.

Much of the sample mask in the data is populated during the Level 0 - Level 0.5 processing pipeline using 2 calibration products (Channel Mask Calibration Product and the Instrument Mode Mask Calibration Product). All masks fall into 1 of 3 categories depending on their impact and severity;

- **Unusable**: reserved for data samples that are afflicted with critical problems to the degree that the data samples should not be considered scientifically valid and are ignored by the pipeline modules.

- **Informational**: represent non-critical problems with the detector data samples for which there exists no correcting data processing module. Data processing modules that encounter an Informational condition should process the sample as normal and propagate the Sample Mask Table.

- **Correctable**: conditions for which a data processing module exists that may be able to correct the condition. These masks always come in pairs: one mask will denote the identification of the condition; the second will denote whether the condition has been corrected.

Note that in addition to the individual mask types, there is also a **Master Bit** to provide a quick reference as to whether a data sample is or is not scientifically valid which is set if any unusable flag is raised. Masks are stored as a bit mask and are described below in Table 8.1 and Table 8.2 (detailed information can be found in the "SPIRE Pipeline Mask Policy" document SPIRE-BSS-DOC-003127);

<table>
<thead>
<tr>
<th>Bit no.</th>
<th>Bit value</th>
<th>Mask Name</th>
<th>Mask Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>MASTER</td>
<td>unusable</td>
<td>If set, then the data should not be used</td>
</tr>
<tr>
<td>Bit no.</td>
<td>Bit value</td>
<td>Mask Name</td>
<td>Mask Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-----------</td>
<td>---------------------------</td>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>INVALID_TIME</td>
<td>unusable</td>
<td>Set in Level 0-0.5 pipeline for sample has an invalid time</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>ADC_LATCH</td>
<td>unusable</td>
<td>Set in Level 0-0.5 pipeline for possible ADC latchup error</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>TRUNCATED</td>
<td>correctable</td>
<td>Set in Level 0-0.5 pipeline for ADC conversion truncated (saturated)</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>TRUNCATED_UNCORR</td>
<td>correctable</td>
<td>Set in Clipping Correction module for ADC truncation not corrected</td>
</tr>
<tr>
<td>5</td>
<td>32</td>
<td>GLITCH</td>
<td>........</td>
<td>not used</td>
</tr>
<tr>
<td>6</td>
<td>64</td>
<td>GLITCH_UNCORR</td>
<td>........</td>
<td>not used</td>
</tr>
<tr>
<td>7</td>
<td>128</td>
<td>ISDEAD</td>
<td>unusable</td>
<td>Set in Level 0-0.5 pipeline for dead detector</td>
</tr>
<tr>
<td>8</td>
<td>256</td>
<td>ISNOISY</td>
<td>informational</td>
<td>Set in Level 0-0.5 pipeline for noisy detector</td>
</tr>
<tr>
<td>9</td>
<td>512</td>
<td>ISNOCHOPSKY</td>
<td>informational</td>
<td>Set in Level 0-0.5 pipeline for channel chopped off array FoV</td>
</tr>
<tr>
<td>10</td>
<td>1024</td>
<td>VOLTAGE_OOL</td>
<td>informational</td>
<td>Set in Flux-Conversion (phot) or Non-linearity (spec) module for a voltage value outside range of fitted bolometer responsivity curve</td>
</tr>
<tr>
<td>11</td>
<td>2048</td>
<td>GLITCH_FIRST_LEVEL</td>
<td>correctable</td>
<td>Glitch detected in 1st level deglitching</td>
</tr>
<tr>
<td>12</td>
<td>4096</td>
<td>GLITCH_FIRST_LEVEL_UNCORR</td>
<td>correctable</td>
<td>Glitch detected in 1st level deglitching not removed</td>
</tr>
<tr>
<td>13</td>
<td>8192</td>
<td>GLITCH_SECOND_LEVEL</td>
<td>correctable</td>
<td>Glitch detected in 2nd level deglitching</td>
</tr>
<tr>
<td>14</td>
<td>16384</td>
<td>GLITCH_SECOND_LEVEL_UNCORR</td>
<td>correctable</td>
<td>Glitch detected in 1st level deglitching not removed</td>
</tr>
<tr>
<td>15</td>
<td>32768</td>
<td>ISSLOW</td>
<td>informational</td>
<td>Set in Level 0-0.5 pipeline if detector deemed as slow</td>
</tr>
<tr>
<td>16</td>
<td>65536</td>
<td>VOLTAGE_BELOW_K3</td>
<td>unusable</td>
<td>Flux conversion (Photometer) or Non-linearity correction (Spectrometer) module encountered</td>
</tr>
<tr>
<td>Bit no.</td>
<td>Bit value</td>
<td>Mask Name</td>
<td>Mask Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-----------</td>
<td>-------------------------</td>
<td>---------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>17</td>
<td>131072</td>
<td>NO_RESP_DATA</td>
<td>unusable</td>
<td>Set in Level 0-0.5 pipeline if flux conversion (Photometer) or linearization (Spectrometer) is not possible</td>
</tr>
<tr>
<td>18</td>
<td>262144</td>
<td>TSIGNAL_HDV</td>
<td>informational</td>
<td>thermistor/DP signal deviations larger than expected from Temperature Drift Correction module</td>
</tr>
<tr>
<td>19</td>
<td>524288</td>
<td>BSM_CHOP_OOL</td>
<td>informational</td>
<td>Set in BSM module if chop motion outside soft limits</td>
</tr>
<tr>
<td>20</td>
<td>1048576</td>
<td>BSM_JIGG_OOL</td>
<td>informational</td>
<td>Set in BSM module if jiggle motion outside soft limits</td>
</tr>
<tr>
<td>21</td>
<td>2097152</td>
<td>JUMP_THERMISTORS_</td>
<td>informational</td>
<td>Set by Jump Detection module for sudden signal jumps in the timelines of thermistors or dark channels. Used by the Temperature Drift Correction module</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DARKS_SIGNAL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>4194304</td>
<td>NO_THERMISTOR_AVAILABLE</td>
<td>informational</td>
<td>Set by Temperature Drift Correction module if all available thermistors are affected by saturation or signal jump. Temperature Drift Correction module will try to use the dark channels.</td>
</tr>
<tr>
<td>23</td>
<td>8388608</td>
<td>NON_NOMINAL_VELOCITY</td>
<td>informational</td>
<td>Set by Associate Sky Position module for all data between nominal scan legs</td>
</tr>
<tr>
<td>24</td>
<td>16777216</td>
<td>NO_DARKCHANNEL_AVAILABLE</td>
<td>informational</td>
<td>Set by Temperature Drift Correction module if all available dark channels are affected by saturation or signal jump. In this case the module will not be able to correct the corresponding building block</td>
</tr>
</tbody>
</table>

**8.4.2. SPIRE Mask Editor Viewer**

The mask and the masked data can be investigated in detail - either by using the Detector Timeline Viewer (see Section 8.2), the SPIRE Mask Editor Viewer, or by writing a script to examine the Mask Table of the detector timelines, etc.

From HIPE v9 onwards, two SpireMask editors, "Level0.5 SpireMaskEditor" and "Level1 SpireMaskEditor", are available. The "Level0.5 SpireMaskEditor" works for Level-0.5 data and "Level1 SpireMaskEditor" works for Photometer Level-1 data. From HIPE v11 onwards, the Spectrometer...
"Level1 MaskEditor" is also available, and it works for Spectrometer Level-1 interferogram data. The viewers are launched from HIPE by highlighting the Observation Context in the Variables tab, and then selecting the "open with" menu item by right clicking the mouse button. The Level0.5 SpireMaskEditor, Level1 SpireMaskEditor or Spectrometer Level1 MaskEditor should be available in the list. Note that the Level-0.5 or Level-1 context must have been loaded into HIPE for the mask editor to appear in the list. The GUI will pop up in the HIPE editor tab, for example, see Figure 8.37 and Figure 8.38. The viewers list all the masks from Table 8.1 and Table 8.2.

The circular radio buttons on the left select which mask bit will be used with the buttons in the right column of the panel (Show/Edit, Set All, etc.). The squares will light up green if the corresponding mask bit in the selected read-out (lower right selector) is set and they can also be edited (toggled) by a mouse click to switch them on and off. The white fields show the number of masks that have the corresponding bit set for the selected building block. The individual mask bit for each read-out can be edited by mouse click. The whole mask array for a selected mask bit and a selected channel can be edited by using Set All, Unset All, and Invert All buttons. The same changes can be also achieved by using Show/Edit button through marking an area to select/de-select.

Note that the viewers are no longer accessible by right clicking the mouse on the Observation Context if the level-x data is not loaded (if the context has not been loaded, it will appear in red in the Observation Context Viewer).
Figure 8.37. The SPIRE Mask Editor
Figure 8.38. The SPIRE Level 1 Spectrometer Mask Editor

The example shown in Figure 8.37 is for a SPIRE Spectrometer mapping observation (OBSID 0x50002650 in hex or 1342187088 in decimal). In this observation some of the detectors have been saturated such that some clipping has occurred. In order to show the samples that have been flagged, the procedure in the spireMaskEditor is: select the FTS scan building block (0xa10600x) in the drop down menu of BBIDs at the top of the editor window, select the truncated mask bit by clicking on the radio button next to it, click Show/Edit on the right hand side of the window – this will pop up a Table Plotter window, in which the detector plotted can be selected. Masked data samples are indicated with a red cross. In the example in Figure 8.39 it is clear that both the maximum and minimum of the interferogram have been flagged as clipped (the red crosses are the flagged data points). In the Table Plotter, mask bits can be set by clicking the Hide button followed by dragging a rectangle around the signals where the mask bit should be set. Mask bits can be unset by using the Unhide button instead. Upon closing the Table Plotter the changes will already be reflected in the main MaskEditorTool panel, but they will only be committed to the observation context if the Save button is used to finally close the MaskEditorTool.
8.4.3. Other ways to view SPIRE masks

Note that the SPIRE Mask Editor Tool is not the only way to view and manipulate masks. The Detector Timeline Viewer (See Section 8.2) can also be used to display flagged samples. For the above example, to open this tool, right click on the Spectrometer Detector Timeline (SDT; or equivalent PDT for the photometer) in the Variables tab of HIPE and openWith Detector Timeline Viewer. Mask bits can be selected using the Color&Mask Preferences button, and the flagged samples are indicated in the plot.

The same information can be obtained in a script by accessing the mask table of each dataset. The method of accessing the mask table for our example of the Spectrometer Detector Timeline (SDT) is shown below but the principal applies equally to photometer observations. The following lines of code operate on the SDT to create a list of detectors with clipped samples:

```python
clipped = []
notclipped = []
for chan in sdt.channelNames:
    clippedSamples = sdt['mask'][chan].data.where(SpireMask.TRUNCATED)
    if clippedSamples.length():
        clipped.append(chan)
    else:
        notclipped.append(chan)
```

A similar method can be used to examine each scan in the SDI product – for example, to find the clipped samples in scan 0001 for detector SSWD4:

```python
clippedSamples=sd['0001']['SSWD4']['mask'].data.where(SpireMask.TRUNCATED)
```
8.5. Quality Control for SPIRE pipeline processed data

8.5.1. Quality Control for SPIRE pipeline processed data

Quality control refers to the assessment of whether data from the SPIRE data processing pipelines is usable for science. The SPIRE quality control system is designed to set flags for data where hardware problems or observing conditions may have rendered the data unusable for science. These flags will have associated text messages that indicate the nature of the problem. Data where these flags are set must then be inspected visually by the support scientists to ascertain whether the observations should be repeated.

Quality control should be used to access data from either the whole SPIRE instrument or individual detector arrays as a whole. It should not be used to assess data from individual detectors. While identifying problems with individual detectors may be important, the goal of quality control, as described here, is to access the usability of observations performed with the instrument as a whole, not individual detectors (unless the usability of data from an observation was contingent on the performance of specific individual detectors).

Quality control does not refer to the long-term monitoring of the condition of the telescope or the instrument (for example, variations in detector responsivity over time). Quality control should also not be used merely to provide information (for example, uncertainties in nonlinearity corrections), although quality control data may also serve an informative purpose (for example, glitch rate measurements). Quality control also should not necessarily assess whether the scientific goals of the observation have been achieved (for example, S/N ratios in final scan maps).

A full treatment of the various SPIRE quality control pipelines for the photometer and the spectrometer is beyond the scope of this document. However, the relevant document, "Quality Control Metrics for SPIRE Data Processing Pipelines" is available on the public SPIRE wiki page at the Herschel Science Center.
Chapter 9. Advanced HIPE Tips

9.1. Saving an observation context to a pool on disk

This saves an Observation Context into a pool on the local disk (e.g. after downloading from the HSA into HIPE). Note that it is important for some parts of the pipeline (e.g. the engineering conversion, and spectrometer bright mode processing) to have the calibration history files available and so saveCalTree should normally be set to True (see also Section 5.4):

```python
saveObservation(obs, poolName="myPool", saveCalTree=True)
```

Note that the recommended way to save an observation to a pool is using `saveProduct`:

```python
saveProduct(obs, pool="myPool", tag="My reprocessed data")
```

9.2. Using TEMP STORAGE to overcome memory problems with HIPE

HIPE can successfully process many observation with 2GB of memory allocated to it. Larger observations may require 4GB but in some rare cases (most often with Parallel Mode Scan Map observations) HIPE can run out of memory and unceremoniously crash. In such cases, it is possible to circumvent this memory problem by using a "TemporalPool". Use of a "TemporalPool" requires some modification to the User script as detailed in the 5 steps below, using the Photometer Large Map Script as an example. Note that care must be taken to preserve the correct indentations. An overview of where the code sections go in the user script is shown in Figure 9.1

1) Import the necessary Tasks: Place the following lines immediately before the User Selectable Options:

```python
#*Temporal Pool part 1*
from herschel.ia.pg import ProductSink
from herschel.ia.pal.pool.lstore.util import TemporalPool
***
```

2) Set a BOOLEAN flag: Add a flag in the Additional Options section in the script (i.e. after optin"D"):

```python
#*Temporal Pool part 2*
# (E) Use a productSink  (Temporal storage in case of memory problems)
# Set this to FALSE if you don't want to use the ProductSink
# and do all the processing in memory
tempStorage=Boolean.TRUE
***
```

3) Initialize the Temporal Pool: Add the following section of code after `level1=Level1Context(myObsid)` and before the main loop in the script:

```python
#*Temporal Pool part 3*
if tempStorage:
    pname="tmp"+hex(System.currentTimeMillis())[2:-1]
```
4) Store each Level 1 Scan Line product in Temporal Store: Replace the code line `level1.addProduct(psp)` with the following section of code:

```java
#*Temporal Pool part 4*
if tempStorage:
    ref=ProductSink.getInstance().save(psp)
    level1.addRef(ref)
else:
    level1.addProduct(psp)
```

5) Store each Baseline subtracted Scan Line in Temporal Store: Replace the code line in the Baseline removal loop `scans.addProduct(psp)` with the following section of code:

```java
#*Temporal Pool part 5*
if tempStorage:
    ref=ProductSink.getInstance().save(psp)
    scans.addRef(ref)
else:
    scans.addProduct(psp)
```
9.3. Filtering variables in the HIPE variable list

After running any pipeline script, the Variables window in HIPE often contains a large number of entries produced by running the pipeline process. Although some of these may be discernable, many others may not be so clear. Moreover, it can be hard to identify the products you want to view from the long entry list. HIPE provides a quick and easy filtering system to reduce the variables displayed in the Variables window. In Figure 9.2 the Variables window after finishing the processing for the SPIRE Large Map example in Section 6.5.1 is shown, listing all the variables produced during the processing. Clicking on the top right hand arrow as shown in the figure, we can...
choose to filter the products only, or datasets or numerics only. Selecting Filter Products as shown in the second panel of the figure results in only products from the processing being listed. The pipeline script does not produce any table datasets (only observational data in the form of products is produced), therefore selecting Filter Dataset as shown in the right panel of the figure results in an empty variable list.

Figure 9.2. Control of variable visibility in HIPE
Chapter 10. Glossary

AOT : Astronomical Observing Template
APPP : Averaged Photometer Pointing Product
Bb : Building Block
BSM : Beam Steering Mechanism
BSMT : Beam Steering Mechanism Timeline
CHKT : Critical House Keeping Timeline
DEC : Declination
DS9 : An astronomical imaging and data visualization application
FFT : Fast Fourier Transform
FTTS : Flexible Image Transport System
FT : Fourier Transform
FTS : Fourier Transform Spectrometer
FWHM : Full Width at Half Maximum
HCSS : Herschel Common Science System
HIPE : Herschel Interactive Processing Environment
HSA : Herschel Science Archive
HSC : Herschel Science Center
ICC : Instrument Control Centre
IFTS : Imaging Fourier Transform Spectrometer
JPP : Jiggle Pointing Product
LSR : Local Standard of Rest
MCUET: Mechanism Control Unit Engineering Timeline
NHKT : Nominal House Keeping Timeline
ObsID: Observation ID
OD : Observing Day
OPD : Optical Path Difference
PDT : Photometer Detector Timeline
PLW : Photometer Long (500 µm) Wavelength
PMW : Photometer Short (350 µm) Wavelength
POF : Photometer Observing Functions
**POT**: Photometer Offset Timeline

**PSW**: Photometer Short (250 µm) Wavelength

**PVP**: Performance Verification Phase

**RSRF**: Relative Spectral Response Function

**SCAL**: Spectrometer Calibration Source

**SCUT**: Sub Control Unit Timeline

**SDI**: Spectrometer Detector Interferogram

**SDP**: Science Demonstration Phase

**SDRG**: SPIRE Data Reduction Guide

**SDS**: Spectrometer Detector Spectrum

**SDT**: Spectrometer Detector Timeline

**SIAM**: Spacecraft Instrument Alignment Matrix

**SLW**: Spectrometer Long Wavelength

**SMEC**: Spectrometer Mechanism

**SMECT**: Spectrometer Mechanism Timeline

**SOF**: Spectrometer Observing Functions

**SOT**: Spectrometer Offset Timeline

**SPIRE**: Spectral and Photometric Imaging Receiver

**SPSM**: SPIRE Pipeline Specification Manual

**SSW**: Spectrometer Short Wavelength

**TAI**: International Atomic Time

**UTC**: Coordinated Universal Time

**ZPD**: Zero Path Difference
Chapter 11. Reprocessing with the SPIA

11.1. Overview

The Herschel Interactive Processing Environment (HIPE) is a versatile program, including mechanisms for data storage and retrieval, graphical interfacing of tasks, a scripting language, a numerical library handling multidimensional arrays, and pipeline scripts that enable observations to be reprocessed in a standard way. Although HIPE includes convenience tools such as data product viewers and plotters for table datasets, most of the interaction with data remains script-based. Scripts allow maximum versatility, but have drawbacks such as being prone to error and requiring substantial training to use. GUIs allow less versatility but are easier to handle and provide better cues to remember functionality after not being used for a time.

The SPIRE Photometer Interactive Analysis (SPIA) package provides a structured GUI-based access to the more intricate parts of the scan map photometer pipeline for SPIRE. The SPIA tasks cast the data analysis steps for SPIRE scan maps into logical modules allowing interactive access to many options. The package consists of tailored functions for I/O, Level 1, and Level 2 processing of SPIRE photometer scan map datasets. Executing a task by hitting the Accept button in the GUI creates a command line that can be included into a Jython script to repeat the same reduction procedure on other datasets. Thus the interactive analysis can be used as a pathfinder to optimize the data reduction, producing template scripts for later automatic bulk processing of larger datasets without obligating the astronomer to learn a lot about scripting.

This manual describes the ‘why’ and ‘how’ of using the SPIA package to process SPIRE photometer data into maps. The SPIA encapsulates the processing steps described earlier in this SPIRE Data Reduction Guide, and in the SPIRE Pipeline Specification Manual. The NASA Herschel Science Center hosts the SPIA website. Additional information can be found via a preprint at this link.

An implementation of this concept for the spectrometer part of SPIRE is anticipated. Additional extensions for further analysis of SPIRE maps may be added in future versions.

Section 11.2 lays out the benefits to using the SPIA for photometer reprocessing, Section 11.3 outlines the typical workflows that a user will need to consider when reprocessing photometer observations. The remaining sections describe how to perform specific tasks with the SPIA package.

11.2. Why reprocess with the SPIA?

The SPIA provides a path in-between the official SPIRE pipeline scripts, and customized scripts. The tasks used in the SPIRE pipeline are encapsulated into a small number of SPIA tasks. Reprocessing SPIRE photometry data with SPIA provides benefits to beginners and advanced users alike:

• For the beginner...
  • Pipeline processing is easy to run and re-run.
  • Key parameters generally appear on the first tab of each GUI.
  • It is easy to get back to the default parameter values.
  • It is easy to see which parameters have been changed from default values.
  • Tooltips are provided for each task and each parameter.

• For the advanced user...
• Nearly all parameters are accessible through the GUI.

• Reprocessing many observations is simplified into a few lines of a script.

• Experimenting with non-default parameter values is straightforward.

• Inspecting intermediate products for artifacts is built into one of the SPIA workflows.

• SPIA processing steps can be mixed with custom scripts.

For all users...

• Running each task in GUI form generates a simple command-line equivalent, which can be copied into a script for automating repetitive processing.

• Optimized convenience functions are provided for retrieving, inspecting and saving SPIRE data.

• Processing improvements sometimes become available in the SPIA plug-in before they appear in official scripts.

11.3. SPIA Setup and Workflows

Setting up HIPE for processing with the SPIA plug-in involves a few steps (see the leftmost column of Figure 11.1 for a chart):

• Locating the SPIA tasks within HIPE (Section 11.4).

• Downloading the SPIRE calibration tree (Section 11.7).

• Retrieving observations from the Herschel Science Archive into local pools (Section 11.5).
Figure 11.1. Common workflows for using the SPIA tasks. The bold headings correspond to sections of this manual, and are followed by the corresponding SPIA task names. The leftmost workflow is for setup and must be performed before the other ones. The middle path is for processing straight through to final maps. The rightmost flow includes the steps for manually flagging artifacts in intermediate products.

The most common workflow in processing an observation is shown in the middle flowchart in Figure 11.1:

- Loading your observation into your HIPE session (Section 11.6).
- Loading SPIRE calibration information into HIPE (Section 11.8).
- Reprocessing your data to calibrated timelines (Section 11.9).
- Making maps from one or more observations (Section 11.10).
- Inspecting your reprocessed maps (Section 11.11).
- Saving the reprocessed results (Section 11.14).

The SPIA enables an expert-level workflow with manual inspection of masks and flagging of artifacts, in the last flowchart in Figure 11.1:
Reprocessing with the SPIA

- Loading your observation into your HIPE session (Section 11.6).
- Loading SPIRE calibration information into HIPE (Section 11.8).
- Processing to Level 0.5 and detecting artifacts (Section 11.13.1).
- Correcting artifacts via manual inspection (Section 11.13.2).
- Repairing artifacts and reprocessing to Level 1 (Section 11.13.3).
- Making maps from one or more observations (Section 11.10).
- Inspecting calibrated timelines (Section 11.12).
- Inspecting your reprocessed maps (Section 11.11).
- Saving the reprocessed results (Section 11.14).

The pathways of data through the SPIA tasks are shown in Figure 11.2. The tasks are capable of handling all aspects needed to produce science-quality maps, from retrieval of observation and calibration data from the Herschel Science Archive, to storage and retrieval in a local pool on your computer, through the various stages of SPIRE processing, and output back to the local pool or to FITS files.

The SPIA plug-in is installed, and observations and the calibration tree are downloaded, the commands for reprocessing the data are very simple. For example:
This example demonstrates the simplicity in scripting enabled by the modular architecture of the SPIA tasks.

### 11.4. Locating the SPIA tasks within HIPE

The SPIA tasks are found in the Tasks view of HIPE. They may be located by selecting By Category → Spire in the Tasks view, then scrolling down through the alphabetical listing to the tasks starting with spia. However, the most practical way to locate relevant SPIA tasks is through the Applicable tab of the Tasks view. When an Observation Context is selected in the Variables view, all SPIA tasks that operate on an Observation Context will appear in the Applicable tab as shown in Figure 11.3.

Figure 11.3. Screenshots of the Tasks view in HIPE, showing (left) the SPIA tasks in the Applicable tab when an observation variable is selected, and (right) the tooltip for the spiaLevel1 task.

Hovering the cursor over a task name brings up a tooltip with a description of what the task is intended to accomplish. Double-clicking on a task name displays the task dialog in the Editor area.

Tooltips are provided for each parameter in the task dialog. Variables may be dragged and dropped onto respective parameter areas in the dialog, or filled in via the keyboard when an input box is displayed. Pressing Accept will execute the task, with the command-line equivalent appearing in the Console window.

#### Tip

The commands in the Console window may be copied from there, or from the History window, into a script for convenient reprocessing of observations. See the HIPE Owner’s Guide for the detailed procedure.

### 11.5. Retrieving observations from the Herschel Science Archive into local pools

The retrieval of observations from the Herschel Science Archive is discussed in detail in the Data I/O chapter of the Herschel Data Analysis Guide. The SPIA package includes specialized downloading task that retrieves from the HSA only the raw data and the minimum auxiliary and calibration data needed to process it. When run with the default parameters (as shown in Figure 11.4), spiaCopyHsa downloads only the raw (Level0) data, the offset history part of the calibration data, and stores it into the pool name provided. The higher levels of processing are easily reproduced in the course of running the other SPIA tasks. Of course, it is possible to include other levels (such as the maps in Level 2) by changing the parameters.
The `spiaCopyHsa` task will take upwards of several minutes to download the observation data from the HSA, depending on network speed. The equivalent command-line is:

```python
obs = spiaCopyHsa(ObsID=1342183479, Pool="spia_1342183479")
```

The data are stored in the pool name that you provide to the task. The output of the `spiaCopyHsa` task is the Observation Context, so that the observation is already loaded into HIPE, for the current session only.

### 11.6. Loading your observation into your HIPE session

Getting observations from your local disk into HIPE is discussed in detail in the Data I/O chapter of the Herschel Data Analysis Guide. The SPIA package includes a convenience task, `spiaLoadObs`, which will load the Observation Context into your HIPE session. Its functionality is similar to that of the `getObservation` function with which many users will already be familiar:

```python
# Equivalent methods for loading an observation into HIPE
obs = spiaLoadObs(1342183479, "spia_1342183479")
ob = getObservation(1342183479, poolName="spia_1342183479")
```

These methods require you to specify the observation ID and the pool name (and optionally the path, if the pool is not in the standard location). If you have many observations and pools on your local disk, it can be much easier to load observations using the Product Browser, discussed in the Data I/O chapter of the Herschel Data Analysis Guide.

**Tip**

Storing your reprocessed data with tags, as discussed in Section 11.14, allows you to differentiate between different versions when you use the Product Browser to retrieve the data.

### 11.7. Downloading the SPIRE calibration tree

The calibration context that is associated with the observation is, by default, the version used when the observation was processed by the Herschel Science Center, and oftentimes is out of date. The SPIA tasks intentionally separate the calibration context from the observation, to make explicit which calibration version is being used. In most cases the most recent calibration context is required. The CalTree entry is shown as “spire_cal_9_0” in these examples, but the SPIRE Instrument and Calibration page should always be checked for the latest version.
Included in HIPE is the `spireCal` task, described in Section 5.5. The SPIRE calibration context and its handling are described in detail in Chapter 5 and users are advised to review that section.

**Warning**

Retrieving the calibration context from the HSA can take several tens of minutes to complete. Fortunately this retrieval need be performed only once for every time a new calibration context is released (which is approximately on the same timescale as a new major HIPE release).

![Task dialog for the spiaLoadCal task](image)

**Figure 11.5.** Task dialog for the `spiaLoadCal` task, configured to read the "spire_cal_9_0" context from a local pool. Setting FromObservation to "Yes" would instead retrieve the calibration from the observation context.

### 11.8. Loading SPIRE calibration information into HIPE

To load the calibration context into the session, the task `spiaLoadCal` is provided. The dialog offers a choice between two sources for the calibration context: i) taking it from a pool in the local store of one’s computer (this is the default). ii) taking the one that came with the observation as part of the observation context. In the first case, hitting Accept with the defaults will load the calibration context into the variable “cal”. To take the calibration context instead from the observation, the observation context needs to be provided to the task as input. Pick the observation context (such as `obs`) from the Variables-view with the mouse pointer using the left mouse button and drag it over the round button that appears to the right of the variable name obs in the GUI, until a plus sign appears. Then drop the variable by releasing the left mouse button. The button will be green and to its right the variable name of the observation context will be shown. Then switch the drop-down menu on the right from its default “No” position to “Yes” so that the calibration context is actually taken from that observation.

Of course, in this second case of retrieving the calibration context provided with the observation context into a variable, this simple command-line will suffice:

```python
cal = obs.calibration
```

### 11.9. Reprocessing your data to calibrated timelines

The first genuine reprocessing piece is to convert raw (Level 0) or engineering-converted (Level0_5) data into calibrated Level 1 timelines, with signals in flux density units and with celestial coordinates attached. The use of scripts for this processing is covered for Large Map mode in Section 6.5, and for Small Map mode in Section 6.6. The SPIA package provides the `spiaLevel1` task for this purpose.

The `spiaLevel1` task contains the largest number of parameter entries in a task within the SPIA, and constitutes the heart of the data processing. Fortunately, the parameters are organized into tabs, with the most important parameters appearing on the first tab (shown in Figure 11.6).
The mere presence of a parameter does not indicate that it is essential to data reduction and needs to be changed. This task is meant to facilitate access to available parameters, but expert-level knowledge is needed to change many of them. Before experimenting with a parameter it is advisable to study the description of the respective pipeline module in the SPIRE Pipeline Specification Manual.

Hovering the mouse pointer over a parameter’s name will bring up specific help as a tool tip. An exact description of every parameter in the Level 1 panel is beyond the scope of this document. Some additional information can be obtained from the tool-tips that exist for each parameter. However, a few parameters that control the data flow within the SPIA scheme will be explained here.

To perform a successful reprocessing of SPIRE mapping data at Level 1, the task must be provided with an observation context and a calibration context. This is done by dragging and dropping the respective context from the Variables-view onto the respective round buttons in the task GUI as described before for the spiaLoadCal task. Both are mandatory input parameters, as indicated by the small asterisk by the parameter name. The next parameter is called CopyObs and can be set to Yes or No. It determines whether a new copy of an observation context should be created before the reprocessing. The copy includes the metadata, the calibration context, the auxiliary context, and the Level 0 context, and if not replaced by a reprocessed one, the Level 0.5 context.

The other parameters are set initially to the pipeline defaults and the reprocessing can be run immediately by pushing Apply on the lower right of the dialogue. A command line will appear in the Console-view indicating the building block that is being processed and warnings if modules are not executed because of corresponding selections in the spiaLevel1 GUI. For instance by default the sigma kappa deglitcher is not selected and a warning will appear if the task is run in its default configuration.
11.10. Making maps from one or more observations

Once the calibrated timelines of Level 1 processing are in hand, the task `spiaLevel2` is used to produce maps. The GUI of this task is launched in the same way as the others by double-clicking on the task `spiaLevel2` in the Task-view. As for the `spiaLevel1` task, the parameters are arranged into tabs, with the main tab shown in . The two mandatory inputs are like in the “`spiaLevel1`” task, the observation context and the calibration context that are provided as usual through the drag and drop procedure described earlier. In this case the observation context for input is named `obsOut`, which is the default name of the output product of the `spiaLevel1` task. The same selector CopyObs as in the previous task decides whether a new copied observation context should be used for the output or whether the task should just modify the input context. The default is set to No, as it is often useful to add the reprocessed Level 2 to an already existing observation context, which itself is a copy of the original that was produced by the `spiaLevel1` task. Independent of the actual choice, in the configuration shown, the default output of the task has the same default name and will be automatically changed to `obsOut_1`.

Figure 11.7. Task dialog for the main tab of the `spiaLevel2` task, showing default parameters.

On the Additional Observations tab, there are several other optional inputs “obs2”..“obs5”, for more observation contexts, of which the Level 1 timeline data can be combined into one resulting map. This option is intended for instance for parallel mode maps, where the orthogonal scan legs across the same region are in a different observation context. In such a case the two orthogonally scanned observations are processed separately to Level 1, and then both observation contexts are provided as input parameters to the Level 2 processing.

The Level 2 GUI gives the choice whether to use baseline removal, which is usually selected, unless the Level 1 context was pre-processed for that issue in a different way, perhaps by a custom script. If baseline removal is selected, additional four parameters provide an option to mask out a circular area around a position within the map that will not be used for determining the medians per scan. This can be useful if the map is dominated by one bright source that distorts the distribution of fluxes.

Back on the Main tab, the choice of mapmaker is between Naïve , MADmap , and Destriper . The pixel sizes in the map can be chosen on the last tab to be different from the defaults 6”, 10”, and 14” for PSW, PMW and PLW respectively. Also on the Main tab, the extended source gains are set to be used by default. By default, the maps will be displayed after they are complete, and the colour browse image will be made, which can be time-consuming.

**Tip**
For the beginning user, the parameter changes to consider include:
1. Using the extended gains, or not, via the extSrcGains parameter. For the best destriping and for extended sources it is recommended to use the default of Yes.

2. Making use of the Planck HFI zero-point correction, as detailed in the SPIRE Instrument and Calibration webpage.

3. When using the destriper, it is recommended to keep the MapMaker parameter at its default of naive. Although the destriper makes maps in the process of destriping, it is best to let the naive mapmaker run after destriping is finished, so that Level 2 deglitching masks are used properly.

4. For large maps, set useTempPool to Yes.

5. When processing many observations, or running via a batch script, it is recommended to turn the display of maps off via the displayMap parameter.

After hitting the Accept to run the task, three map viewer windows appear, one for each wavelength. These are just to show immediately the result of the processing and can be closed at any time, without any impact on the results. The actual results are being linked into an observation context according to the selection made about producing a copy first or using the original. While the first map displays are already available, the generation of a colour map for the browse image is still ongoing, provided this option was selected. Processing is only finished after the circling dot in the lower right of the HIPE panel comes to a stop. It should also be noted that if the reader followed up to this point in his/her own HIPE session, and opted to not create a copy of the observation context for Level 2 processing because already the context named obsOut is a copy of the original, there will still appear a variable named obsOut_1 in the Variables-view. This however is only a reference to the input observation context that can be deleted without deleting the result. It is good practise to do so through the right-click menu, to keep the number of variables down.

11.11. Inspecting your reprocessed maps

The popup windows that appear during Level 2 processing should already give a good idea about the result, as the Map Viewer itself has a wide range of functionalities going beyond the scope of this manual. Note that the same tools and more are accessible from the level2 icon in the Outline-view. Double click or right click and select the Context Viewer to bring up a viewer.

Only the eventually constructed map will reveal certain artifacts that have been missed and are difficult to detect in the time domain, for instance detector jumps and residual glitches. The main question then is to find out in which building block and in what detector to look for the event. The spiaPlotPosition task can help in this quest. Provided with an observation context that contains Levels 1 and 2, it plots the map of a specified detector array and overplots the traces of a selected detector within this map. By default the scan or building block numbers, starting with 0, are annotated. This annotation can be switched off if it becomes too confusing.

The default output plot is shown in Figure 11.8. By default the trace of the central detector of the PSW array E8 is plotted as a green line, which is also the origin of the detector angular offset calibration of all photometer detectors of SPIRE. Any other detector channel can be entered for plotting. The annotations are always placed near where the building block starts to scan.
11.12. Inspecting concatenated calibrated timelines

When finished producing a new Level 1, the results can be inspected by double-click on the level1 entry in the Outline-view. All Level 1 building blocks appear in a Photometer Scan Product-view. A single click on an icon for a building block to the left, produces the display of signal timeline data. More details of this inspection are described in Section 6.1.4.

To see longer term trends like temperature drifts, it is necessary to look at the timelines of all scans of an observation together. The spiaL1Concat task will produce a table of all scans concatenated into easily-viewed timelines. The task takes an observation context as input and concatenates the signals of selected detector channels into one TableDataset. Optionally the thermistor channels, dark channels, masks, and sky coordinates can be added as well. By default the two central detectors of PSW and PLW are selected and the detectors D7 and C5 of PMW, which has no central detector. Thermistors are added by default, but these selections can all be changed. There is also the option to add all channels into one large table.

The result appears by default as variable outSignal, which is a TableDataset object. This object can be displayed in the Dataset Viewer, Table Plotter or Over Plotter. A typical use would be to right click on outSignal in the Variables view and select “open with” – “OverPlotter”. Repeating this once more leaves two plot layers in the OverPlotter of which we configure one to show the PSWE8 channel (this is the default) and the other to show the PSWT1 thermistor channel. With some individual scaling along the y-axis (without touching the x-axis) the result looks like the left side of Figure 11.9. If we had reprocessed the data using the “extend” option in the processing to Level 1, the result would look more like the right side of Figure 11.9. In both diagrams, we see the temperature corrected detector signal of PSWE8 as black symbols, crossing the target source eight times, which happens at the eight distinct excursions from the thick baseline. The orange line represents one of the two thermistor channels of the PSW array that was used to correct the signal for temperature drift.
11.13. Correcting artifacts via manual inspection

Two additional tasks have been provided named spiaLevel05 and spiaLevel1Repair as an alternative processing route to the spiaLevel1 task, which represents the standard pipeline approach. The issue is that the deglitcher tasks in the standard pipeline do not only find, but also “repair” glitches in the processing towards Level 1. This makes manual inspection and correction of glitches at Level 1 very difficult as the data has now been altered, obscuring why the deglitcher found them in the first place, and for the same reason also preventing an easy way back in case of an erroneous detection. Simply removing the masks with the spireMaskEditor will not bring an accidentally removed source back at Level 1.

SPIA offers an alternative workflow that sets the masks already at Level 0.5 with the spiaLevel05 task. All deglitches are available, but the “repair” sections are turned off by default. The voltages are not changed and a full inspection of the masks set by the deglitches and the jump detector in conjunction with the data is now possible using the spireMaskEditor. Masks can even be changed after the fact at Level 0.5 to correct for imperfect deglitching or jump detection. After this the task spiaLevel1Repair is run that only runs the “repair” section of the Sigma-Kappa deglitcher to remove/interpolate over all masked out parts of the signal timeline without trying to detect new glitches. After this the standard pipeline processing is continued up to Level 1 with the same opportunities to change parameters as in the spiaLevel1 task.

It should be clear that this alternate route can not exactly replicate standard pipeline results, since the repairing of glitches right after detection influences the detection of glitches of a subsequent deglitcher, which is different in the alternate approach. The following sections give a quick run-through of the alternative reduction steps to Level 1.

Note that in addition to the SPIA routines for correcting artefacts, user can also use the Bolometer Finder, described in Section 8.3 in order to inspect maps and locate artefacts for correction.

11.13.1. Processing to Level0.5 and detecting artifacts

The default GUI of spiaLevel05 (see Figure 11.10) looks identical to that of “spiaLevel1” except that is ends after the last parameter for the Sigma-Kappa deglitcher. By default all repair mechanisms of the deglitches are switched off, although turning them back on is possible. Observation context and calibration context are dragged onto the respective obs and cal dots, and pressing Accept starts the processing. Note that also here there is a choice between full engineering reprocessing of Level 0.5, or accepting the Level 0.5 data that came with the observation, and performing glitch and jump detection only, resulting in substantial reduction of run-time.
Figure 11.10. Task dialogue for spiaLevel05. This task runs the delitchers to detect glitches, but repair is deferred to a later stage, after manual inspection.

The result will appear as usual under the name ObsOut or an alternative if that name is already taken. This is regardless whether the copy option was chosen or not (CopyObs parameter). If the observation context was not copied, the ObsOut variable will still show in the Variable-view, but in this case it can be safely deleted there as it is just a reference.

11.13.2. Correcting artifacts via manual inspection

The inspection and flagging of the Level 0.5 masks can be accomplished with the spireMaskEditor viewer. The main dialogue of the Mask Editor is shown in Figure 11.11. All mask bits are shown in a table. A green square indicates that a mask bit is set for the channel and readout selected at the bottom of the dialogue. These bits can be toggled with the mouse pointer. The top panel allows to select the context (Level 0.5 or 1), the building block, and it shows the observation ID and the total number of readouts. The red radio button selects mask bits for display in an XY-plot with the Show/Edit button on the right.

Figure 11.11. Dialogue for the SPIRE Mask Editor viewer. Each mask bit is listed in a table and can be edited here, or within a signal/voltage plot.

An example plot is given in Figure 11.12. These plot windows are effectively Table Plotter windows and allow editing of the mask bits via the Hide/Unhide buttons. Before leaving the Mask Editor, the
system verifies whether to keep the applied changes. For instance this way a wrongly masked source could be unmasked manually at Level 0.5. Also thermistor signal jumps missed or falsely detected could be corrected at that level.

![Figure 11.12. The SPIRE Mask Editor voltage/signal display. For Level 0.5 the voltage is plotted versus time (top) and the readouts where the selected mask bit is set, are indicated by red crosses. The bottom plot shows the equivalent display for Level 1. Note that the Level 0.5 signal still contains the glitch that was maked by the deglitcher, while at Level 1 it is “repaired” but still masked.](image)

**11.13.3. Repairing artifacts and reprocessing to Level 1**

After running spiaLevel0_5 and inspecting/correcting Level 0.5, the masked glitches need to be repaired and the remaining processing steps up to Level 1 need to be executed. This is done using the task spiaLevel1Repair. A picture of the respective task dialogue is shown in Figure 11.13.
11.14. Saving the reprocessed results

The new observation contexts that are being produced during such an interactive analysis session are still residing in memory, at least in part, while the data content is tucked away in a temporary storage pool on disk that is destroyed as soon as the HIPE session terminates. To keep at least the important results, the task `spiaSaveObs` provides two specialized features beyond those discussed in the [Herschel Data Analysis Guide](https):

1. The `outputSelection` parameter will save the full Observation Context and all its associated data by default. It also allows only selected portions of the observation to be saved. For example, you may wish to save only the Level 2 data (maps), or the Level 1 (calibrated timelines) and Level 2 (maps) portions that you have reprocessed.

2. The `nameTag` parameter is a string that you can use to identify these results. The tag is stored with your data, and can be accessed by the Product Browser at a later date. The tag feature enables you to save multiple versions of reprocessing to a single pool, with a short note to allow you to differentiate the different versions.

The reprocessed maps may also be saved directly to FITS files. The general case of saving Herschel products to FITS files is covered in the [Herschel Data Analysis Guide](https). The `spiaSaveMaps` task accepts as input an observation context containing a Level 2 context. It saves three FITS files, one for each detector array, each containing three extensions representing flux map, error map, and coverage map. The filenames are generated from a user supplied suffix, the detector array name, and the observer.
vation identifier. The output path is set by default to be the location from where HIPE was started, and can be changed as needed. A switch that enables file overwrite warning completes this GUI.

![spiaSaveMaps2Fits task](Image)

**Figure 11.15.** Task dialog for the spiaSaveMaps2Fits task. Taking an Observation Context as input, the task outputs the three maps for the PSW, PMW and PLW arrays with a single command.

### 11.15. Worked Examples

The scripts in this section show how to configure the spiaLevel1 and spiaLevel2 tasks for typical observations. The examples assume that the spire_cal_14_0 calibration tree is saved to a local pool of the same name.

#### 11.15.1. Map-making with turnaround data and destriping

This example shows how to include extra data at the map edges along with the destriper, and implicitly, the extended gains.

```python
# Load the calibration tree from local storage
cal = spireCal()

# Retrieve an observation from the archive
obs = getObservation(1342211401, useHsa=True)

# Process to Level 1 including the turnaround data
obsOut = spiaLevel1(obs=obs, cal=cal, extend='Yes')
```

#### 11.15.2. Destriping two large maps

This example shows how to process two large parallel maps making use of temporary storage to lower demands on computer memory.

```python
# Load the calibration tree from local storage
cal = spireCal()

# Retrieve the two parallel observations from the archive
obs1 = getObservation(1342183046, useHsa=True)
obs2 = getObservation(1342183047, useHsa=True)

# Make maps with destripping with default values
obsOut = spiaLevel2(obs=obs1, cal=cal, obs2=obs2, useTempPool='Yes')
```
Chapter 12. References

12.1. SPIRE DRG References

The SPIRE Data Reduction Guide refers to the following scientific publications:


[Valtchanov et al. 2014] *Relative pointing offset analysis of calibration targets with repeated observations with Herschel-SPIRE Fourier-transform spectrometer* I. Valtchanov, R. Hopwood, E. Polehampton et al. 2014, Experimental Astronomy, 37, 207