SPIRE Data Users Manual

version 1.0.dev, Document Number: SPIRE-RAL-DOC 00XXXX
09 December 2009

SPIRE Data Users Manual

Table of Contents

Preface

1. Versioning

On the front page of this manual is a version number made of three digits. The first two digits follow a traditional versioning system (0.1, 0.2, ...), and the changes introduced with each version are detailed below. The third digit is the SPIRE build number to which each edition of the manual is associated. Also shown on the front page is the date of publication of the manual.

1.1. Changelog

The following was changed for v0.1

• First version of the SDUM manual.

Chapter 1. Introduction

1.1. Scope of this Data User's Manual

The purpose of this document is to provide a comprehensice reference for all SPIRE users in terms of the data structure users will encounter for on inspection of the different types of SPIRE observations, but also as a guide on how to reprocess the data and inspect the products through the full SPIRE pipeline. This document superceeds the SPIRE pipeline reduction formerly included in the HOWTOs document, but has been expaned to include all modes and insights on the data struture and types.

The data structure and reprocessing guide examples contained within the SPIRE Data Users Manual are based upon the HIPE 2.0 release - views may differ and examples may not work on previous and subsequent releases of HIPE. If you are using a release of HIPE other than the 2.0 build, please consult the relevant version of the SPIRE Data Users Manual.

For more information on obtaining HIPE and on how to install it, getting started with it, please go to the HIPE Quick Start Guide and the HIPE Owners Guide for a more more indepth overview of getting started with the HIPE environment.

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 observaing 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 is scanned building up a map, scan line by scan line. Scan lines can be orthonally cross-linked to produce high quality maps.

- **Small Map Mode (64-point Jiggle, POF3):** Used for observations of large fields (>4x4 arcmins). The telescope stares at a target and the detector arrays are jiggled, using a Beam Steering Mirror (BSM), over the target area to build up a fully sampled map using a 64-point pattern. The background is removed by chopping with the BSM and Nodding with the telescope.

- **Point Source Mode (7-point Jiggle, POF2):** Used for observations of point sources. The telescope stares at a target and the detector arrays are jiggled, using 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 created with both SPIRE and PACS in parallel. These are essentially equivalent to Large Map observations.

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

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

1.3. Structure of this document

Astronomer users will receive data that has already been processed through the standard pipelines to several Levels. The processing levels of the SPIRE pipeline and user deliverables are outlined below in Figure 1.1.

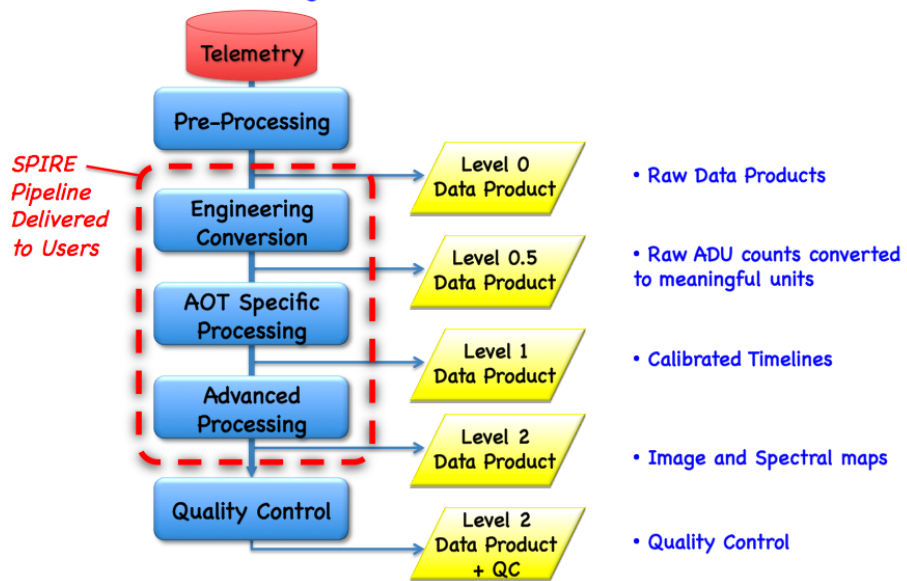**Figure 1.1. The processing levels of the SPIRE pipeline and user deliverables.**

This document is divided into two broad topics. An introduction to the data structure as received from the Herschel Science Archive (HSA) is described in Chapter 2 which includes all relevant observation modes and processing Levels. The pipelines themselves and details on reprocessing your observations are covered in Chapter 3.

Chapter 2. Looking at your data

2.1. SPIRE Observation Context Data Structure

2.1.1. 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 Owners 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 its 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 to previous astronomical datasets you have encountered.

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

- ....

Products can contain the following (pictorially visualized in Figure 2.1);

- Meta Data

- One or more Datasets

- Processing History

Datasets can be;

- Array Tables

- Image arrays

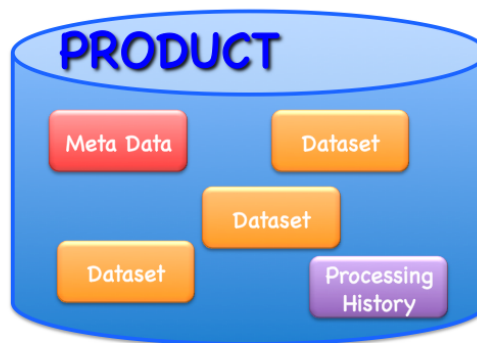- Composite (nested) Tables

- ....

Figure 2.1. General structure of a SPIRE data Product

SPIRE (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 HSPOT when you submitted the proposal, and the pipeline calibration tables). 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**" 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 by editing the HCSS user.props file. The structure of the Local Store is visualized in Figure 2.2
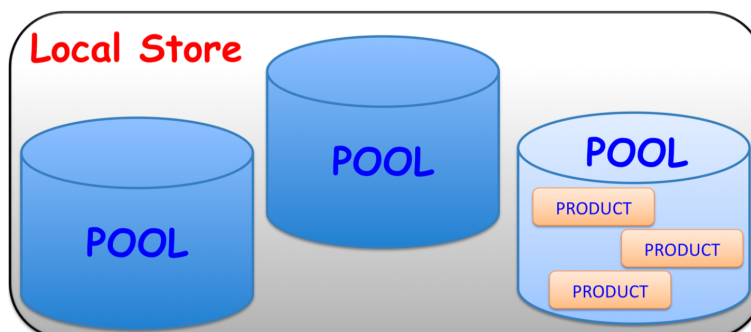

Figure 2.2. General structure of the Local Store

2.1.2. 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 2.3. 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 Obseravtional Context is shown in Figure 2.4 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 Herschal Science Archive (HSA).
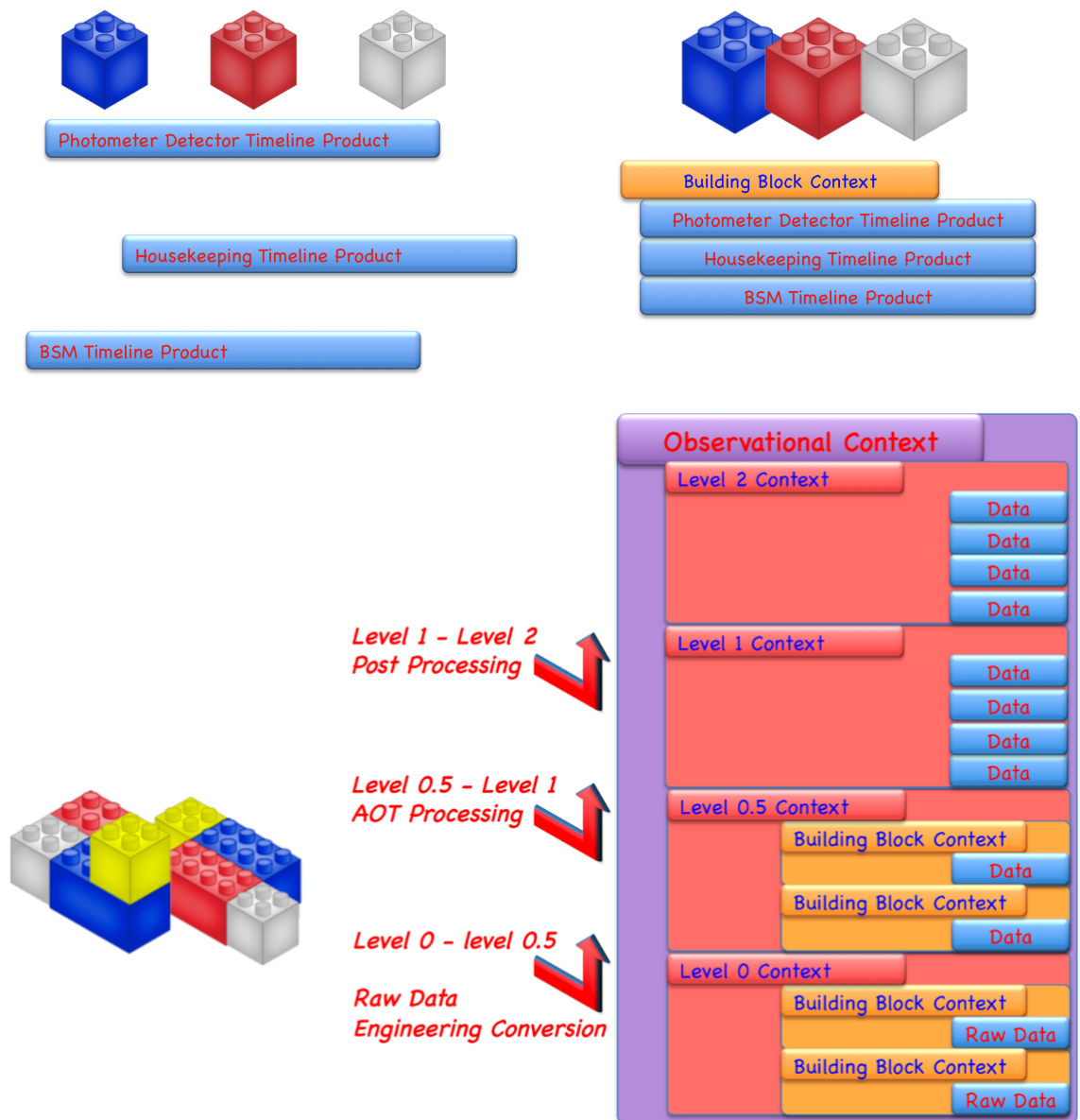

Figure 2.3. The Context structure within HCSS. The smallest "piece" of SPIRE observational data are Building Blocks. Building Blocks and other Products are grouped into a context. All the data within an entire SPIRE observation are linked by an Observation Context.
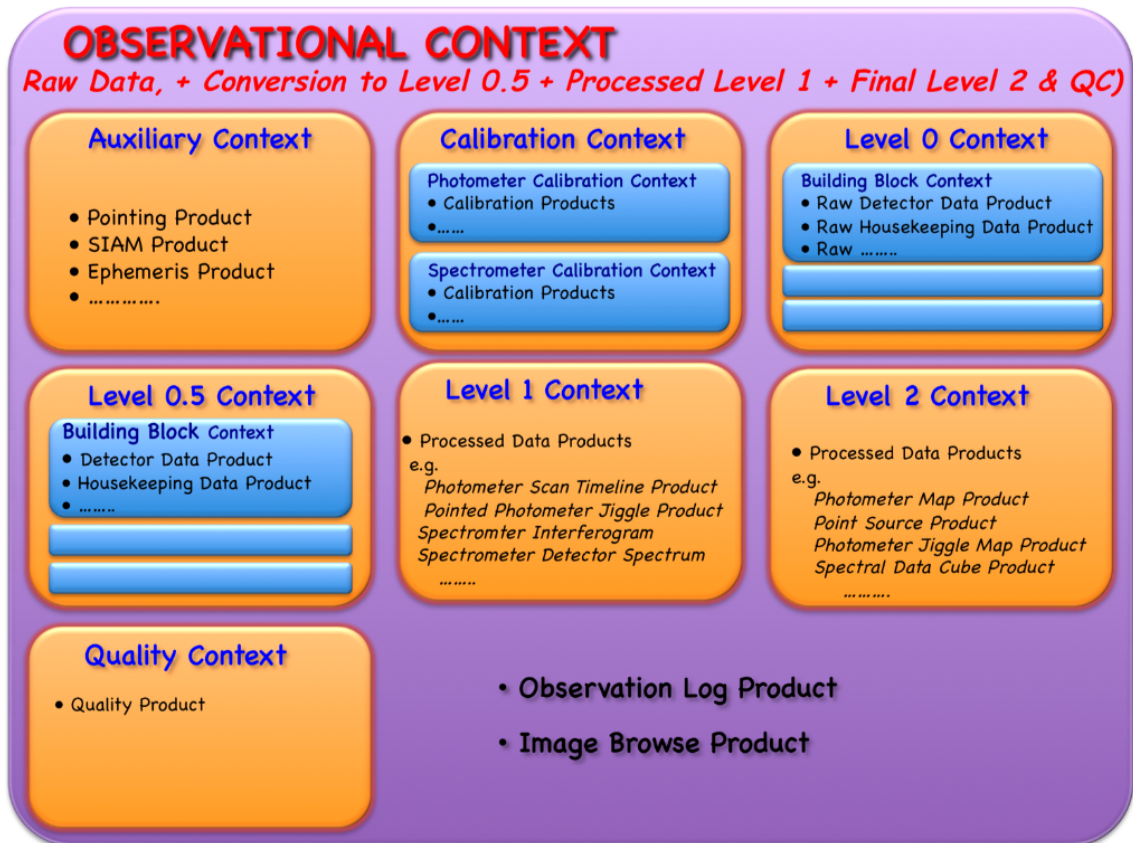
Figure 2.4. The complete Observation Context of a SPIRE observation

2.1.3. 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 has already been downloaded from the archive and has already been stored in a pool named `GalaxyScanMap` in the Local Store. We therefore have to load this pool into the HIPE environment 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 a the few lines of code shown below;

```
Pool  = 'GalaxyScanMap'                       # Select the pool name
storage=ProductStorage(Pool)                  # Register the pool
queryResults = storage.select(Query("type=='OBS'"))  # Query the pool
MyObsContext = queryResults[0].product        # Extract the  Context
```

The first line of code selects the desired Pool from our Local Store on disk. This Pool is read in to a storage area in memory (referred to as *Registering the Pool*") which we have decided to call `storage`. Once the Pool has been registered, it can then be *queried* for the observation context by searching the storage for the Product Type `OBS`. Finally, the Observation Context Product is stored in a variable we choose to call `MyObsContext`. After running the above lines we see five new entries `Variables` pane of HIPE shown in Figure 2.5. These variables have already been described above (Note: the `p` is simply a place holder). Double clicking on the `obsContext` in the variable list brings up the *Observation Context* observation in a new window as also shown in Figure 2.5. 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 2.1. The Observation Context `Data` pane contains pointers to all other contexts and data products contained in the Observation Pool. The `Data` pane contains many entries, listed below and in Figure 2.6 (See also Figure 2.4);

- `level 0`: The Level 0 context containing links to the Level 0 raw dData 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.

- `calibration`: The Calibration context pointing to all calibration products required for the processing of SPIRE data.

- `auxiliary`: The context pointing to all .

- `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`;

```
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 next section, 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 2.1. Description of Meta Data in the SPIRE Observation Context

| Meta Data	Description
odNumber	The Observational Day when the observation was made
obsid	The unique Observation ID (in decimal)
startDate	The start date of the observation in TAI, Zulu Time
endDate	The end date of the observation
creationDate	The creation date of this Product
creator	How the product was created (e.g. Standard Product Generation (SPG) version)
modelName	Whether the data is from Flight or Flight Spare, etc
obsState	How far has the observation been processed by the pipeline (Level 0, 0.5, 1 or 2)
type	The Product Type (OBS = Observation Context)
instMode	The instrument mode (The AOTs defined internally as POF5 for Large Map Mode)
instrument	The instrument name, in this case SPIRE
cusMode	How the AOT is referred to in the observaion logs and scheduling (SpirePhotoLargeScan)
description	The Product name
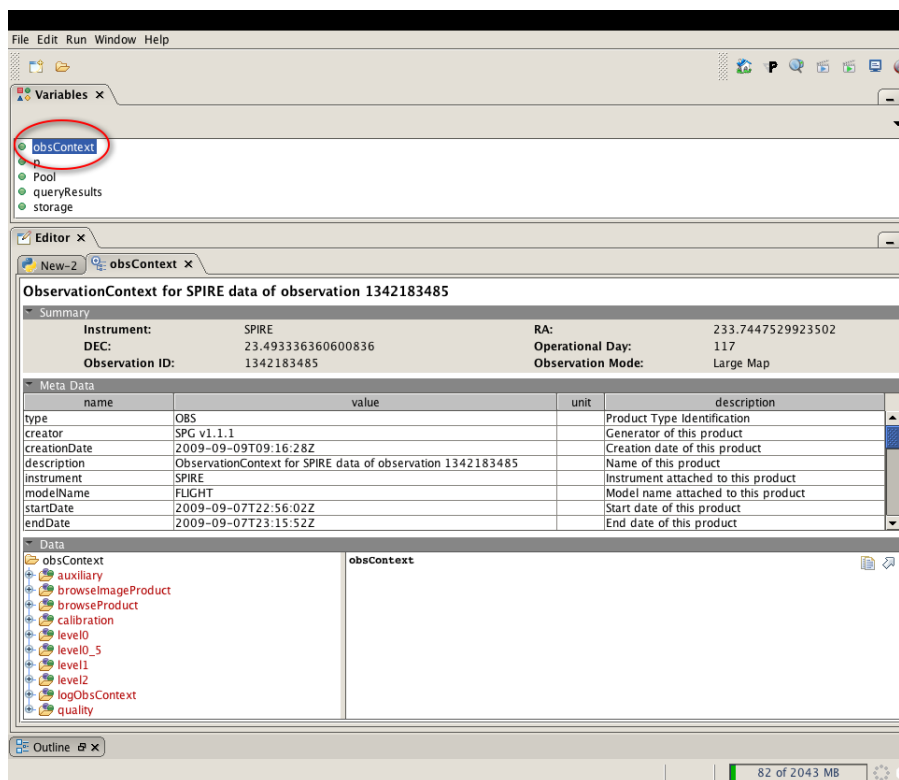
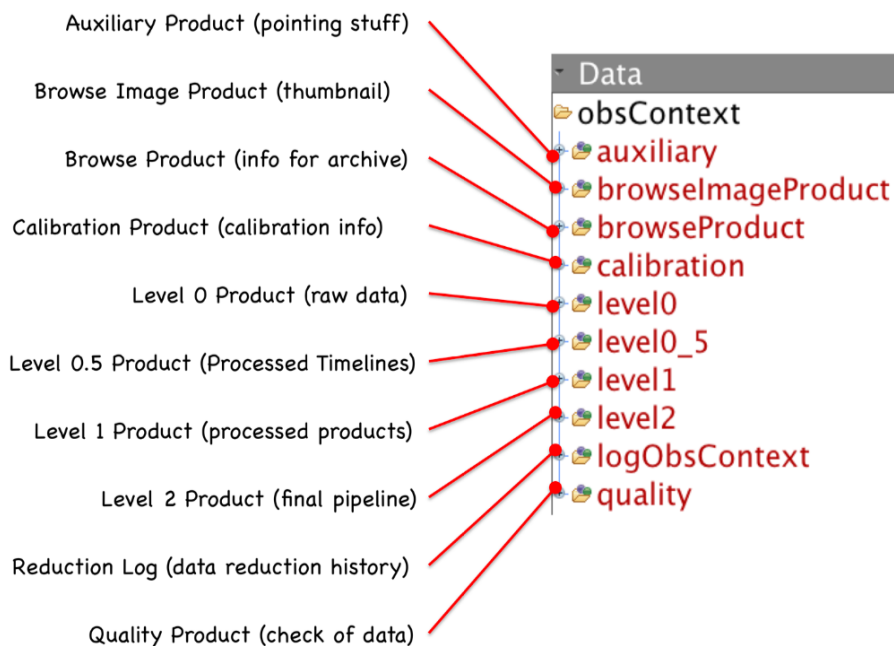**Figure 2.5. The Observation Context within HIPE**

Figure 2.6. Inside the Observation Context within HIPE.

2.2. SPIRE Small Map and Point Source Mode Data Structure

2.2.1. The Point Source Observation Mode

All the information for a given SPIRE observation is contained with the Observation Context (described in Section 2.1). In this section we shall see how to examine the data for a SPIRE **Point Source** observation. A point source observation carries out a staring observation of a point source. In order to recover the source successfully 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 arc sec 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 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 involves makes 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 2.1. The Observation Pool can be loaded into HIPE using the following 4 lines of Jython Code (where the **Pool** is whatever name you called your Pool for this observation in your Local Store on disk;

```
Pool  = 'OD117-7ptNGC5315-0x50001832'          # Select the pool name
storage=ProductStorage(Pool)                   # Register the pool
queryResults = storage.select(Query("type=='OBS'"))  # Query the pool
MyObsContext = queryResults[0].product         # Extract the  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. Running the above script, reads the Observation Context into memory into the variable **MyObsContext** which appears in the **Variables** pane of HIPE (See Figure 2.7). 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 2.1 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 acccesed by clicking on the **level2** folder as shown in Figure 2.8, 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 2.8. 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 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 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)

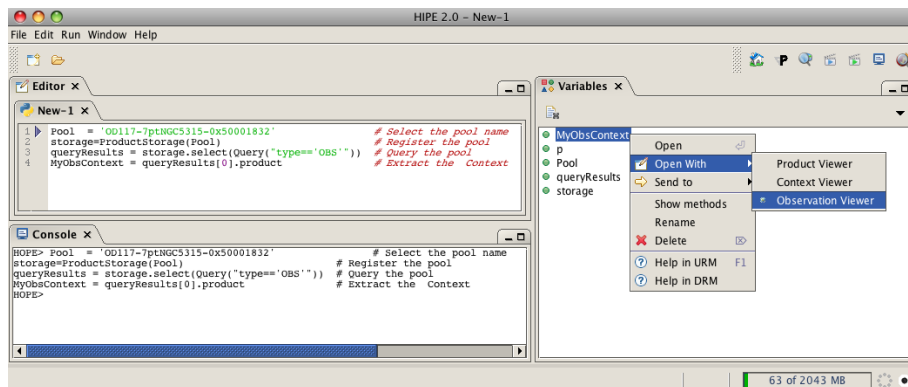- **Error:** A column listing the error on the fitted signalfor each array

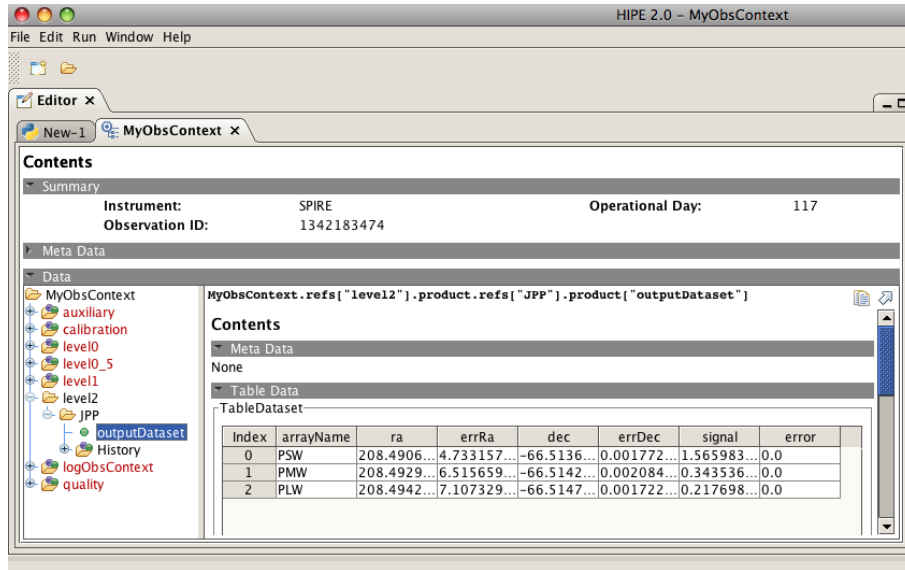**Figure 2.7. Loading and viewing the Observation Context for the Photometer Point Source Observation.**

Figure 2.8. Accessing the final Level 2 Product Jiggled Photometer Product

2.2.2. Reading the JPP into memory and saving it as a FITS file and reading it in again

It is possible that me 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 admittadly long-winded command from the command line;

```
# read entire Product
myJPP=MyObsContext.refs["level2"].product.refs["JPP"].product
#
# 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;

```
FitsArchive().save('mypath/myJPP.fits', myJPP)
```

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 2.9 where we can type in our desired filename and path. Click on `Accept` at the bottom of the panel to save the FITS file.
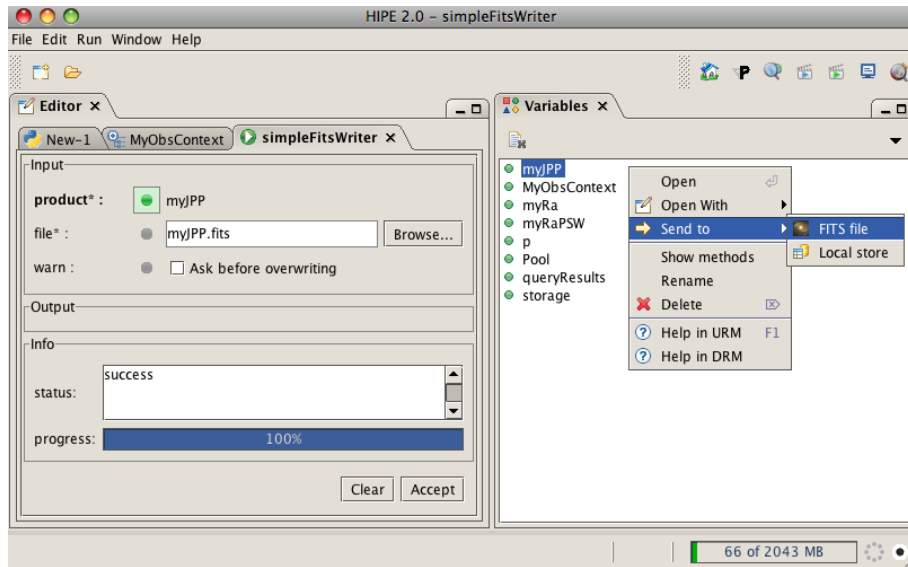
Figure 2.9. Exporting the JPP as a FITS file

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;

```
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.

> **Note**
>
> 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 poolname is "OD117-7ptNGC5315-0x50001832"). The JPP will have the `hspirephotometer.......jpp.fits`

2.2.3. 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 2.10 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-nodded.

Each Averaged Pointed Photometer Product contains 7 individual Table Datasets (and a Product containing the processing history) as shown in Figure 2.10 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 the **SPIRE Pipeline User Guide** document

The APPP 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 2.11 where we have selected to plot the Jiggle ID against the Signal from the PSW E10 bolometer for the APPP.

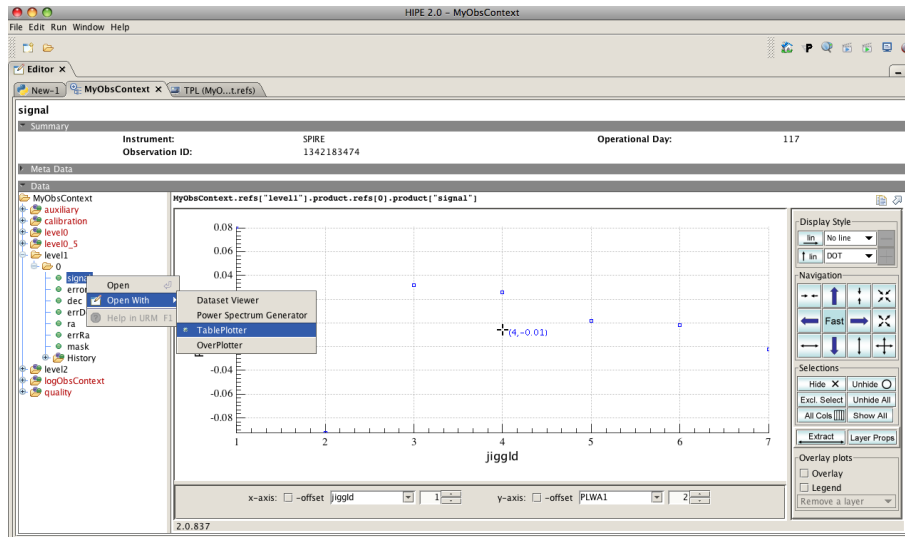**Figure 2.10. Viewing the Level 1 Averaged Pointed Photometer Product**

Figure 2.11. Plotting Level 1 APPP Data Product

_____

2.3. SPIRE Large Map and Parallel Mode Data Structure

2.3.1. A first look at your image maps

All the information for a given SPIRE observation is contained with the Observation Context (described in Section 2.1). 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 orthoganal direction. The entire process was then repeated (i.e. this observation has 2 repetitions) giving in total 6 scans in each orthoganal 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 2.1. The Observation Pool can be loaded into HIPE using the following 4 lines of Jython Code (where the **Pool** is whatever name you called your Pool for this observation in your Local Store on disk;

```
Pool  = 'OD117-ScanNGC5315-0x50001833'        # Select the pool name
storage=ProductStorage(Pool)                  # Register the pool
queryResults = storage.select(Query("type=='OBS'"))  # Query the pool
MyObsContext = queryResults[0].product        # Extract the  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. Running the above script, reads the Observation Context into memory into the variable **MyObsContext** which appears in the **Variables** pane of HIPE (See Figure 2.12). 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 2.1 and he 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 acccesed by clicking on the **level2** folder as shown in Figure 2.13, 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, error and coverage maps for each array and these are revelaed by *clicking* on the + sign next to the array folder.

The image map can be viewed by clicking on the appropriate array folder (PSW, PMW, PLW) 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 2.14. This action opens the image in the `Image Viewer` where the image can be panned, magnified etc. Colours, cut-levels, annotation options can be accessed by *right-clicking* anywhere on the image. 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 ArrayDatasets` 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 2.15.

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:

```
MyMapProduct=MyObsContext.refs["level2"].product.refs["PSW"].product
# 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. 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.
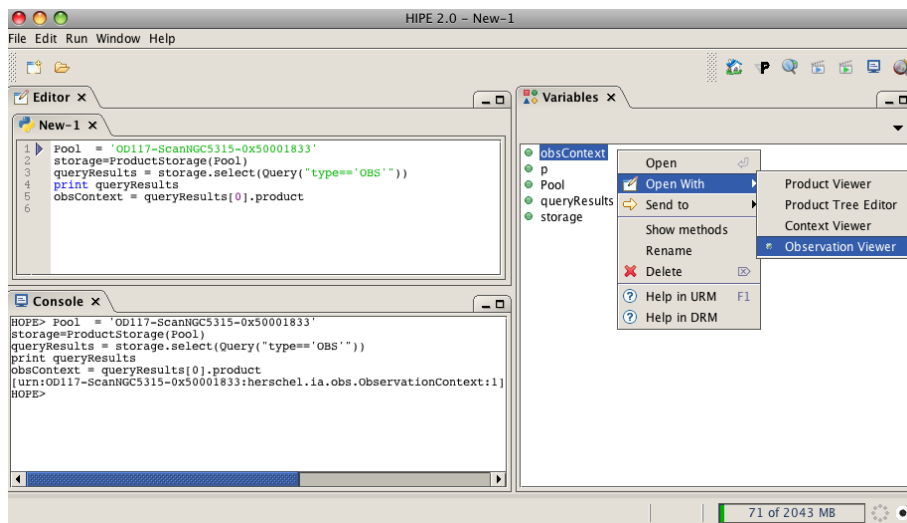

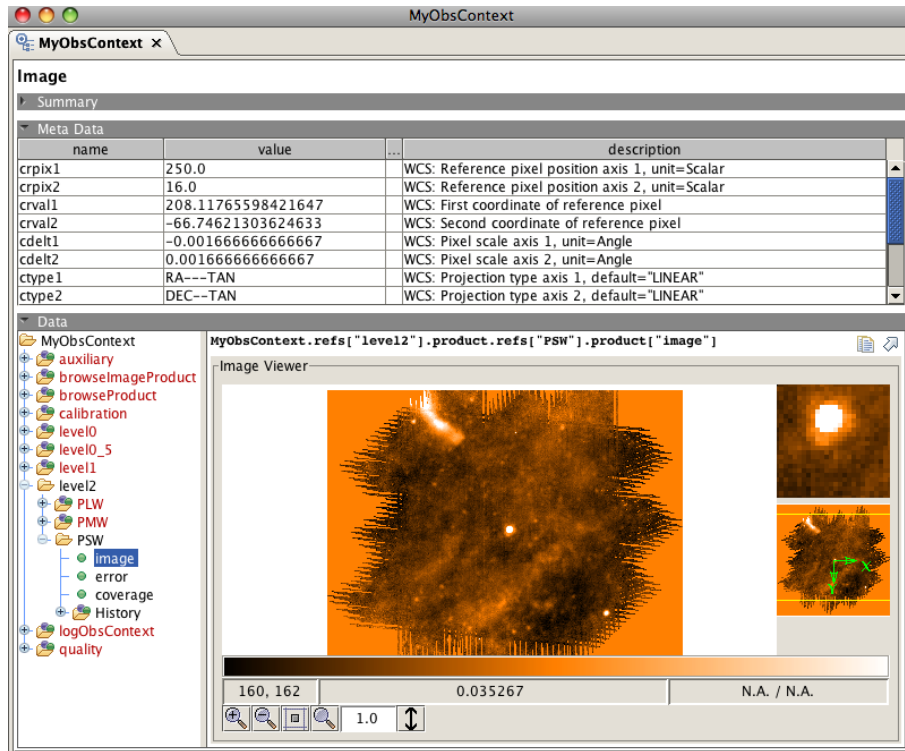**Figure 2.12. Loading and viewing the Observation Context for the Large Map Observation.**

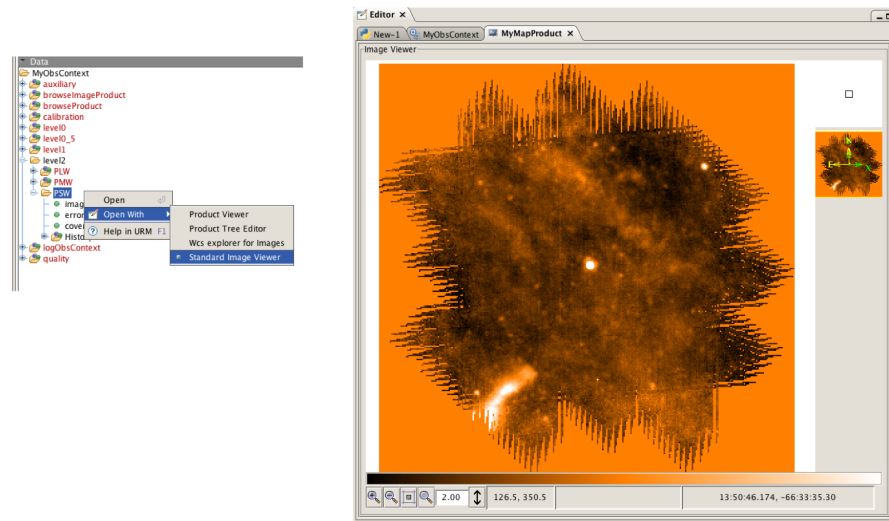**Figure 2.13. Accessing the final Level 2 Product maps**

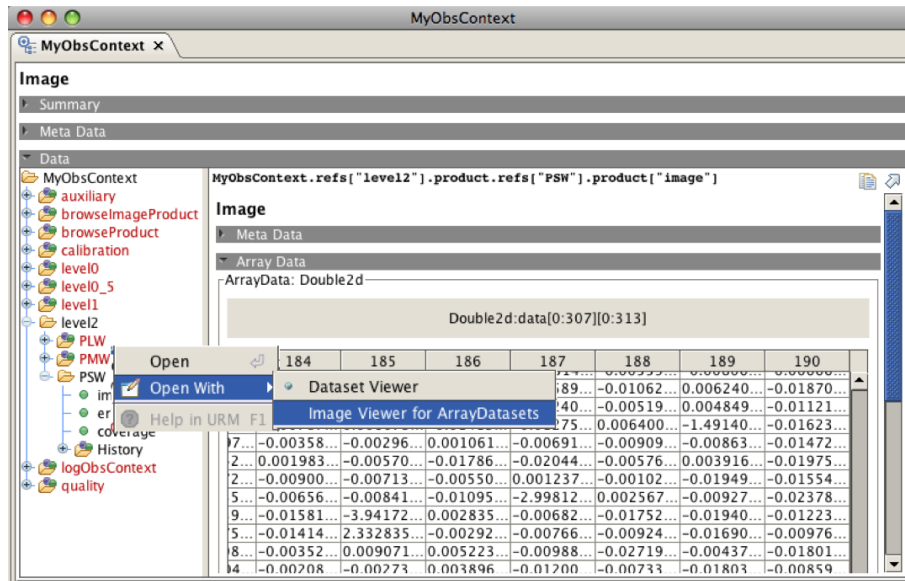**Figure 2.14. Viewing the Level 2 Image Maps**

Figure 2.15. Viewing the Level 2 Image Array Datasets

2.3.2. Saving a map as a FITS file and reading it in again

It is possible that me 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 and selecting `Send To-FITS file` from the drop down menu. This will open the FITS writer panel as shown in Figure 2.16 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, error and coverage maps that can then be read into DS9 as a data cube and viewed. The same effect can be acheived on the command line by;

```
FitsArchive().save('mypath/myMap.fits', MyMapProduct)
```

which again saves the products as a multi-extension fits file containing the image, error and coverage maps.
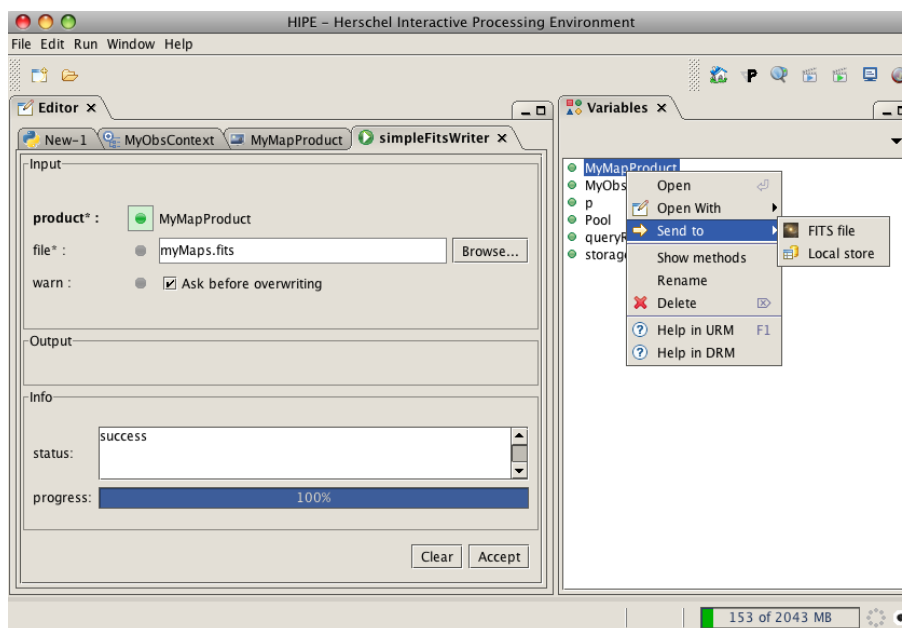
Figure 2.16. Exporting Image Maps as FITS files

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;

```
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.

> **Note**
>
> 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 pool-name is "OD117-ScanNGC5315-0x50001833"). The Photometer Map Products having the form `hspireplw..........pmp.fits`

2.3.3. Looking at the Level 1 Timeline Data

The image maps have been created from the individual timelines of detectors as they were scanned accross 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**. In Figure 2.17 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 orthoganal 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)

Each Photometer Scan Product contains 5 individual Table Datasets (and a Product containing the processing history) as shown in Figure 2.17 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 the **SPIRE Pipeline User Guide** document

- **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 2.18 where we have selected to plot the Sample Time against the Signal from the PSW D16 bolometer for this particular scan line.
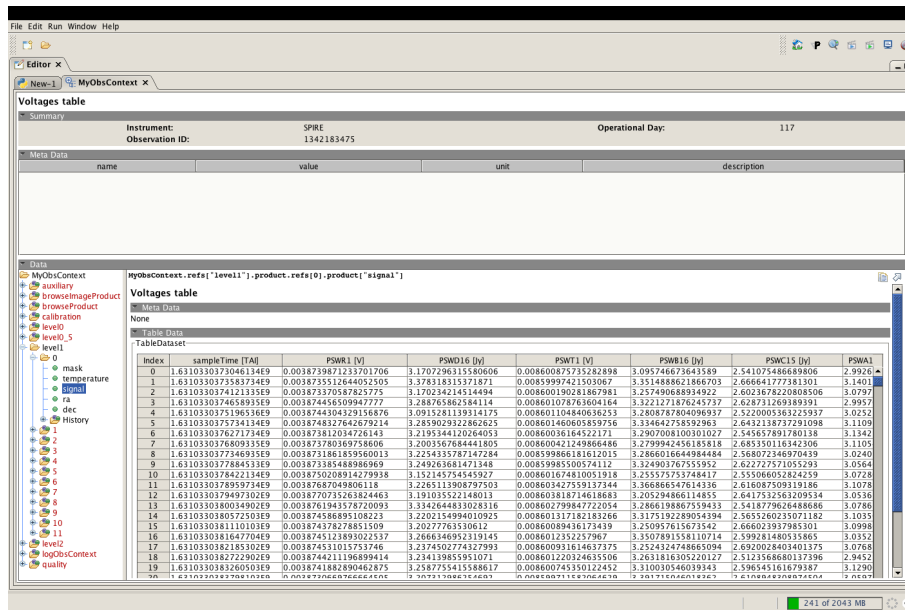

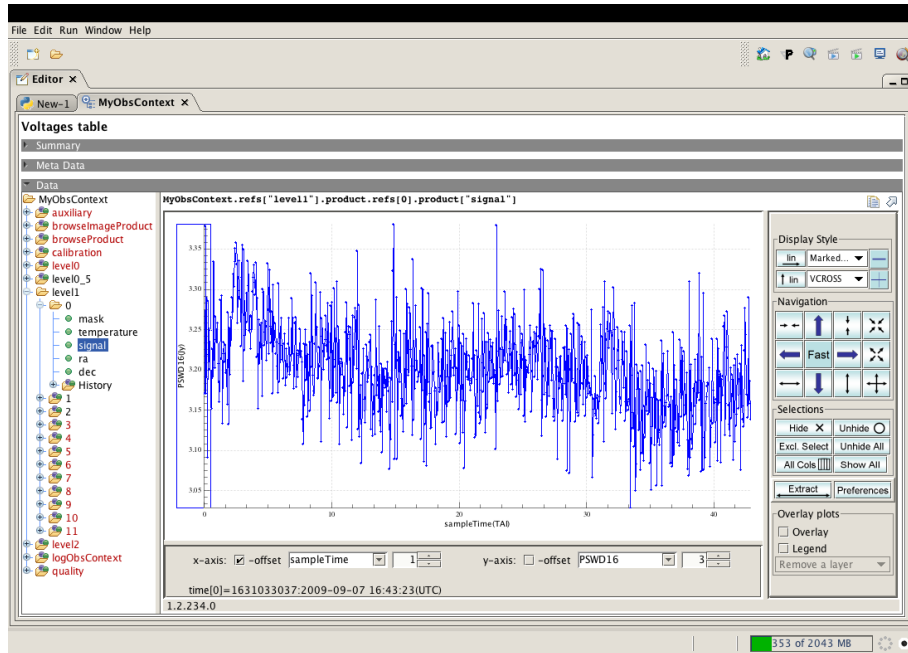**Figure 2.17. Viewing the Level 1 Photometer Scan Products**

Figure 2.18. Plotting Level 1 Photometer Scan Product Timeline Data

Individual Table Data Sets can also be extracted from the Observational Context using the alternative command line script. Using Figure 2.18 as a guide we can see the following;

```
# 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
print PSWD16ScanLine1
```

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 extarcted as normal arrays of numbers. The final list of variables in the HIPE Variable Pane is shown in Figure 2.19.
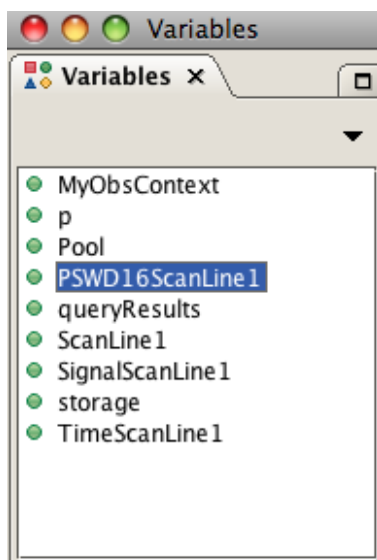
Figure 2.19. Plotting Level 1 Photometer Scan Product Timeline Data variable list

2.3.4. Looking at the Level 0.5 Timeline Data

These timeline data has 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 2.20 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. 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 2.20 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 1-30 and scrolling down the Meta data window pane to the BBtypeName entry. The individual blocks are described below in Table 2.2;

Table 2.2. Description of the Building Blocks in a Large Map Level 0.5 Context

| BB number	BB Type	BB Hex prefix	Description
0	SpireBbObsConfig	0xAF01	Initial configuration
1	SpireBbPhotSerendipity	0xA104	Slew to target
2	SpireBbPOF5Config	0xA050	AOT configuration
3	SpireBbPOF5Init	0xA051	Initialize the AOT
4	SpireBbPcalFlash	0xA801	Photometer Calibration Lamp Flash
5	SpireBbScanLine	0xA103	A large map scan line
6	SpireBbMove	0xAF00	Scan Line turnaround movement
7	SpireBbScanLine	0xA103	A large map scan line
8	SpireBbMove	0xAF00	Scan Line turnaround movement
..	SpireBbScanLine	0xA103	A large map scan line
..	SpireBbMove	0xAF00	Scan Line turnaround movement
..	...	...	....
27	SpireBbScanLine	0xA103	A large map scan line
28	SpireBbMove	0xAF00	Scan Line turnaround movement
29	SpireBbPcalFlash	0xA801	Photometer Calibration Lamp Flash
29	SpireBbPOF5End	0xA052	End of AOT
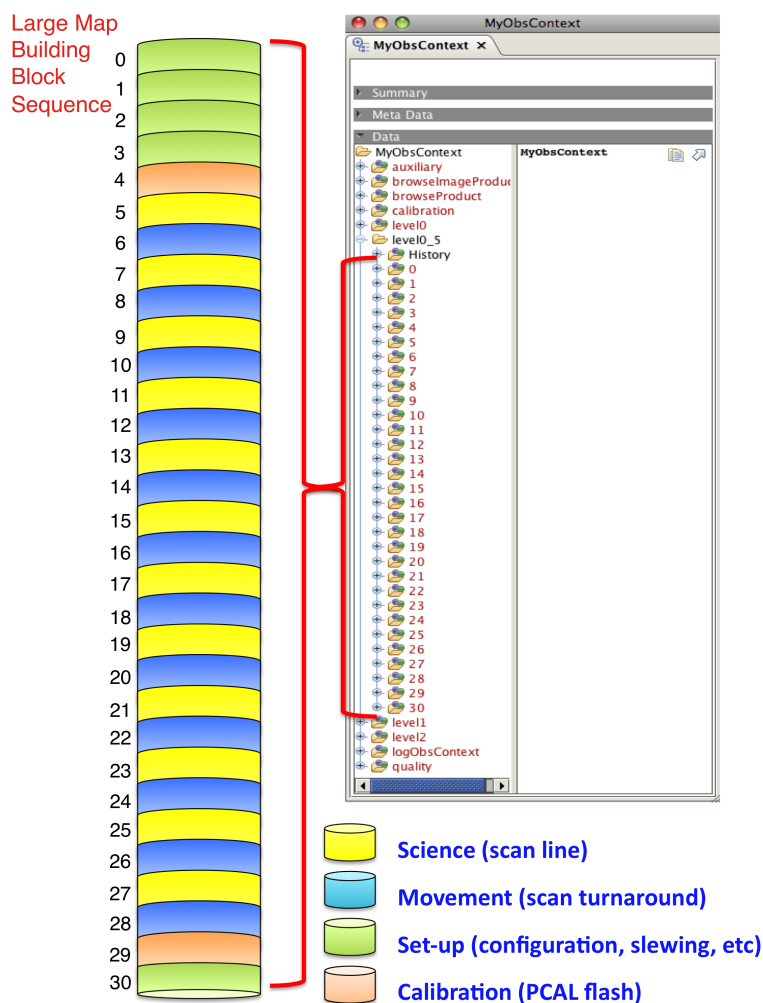
Figure 2.20. Anatomy of Level 0.5 Building Block structure for a Large Map observation

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 2.21 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 ancronyms 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, containg data arrays and a Mask table containing flag information. The Level 0.5 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 the **SPIRE Pipeline User Guide** document

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

In Figure 2.21 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-Dataset Viewer as shown in Figure 2.22. In the plot window the bolometer signal to plot can be selected from the Y-axis menu and many bolometers can be overlayed by ticking the overlay box (both circled in the plot window).
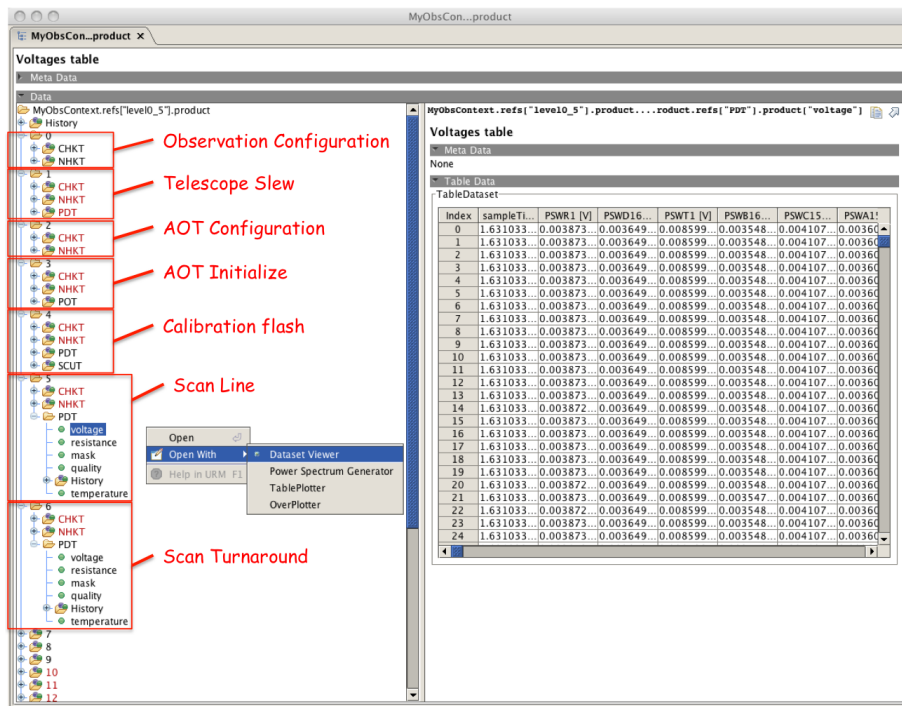

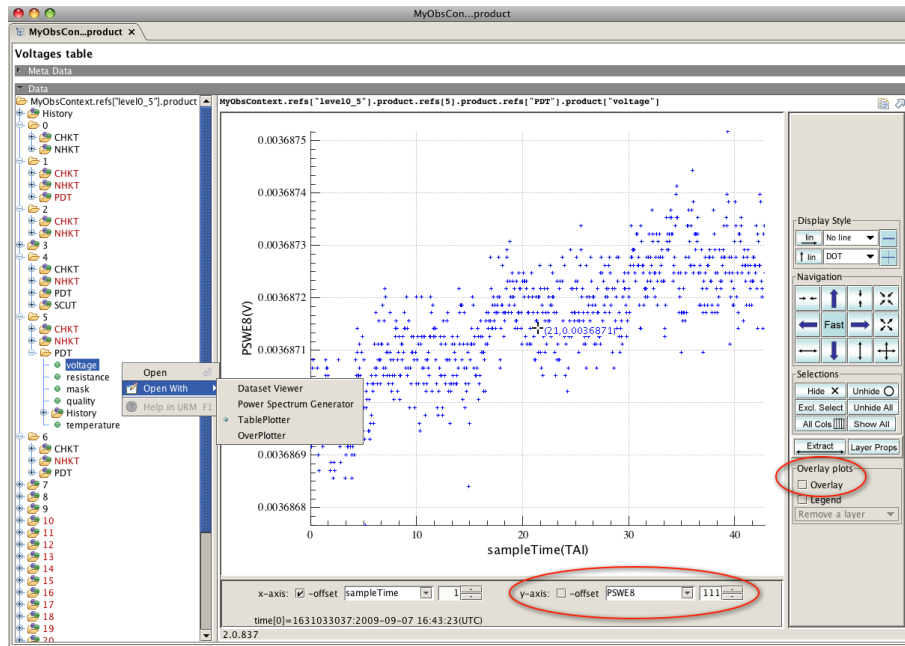**Figure 2.21. Inside the Level 0.5 Building Block structure for a Large Map observation**

Figure 2.22. Plotting the Level 0.5 data for a Large Map observation

2.3.5. 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 2.23, contains 30 entries, each corresponding to an individual block in the observation. the building block types are identical to the Level 0.5 data (see Table 2.2). *Clicking* on a given number within the Level 0 context reveals the products contained within each building block. These products are the *raw* data versions of the Level 0.5 `CHKT`, `NHKT`, `PDT`, `POT`, `SCUT` products and are simply prefixed by an "*R*". The Raw Photometer Detector Timeline (RPDT) product contains a single Table Dataset referred to as `PHOTF`. When we view this dataset (by *right-clicking* and selecting `Open-with - Dataset Viewer`, see Figure 2.23), 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 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 `PHOTFADCFLAGS` 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).

Figure 2.23. The Level 0 Raw Data within the Observation Context

2.4. SPIRE Spectroscopy Data Structure

2.4.1. SPIRE spectrometer introduction

This section is dedicated to familiarizing the reader with the appearance of the data from the SPIRE spectrometer within HIPE and how to visualize the data.

The introductory script *SPIRE_spectrometer_intro.py* guides the user through the steps described in the subsequent sections: A fully processed observation context is loaded into HIPE and inspected. Level-1 data products are extracted from the observation context and then visualized. Finally, portions of a data product are extracted and plotted, giving the user direct access to the data. The data, shown here, derive from an observation of the galaxy IC342. The observation was made on September 21, 2009, Herschel's Operational Day 130.

2.4.1.1. Load an observation context into HIPE

In HIPE, one can access the observation contexts from data pools as follows:

1. Declare a ProductStorage: i.e. the name of the pool:

```
storage = ProductStorage("name-of-pool")
```

2. Query for an observation context which is identified by its type being equal to OBS:

```
results = storage.select(Query("type=='OBS'"))
```

3. Load the observation into the HIPE session:

```
observation = results[0].product
```

The introductory script loads three observation contexts from three separate data pools. Please refer to the script for the exact syntax. An observation context is a HIPE object which can contain several data products.

2.4.1.2. Inspect an observation context in HIPE

HIPE provides convenient GUI tools to inspect an observation context. Begin with the observation context for the low resolution observation (OBSID=0x50001AB8). In the HIPE Variables View, select lrObservation with a right mouse click and then Open With > Observation Viewer. HIPE will present the Summary view of the observation, including the image of four spectra, one unapodized and one apodized, derived from each of the center detectors of the two SPIRE spectrometer detector arrays: SLWC3 and SSWD4. Clicking the small arrow to the left of Summary in the observation viewer will hide the observation summary and present the detailed view of the observation context:
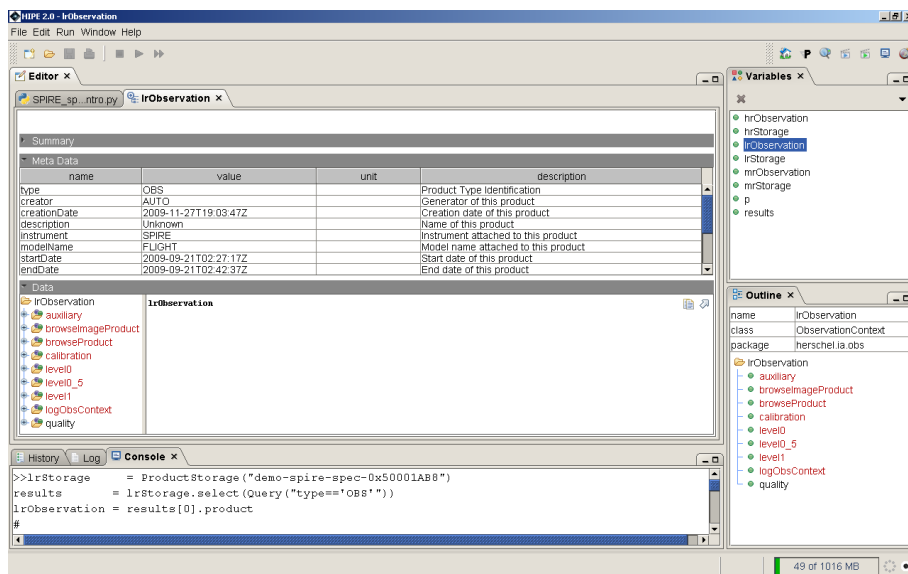

Figure 2.24. Viewing the SPIRE observation context

The viewing pane shows the many sub-contexts contained in the observation context in a folder-like layout.

Next, inspect the level-1 context. In the Data area of the Editor for lrObservation, select level1 with a right mouse click and select Open With > Context Viewer. Inside the Level 1 context there is one main entry named "Point_0_Jiggle_0_LR" which stands for the first and only raster point (index 0), the first and only jiggle position (index 0) at Low Resolution. This is the only building block contained in this observation. Double-click this building block to see the three entries it contains. Each one represents a different SPIRE spectrometer level-1 data product:

1. apodized_spectrum: Level 1 Apodized Spectrum Product

2. interferogram: Level 1 Interferogram Product

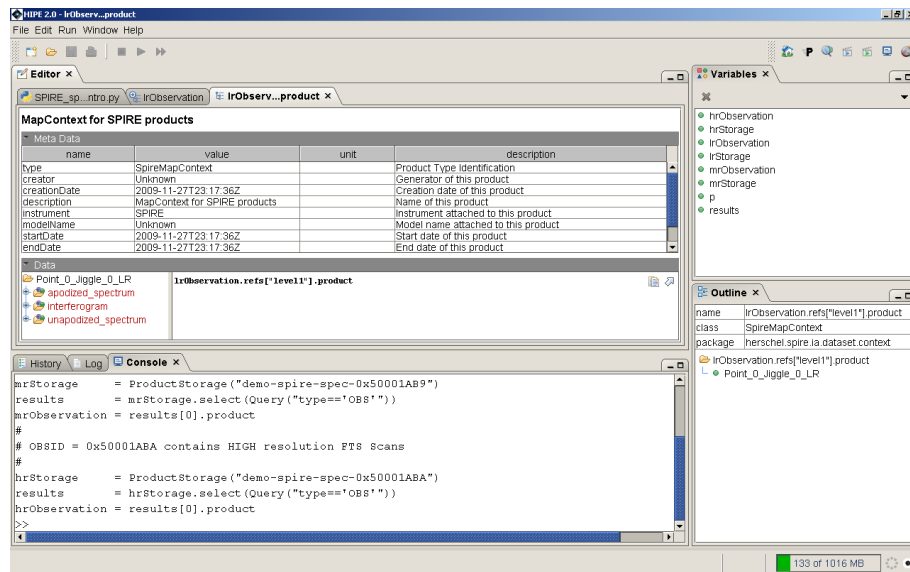3. unapodized_spectrum: Level 1 Unapodized Spectrum Product

Figure 2.25. Viewing the SPIRE Level 1 context

2.4.1.3. Extract the Level 1 data products

Before inspecting the contents of the level-1 data products, we first extract a selection of these products as separate variables in HIPE. The syntax required to access a level-1 product within an observation context is as follows:

```
Level1Product         =         observation.refs["level1"].product.
refs[BuildingBlock].product.refs[ProductName].product
```

For example, the following command will extract the level-1 interferogram product from the high resolution observation context:

```
hrInterferogram        =        hrObservation.refs["level1"].product.
refs["Point_0_Jiggle_0_HR"].product.refs["interferogram"].product
```

Note that the right hand side of this command is spelled out at the top of the Data area of the Context Viewer in HIPE. Clicking the copy icon at the top right corner will copy the command string into the clipboard and can then be pasted into the command console.

2.4.1.4. Inspect the Level 1 data products

HIPE offers dedicated visualization tools to inspect the level-1 interferogram and spectrum products.

The following steps demonstrate how one can inspect the contents of the datasets within a level-1 data product as tables. In this example, a dataset in the level-1 interferogram product of the high resolution observation is examined.

1. Select the hrInterferogram variable with right mouse click, select Open With > Product Viewer.

2. Scroll down to the bottom of the newly opened view. Within the folder-like structure, unfold Dataset 0001 by clicking the plus symbol to its left and select SLWC3 with a right mouse click. Select Open With > Dataset Viewer to view the numeric values of the dataset.

3. These values can be easily written into a text file with comma-separated values with the command quoted below. The equivalent command will work to save a particular spectrum into a text file:
```
asciiTableWriter(file="C:/SLWC3Interferogram.txt",
table=hrInterferogram["0001"]["SLWC3"])
```

```
asciiTableWriter(file="C:/SLWC3Spectrum.txt",
table=hrSpectrum["0000"]["SLWC3"])
```
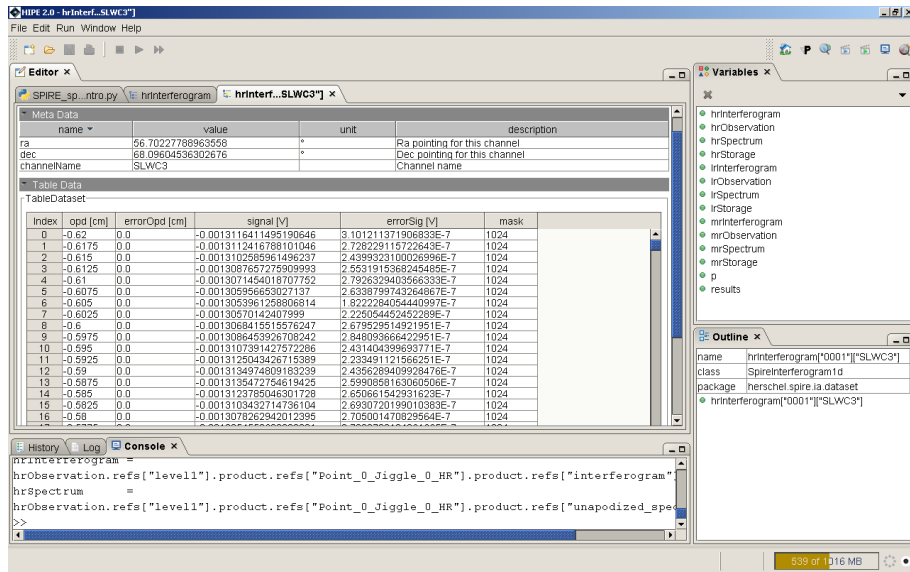
Figure 2.26. Inspecting data from a level-1 product as tables

The following steps demonstrate how one can conveniently plot the contents of the level-1 data product. In this example, the interferograms for a given detector in the level-1 interferogram product of the high resolution observation are examined.

1. In the Variables pane, select the hrInterferogram variable with right mouse click, select Open With > Spec SDI Explorer. Do the same for mrInterferogram, and lrInterferogram.

2. In the hrInterferogram view, select detector SLWC3 with a left mouse click. In the other views, select the same detector but do so with a *double-click* of the left mouse button to over-plot the interferograms.
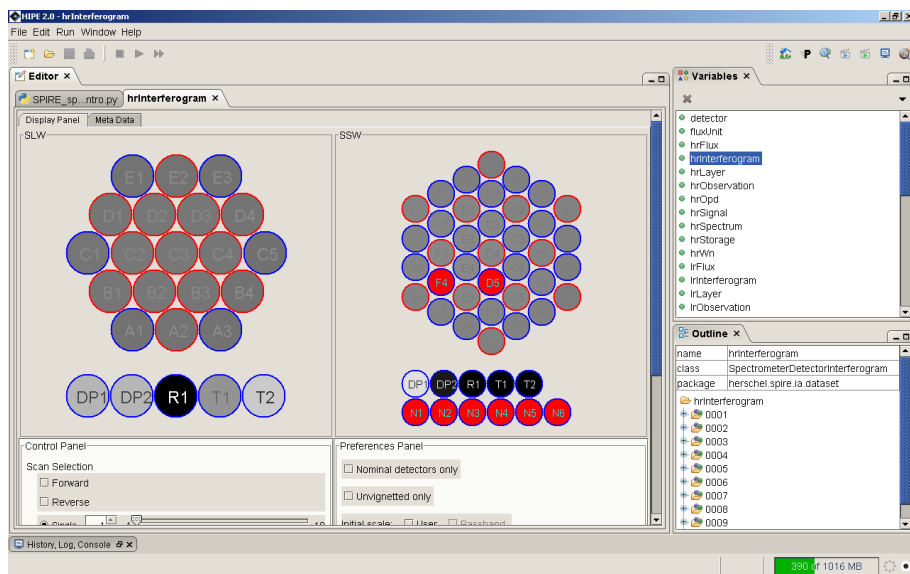

Figure 2.27. The SDI Explorer allows to select and plot data from a level-1 interferogram product

2.4.1.5. Extract and plot Level 1 data

The remainder of the *SPIRE_spectrometer_intro.py* demonstration script shows how to extract and plot interferograms and spectra:

1. Extract the individual data vectors from the product datasets

General syntax:

```
wave = spectrum[scanNumber][detector].getWave()

flux = spectrum[scanNumber][detector].getFlux()
```

Specific syntax:

```
hrWn = hrSpectrum[0]["SLWC3"].getWave()

hrFlux = hrSpectrum[0]["SLWC3"].getFlux()
```

2. Plot the results.

General syntax:

```
p = PlotXY()

p.addLayer(LayerXY(x,y))
```

Specific sample syntax:

```
detector = "SLWC3"

plotTitle = "Inspect Level 1 Spectra "+detector

p = PlotXY(titleText = plotTitle)

hrLayer = LayerXY(hrWn, hrFlux, name="HR")

p.addLayer(hrLayer)
```

After execution of the remainder of *SPIRE_spectrometer_intro.py*, the following plots should be displayed:
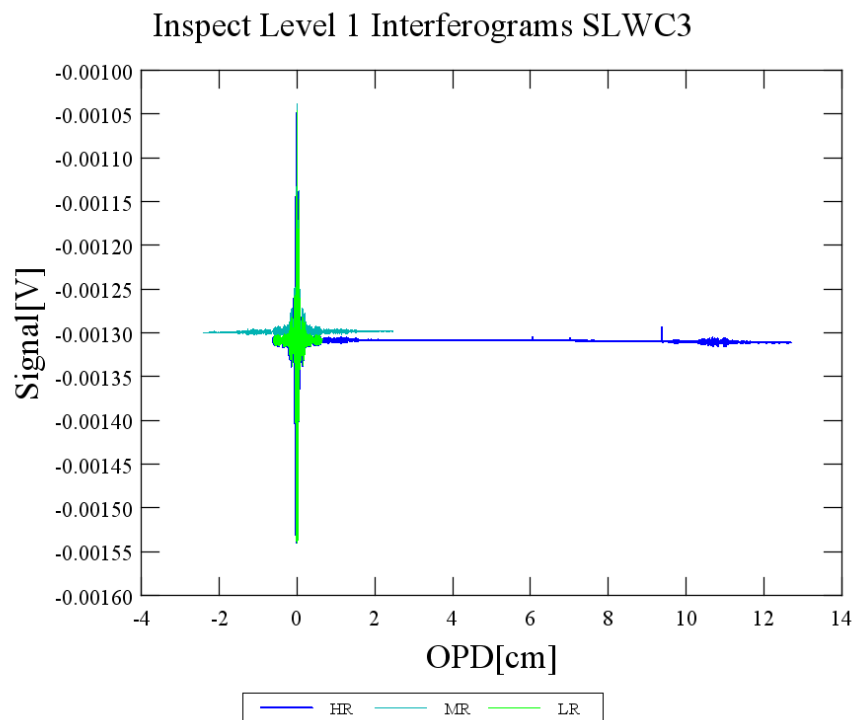
Inspect Level 1 Interferograms SLWC3

Figure 2.28. Comparing three interferograms from the SLWC3 detector

Inspect Level 1 Spectra SLWC3

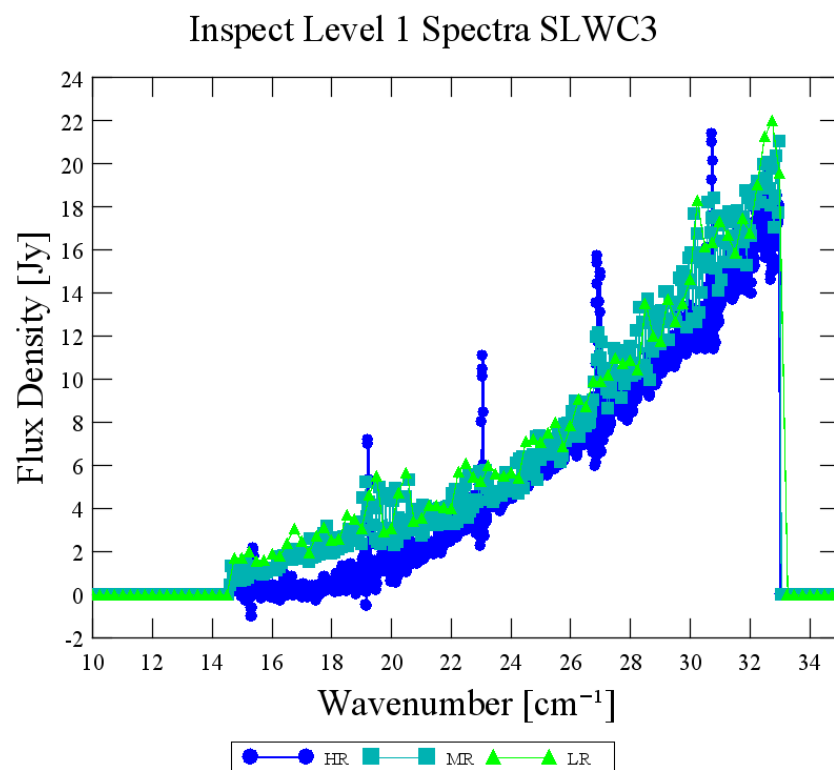**Figure 2.29. Comparing three spectra from the SLWC3 detector**

Chapter 3. Reprocessing your data

3.1. SPIRE Point Source Mode Data Processing

3.1.1. Reprocessing SPIRE Point Source Mode Data

Now that you have inspected your data products, you may feel that you would like to reprocess your data from the Level 0.5 products onwards, and in time to diverge away from the standard pipeline processing provided by the HSC. This chapter provides an overview of the steps required to process your datasets from Level 0.5 onwards, and on how to inspect your final Level 1 and Level 2 products.

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". The figure below outlines the steps requierd to process the Jiggle pipeline.

Figure 3.1. The SPIRE Jiggle mode pipeline.

First, we need to make sure that you have imported all needed classes and task definitions required to run the Large Map/Parallel Mode pipeline:

```
# Import all needed classes
from herschel.spire.all import *
from herschel.spire.util import *
from herschel.ia.all import *
from herschel.ia.task.mode import *
from java.lang import Long
from java.util import *

# Import the script tasks.py that contains the task definitions
from herschel.spire.ia.pipeline.scripts.POF2.POF2_tasks import *

# Import the script input.py that contains the input definitions
```

```
from herschel.spire.ia.pipeline.scripts.POF2.POF2_input import *
```

If you do not have Level 0.5 products to hand, you will need to make the engineering conversion first from the raw Level 0 products. First, we must search our local pool "1342183474_POF2_NGC5315" for our observation context:

```
store = ProductStorage()
myPool = LocalStoreFactory.getStore("1342183474_POF2_NGC5315")
store.register(myPool)

obs = QUERY_RESULT[0].product
```

We also need to set a workaround for the nodId, and extract the calibration products required for processing:

```
# workaround for wrong nodId in PFM data
nodId=[0L,1L,1L,0L]


# Extract from the observation context the calibration products that
# will be used in the script
bsmPos=obs.calibration.phot.bsmPos
bsmOps=obs.calibration.phot.bsmOps
detAngOff=obs.calibration.phot.detAngOff
elecCross=obs.calibration.phot.elecCross
optCross=obs.calibration.phot.optCross

# Extract from the observation context the auxiliary products that
# will be used in the script
hpp=obs.auxiliary.pointing
siam=obs.auxiliary.siam
```

Run the engineering pipeline from the Level 0 products obtained from the HSA to Level 0.5:

```
level0_5= engConversion(obs.level0,cal=obs.calibration)

# Add the result to the observation in level 0.5
obs.level0_5=level0_5
```

Set dpparr as an array to host input from the demodulated data, and obtain the number of building blocks from the Level 0.5 products:

```
dpparr=[DenodInput()]
nrep=1
nblocks=len(level0_5.getBbids(0xa321))
```

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 angle and Z angle timeline and then into a chopper id and jiggle id timeline. We use these to create the SPIRE pointing product.

The next step then is perform a number of corrections to the data - the electrical crosstalk coreection, deglitching, flux conversion, sky position association, demodulation of the data, second level deglitching and averaging of the demodulation data. Lastly, we extract the number of cycles and add the demodulated data to the "dpparr" variable as input to the denodding module.

```
for bbid in level0_5.getBbids(0xa321):
  print "Starting BBID=",hex(bbid)
  block=level0_5.get(obsid,bbid)
  # Get basic engineering data products
  pdt  = block.pdt
  bsmt = block.bsmt
```

```
#
#
# run the task to convert BSM telemetry in a Y angle and Z angle timeline
bat=calcBsmAngles(bsmt,bsmPos=bsmPos)

# run the task to convert BSM telemetry in a chopper id and jiggle id timeline
cjt = calcBsmFlags(bsmt, bsmOps=bsmOps)
#
#create the SpirePointingProduct
spp=createSpirePointing(detAngOff=detAngOff,bat=bat,hpp=hpp, siam=siam)
#
# run the electrical crosstalk correction
pdt=elecCrossCorrection(pdt,elecCross=elecCross)
#
# run the deglitch
pdt=deglitchTimeline(pdt, scaleMin=1.0, scaleMax=8.0, scaleInterval=5,
holderMin=-1.6,\
  holderMax=-0.1, correlationThreshold=0.6, correctGlitches=inputs.correctGlitches)
#
# run the flux conversion

fluxConv=obs.calibration.phot.fluxConvList.getProduct(pdt.meta["biasMode"].value,pdt.startDate)
pdt=photFluxConversion(pdt,fluxConv)
#
# associate the sky position
ppt=associateSkyPosition(pdt,spp=spp)
#
# run the Demodulation task
dpp = demodulate(ppt, cjt=cjt)
#
# second level deglitching
dpp = secondDeglitching(dpp)
#
# average on jiggle position
dpp = jiggleAverage(dpp)

ncyc=((dpp.bbCount-1)/4)+1
if ncyc >= nrep+1:
  for k in range(ncyc-nrep):
   dpparr.append(DenodInput())
  dpparr[ncyc-1].addProduct(dpp)
  nrep=ncyc
else:
  dpparr[ncyc-1].addProduct(dpp)
print "Completed BBID=",hex(bbid)
```

We can now inspect the output from the demodulation module - below is the demodulated dpp product
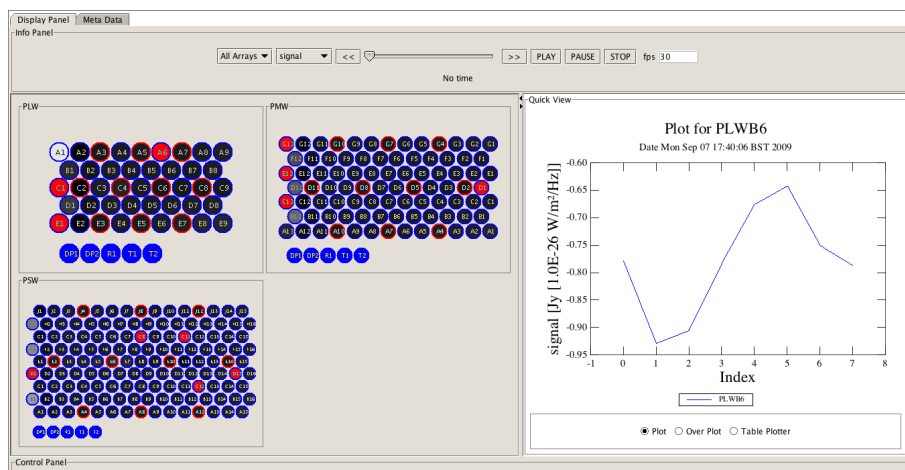data for the PLWB6 detector:

Figure 3.2. DPP product - PLWB6 detector

Now we can apply the denodding module to the demodulated data in order to remove the nodding, and append the denodded data to the PhotPointedProduct (PPP):

```
ppps=[]
for i in range(nrep):
 denin=dpparr[i]
 ppp=deNodding(denin)
 ppps.append(ppp)
```
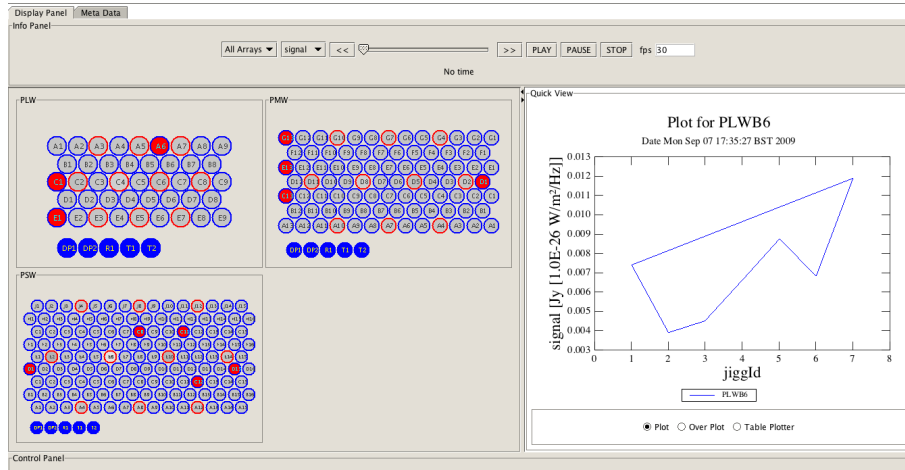
Inspecting the denodded PPP:

Figure 3.3. PPP product - PLWB6 detector

Finally, we can average the denodded data, per nodID:

```
appp = nodAverage(ppps)
```

We now have our Level 1 product - the Averaged Phot Pointing Product (APPP). Inspecting the APPP via the Detector Timeline Explorer:
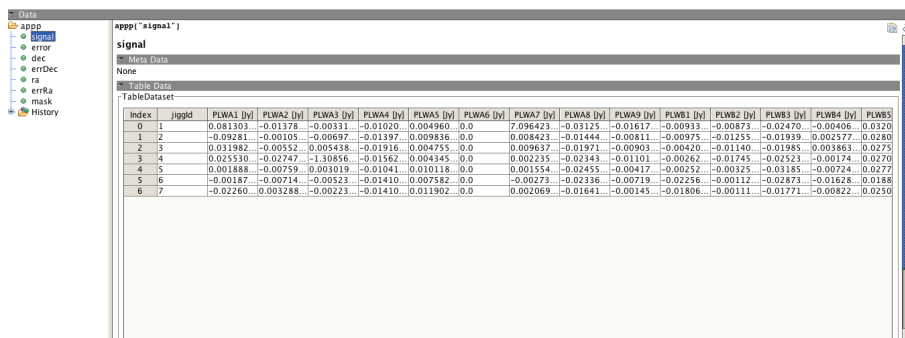

Figure 3.4. APPP product - PLWB6 detector

Finally, 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:

```
# user products
jpsfp = pointSourceFit (appp)
jpp   = sourceFlux (jpsfp)
```

Figure 3.5. JPPP product

Congratulations! You have now successfully reprocessed your point source data from Level 0 to the final Level 2 user products!

3.2. Reprocessing SPIRE Large Map and Parallel Mode Data

As mentioned earlier, 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 Level 0.5 data products from the HSC, and have stored them in a storage pool"1342183475_POF5_NGC5315". The pipeline for **Level 0.5 to Level 1 processing** involves the following sequence of processing modules. The pipeline works on a Photometer Detector Timeline (PDT) and requires the Nominal Housekeeping Timeline (NHKT). Additional auxilliary products are required for the telescope pointing information (see the flowchart below)
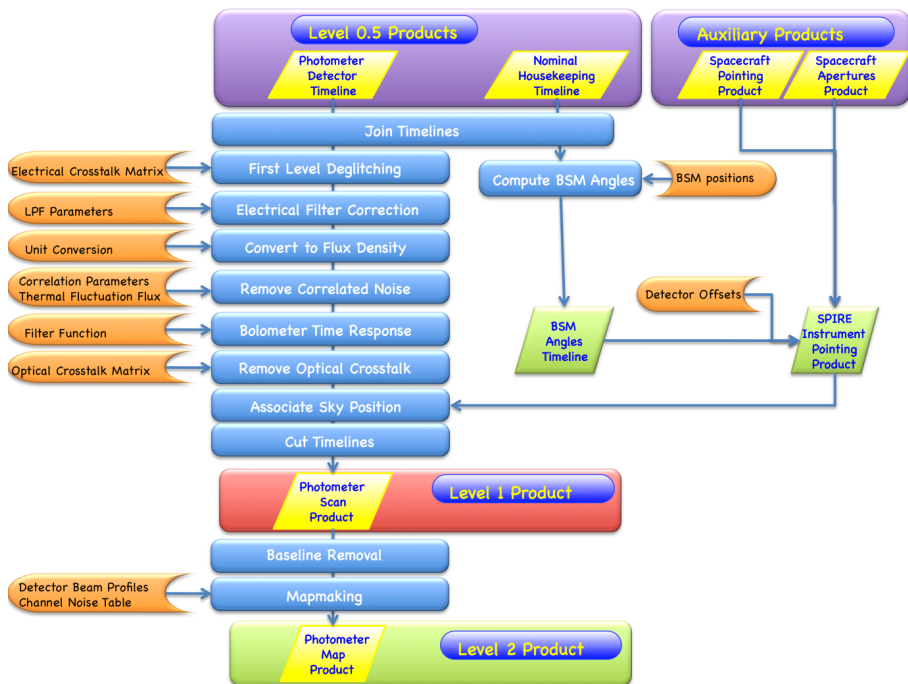

Figure 3.6. The SPIRE Large Map/Parallel mode pipeline.

First, we need to make sure that you have imported all needed classes and task definitions required to run the Large Map/Parallel Mode pipeline:

```
from herschel.spire.all import *
from herschel.spire.util import *
from herschel.ia.all import *
from herschel.ia.task.mode import *
from herschel.ia.pg import ProductSink
from java.lang import *
from java.util import *
from herschel.ia.obs.util import ObsParameter
from herschel.spire.ia.pipeline.scripts.POF5.POF5_tasks import *
from herschel.spire.ia.pipeline.scripts.POF5.POF5_input import *

store = ProductStorage()
myPool = LocalStoreFactory.getStore("1342183475_POF5_NGC5315")
store.register(myPool)

obs = QUERY_RESULT[0].product
```

If you do not have Level 0.5 products to hand, you will need to make the engineering conversion first from the raw Level 0 products, determine the number of scan lines to be processed and set up a variable to keep track of saved prtoducts:

```
# Extract from the observation context the calibration products that
# will be used in the script
bsmPos=obs.calibration.phot.bsmPos
lpfPar=obs.calibration.phot.lpfPar
detAngOff=obs.calibration.phot.detAngOff
elecCross=obs.calibration.phot.elecCross
optCross=obs.calibration.phot.optCross
chanTimeConst=obs.calibration.phot.chanTimeConst
chanNum=obs.calibration.phot.chanNum

# Extract from the observation context the auxiliary products that
# will be used in the script
hpp=obs.auxiliary.pointing
siam=obs.auxiliary.siam


# Set this to FALSE if you don't want to use the ProductSink
# and do all the processing in memory
tempStorage=Boolean.TRUE

# Run the engineering conversion pipeline
level0_5= engConversion(obs.level0,cal=obs.calibration, tempStorage=tempStorage)

# Attach the result to the observation in level 0.5
obs.level["level0_5"]=level0_5

# Get the list of BBIDs of scan lines (BBID means Building Block IDs)
# 0xa103 is the BBTYPE of scientific data for scan map.
bbids=level0_5.getBbids(0xa103)

# number of scans lines to be processed
nscans=len(bbids)
print "number of scan lines:",nscans

# this variable will be used to keep references
# to saved products
pdtList=[]
```

Start the pipeline running the first correction - the Electrical Crosstalk Correction. We can execute this in a loop for all scan lines:

```
# start the pipeline running the first correction:
# Electrical Crosstalk Correction
# This is executed in a loop for all scan lines
for bbid in bbids:
 block=level0_5.get(obsid,bbid)
 pdt=block.pdt
 pdt=elecCrossCorrection(pdt,elecCross=elecCross)
 pdtList.append(sink.save(pdt))
```

Now,for this observation, we know that detector timeline #5 contains a glitch in detector "PLWB5".

```
pdt=pdtList[5].product
```

We can start to take steps to correct this glitch. First we get the voltage of detector "PLWB5". The getVoltage() method is defined for DetectorTimeline objects:

```
voltage=pdt.getVoltage("PLWE8")
```

Next we get the sample times. We are using a jython syntax to call the method getSampleTime() defined for DetectorTimeline objects:

```
time=pdt.sampleTime
```

Here we shift the time origin to center on the glitch:

```
time=time-time[135]
```

Get the name of the unit of the voltage:

```
uni=pdt.getVoltageUnit("PLWE8").toString()
```
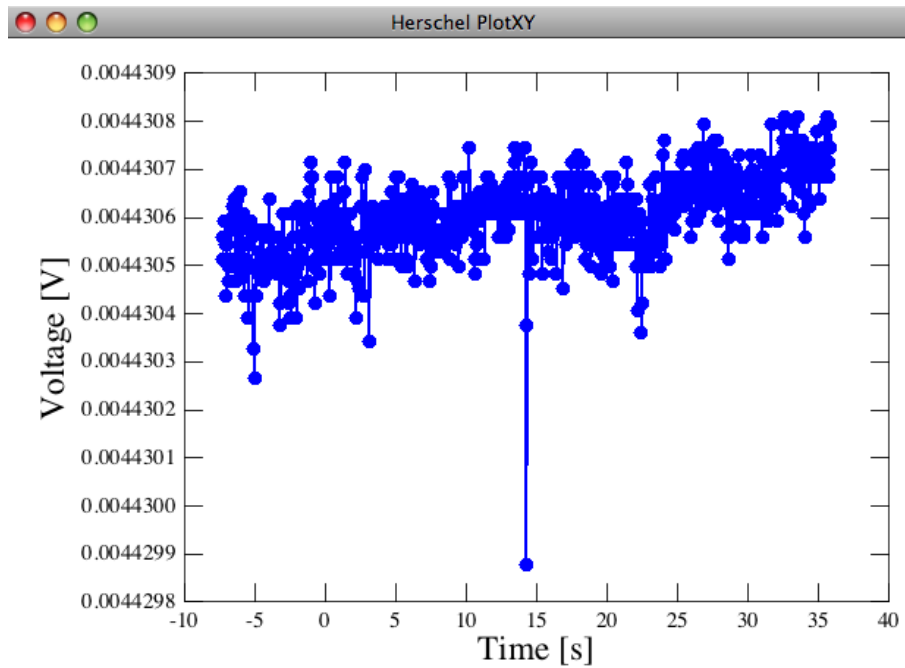
Figure 3.7. Plotting voltage against time.

Now we can plot the voltage versus time to view the glitch:

```
# Plot the voltage versus time
plot1=PlotXY(time,voltage,color=Color.blue,\
 xtitle="Time [s]",ytitle="Voltage ["+uni+"]")
plot1[0].style.stroke=1
plot1[0].style.line=2
plot1[0].style.symbol=14
```

To correct, we run deglitching on all scan lines:

```
for i in range(nscans):
 pdt=pdtList[i].product
 pdt=deglitchTimeline(pdt)
 pdtList[i]=sink.save(pdt)
```

Now we get the same timeline after deglitching:

```
# now I get the same timeline after deglitching
pdt_deg=pdtList[5].product
```

Again we get the voltage of detector PMWA13:

```
volt_deg=pdt_deg.getVoltage("PLWE8")
```
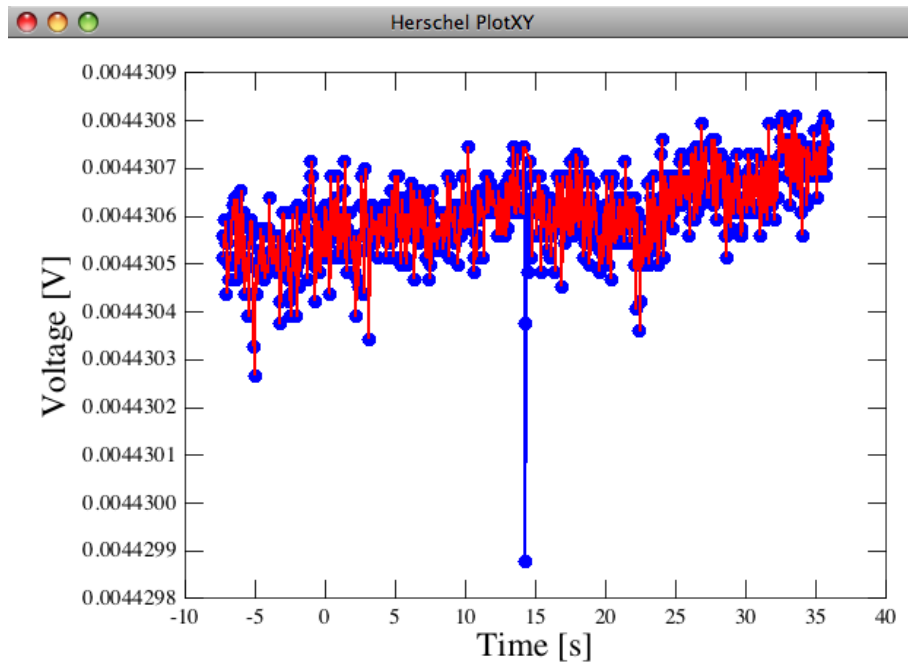
Figure 3.8. And we can overplot on the old timeline.

Overplot on the old timeline:

```
plot1[1]=LayerXY(time,volt_deg,color=Color.red)
plot1[1].style.stroke=1
```

Now we apply the Electrical Filter Response Correction

```
for i in range(nscans):
 pdt=pdtList[i].product
 pdt=corrElecFiltResponse(pdt)
 pdtList[i]=sink.save(pdt)
```

Now we can set up and run the Flux Conversion task:

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

for i in range(nscans):
 pdt=pdtList[i].product
 pdt=photFluxConversion(pdt,table=obs.calibration.phot.fluxConv)
 pdtList[i]=sink.save(pdt)
```

Apply correction for temperature drift

```
for i in range(nscans):
 pdt=pdtList[i].product
 pdt=temperatureDriftCorrection(pdt,table=obs.calibration.phot.tempDriftCorr)
 pdtList[i]=sink.save(pdt)
```

Get the corrected PDT:

```
pdt_corr=pdtList[0].product
```

Get the signal of the same detector:

```
signal_corr=pdt_corr.getSignal("PSWE10")
```

Let's look at the voltage of the PSWT1 thermistor:

```
signal_pswt1=pdt_corr.getSignal("PSWT1")
plot3=PlotXY(time,signal_pswt1,color=Color.blue,\
xtitle="Time [s]",ytitle="PSWT1 Voltage [V]",name="Thermistor voltage")
```

Apply the bolometer response correction:

```
for i in range(nscans):
 pdt=pdtList[i].product
 pdt=corrBolTimeResponse(pdt)
 pdtList[i]=sink.save(pdt)
```

Apply the Optical Crosstalk Correction:

```
for i in range(nscans):
 pdt=pdtList[i].product
 pdt=photOptCrossCorrection(pdt,optCross=optCross)
 pdtList[i]=sink.save(pdt)
```

Create a Spire Pointing Product:

```
spp=SpirePointingProduct(detAngOff=obs.calibration.phot.detAngOff,\
    hpp=obs.auxiliary.pointingProduct,siam=obs.auxiliary.siamProduct)
```

Create a ScanContext where we will attach all the timelines. This will be used as input for map making:

```
scanCon=ScanContext(obsid)
scanCon.modelName=obs.level["level0"].modelName
```

In this loop we compute the pointing:

```
for i in range(nscans):
 block=level0_5.get(obsid,bbids[i])
 nhkt  = block.nhkt
# calculate BSM angles
bat=calcBsmAngles(nhkt,bsmPos=obs.calibration.phot.bsmPos)
#
# add the Bsm Angles Timeline to the SpirePointingProduct
spp.bat=bat
# associate sky positions to flux samples
pdt=pdtList[i].product
ppt=associateSkyPosition(pdt,spp=spp)
scanCon.refs.add(sink.save(ppt))
```

We now need to set up and remove the baseline before we can generate our Level 1 mapping products:

```
 Flag to switch on and off the baseline removal
useRemoveBaseline=True

# create a SpireListContext to be used as input of map making
scans=SpireListContext()
```

```
# Run baseline removal and populate the map making input
for i in range(level1.count):
 if useRemoveBaseline:
  pdt=level1.getProduct(i)
  pdt=removeBaseline(pdt,chanNum=chanNum)
  if tempStorage:
   ref=ProductSink.getInstance().save(pdt)
   scans.addRef(ref)
  else:
   scans.addProduct(pdt)
 else:
  scans.addRef(level1.refs[i])
pass
```

Level 1 to Level 2 processing (using Naive Mapping or MadScanMapper) for the mapping pipeline processing produces the final *PLW/PMW/PSW* products.

Run naive map making for the three bands:

```
mapPlw=naiveScanMapper(scans, array="PLW")
mapPmw=naiveScanMapper(scans, array="PMW")
mapPsw=naiveScanMapper(scans, array="PSW")
```

Save maps in the sink and attach them in the ObservationContext

```
level2=MapContext()
level2.refs.put("PLW",sink.save(mapPlw))
level2.refs.put("PMW",sink.save(mapPmw))
level2.refs.put("PSW",sink.save(mapPsw))

obs.level["level2"]=level2
obs.obsState = ObservationContext.OBS_STATE_LEVEL2_PROCESSED
```

Saving the data maps for each photometer array

When the pipeline is finished running, a new dialog will appear on screen, asking you whether you wish to save the processed ObservationConext. Click yes to proceed. This enables you to save the final observation context in a new location.
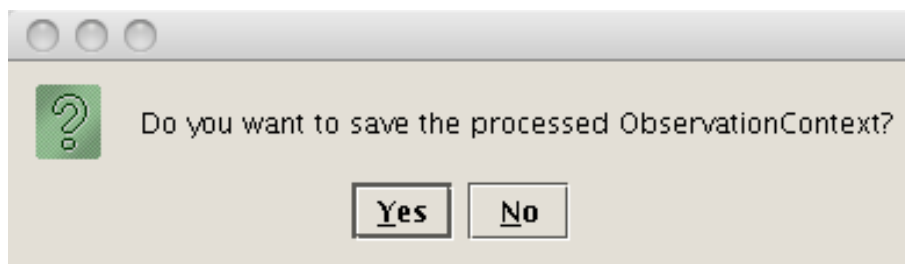

Figure 3.9. Observation context dialog.

Now enter the name of the pool where the user wants to save all the processed data in the dialog that pops up.

Saving the data maps for each photometer array

In order to browse the processed data, within the 'Variables' window, select 'obs' from the list of the available variables - this is the variable containing the final observation context.

We can now access the level 1 and level 2 product maps from the Large scan map pipelines, namely the PSW, PLW and PMW map products:
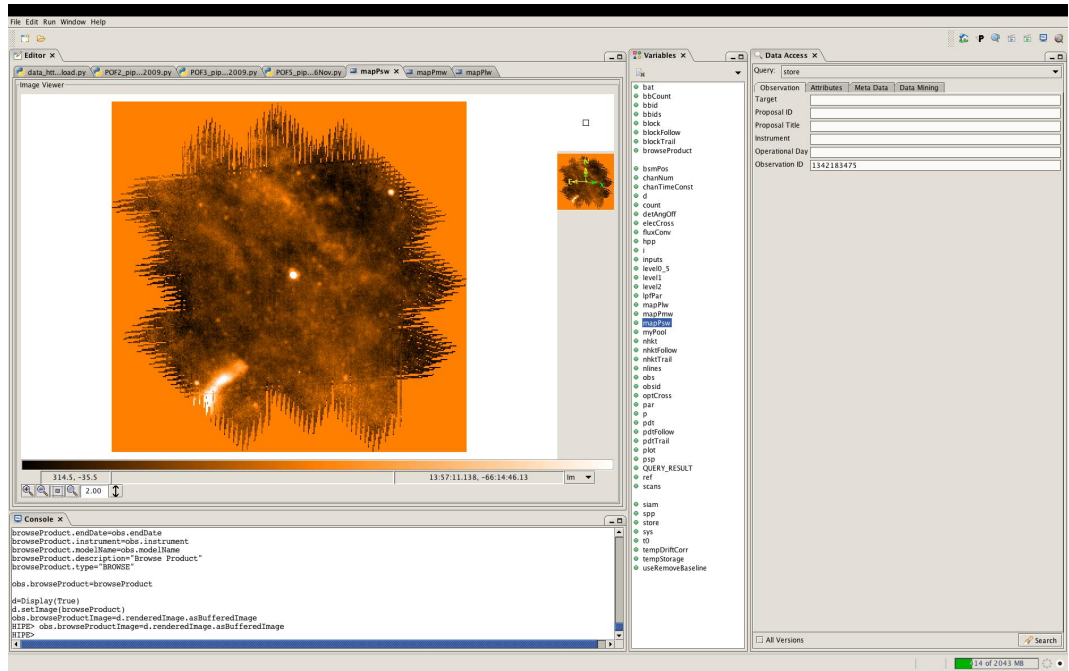
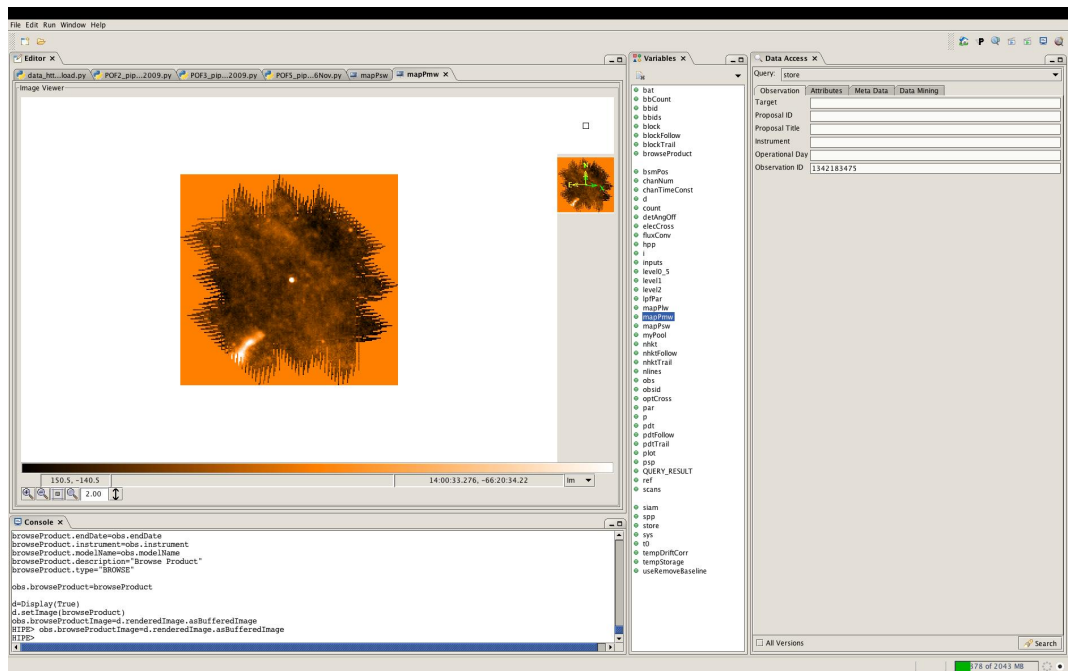**Figure 3.10. Level 2 PSW product for NGC 5315 from Large Map pipeline.**

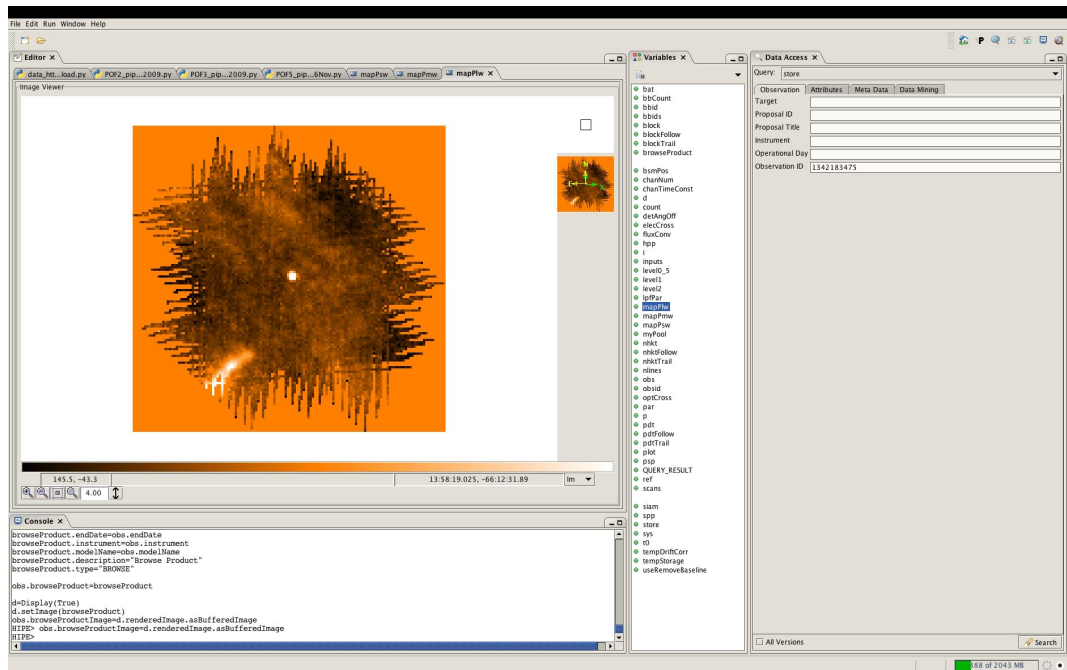**Figure 3.11. Level 2 PMW product for NGC 5315 from Large Map pipeline.**

Figure 3.12. Level 2 PLW product for NGC 5315 from Large Map pipeline.

3.3. SPIRE Spectroscopy Data Processing

3.3.1. Reprocessing SPIRE spectrometer data

Two pipelines process data from the observations with the SPIRE imaging spectrometer, one for point or one for 4 and 16 point jiggle observations. In either case the observation may consist of a set of observations at a raster of points in the sky. Data will undergo a sequence of processing steps which are illustrated schematically in Figure 3.13.

Figure 3.13. The SPIRE Spectrometer pipeline.

The script *SPIRE_spectrometer_apodization.py* demonstrates how to process SPIRE FTS data inter-
actively. Rather than processing the data through the whole pipeline, this demonstration focuses on one
particular processing module: Apodization. The Instrumental Line Shape of the SPIRE spectrometer
is a very similar to a sinc function which has an infinite number of side-lobes with decreasing ampli-
tude. The side-lobes can pose challenges when trying to identify lines against a noisy spectral baseline.
Apodization is a technique by which the side-lobes are reduced by multiplying interferograms with
a tapering function, i.e. convolving the spectrum with a smoothening kernel. The following sections
demonstrate the effects of applying different apodizing functions on the instrumental line shape. There
can be many other reasons to process data, such as making use of alternative task parameters or updated
calibration products. For this demonstration, the high resolution observation (OBSID=0x50001ABA)
is used. The script is divided into four main sections

1. Load the observation context and reprocess the level-1 interferograms.

2. Apply different apodizing functions to the interferograms.

3. Process the apodized interferograms to level-1 spectra.

4. Compare the resulting spectra.

3.3.1.1. Load the observation context and reprocess interferograms

With the *SPIRE_spectrometer_apodization.py* script, load the high resolution observation and reprocess the Level-1 interferogram up to, but not including the Apodization step. This entails applying the following steps:

1. SCAL and Telescope Correction, to account for the emission from the instrument and the warm telescope (loading the reference interferogram manually if required)

2. Interferogram Baseline Correction, to account for vignetting that changes as a function of Optical Path Difference or 1/f-like noise.

3. 2nd Level Deglitching, which compares a particular interferogram to the average interferogram to identify outliers and correct these samples.

4. Phase Correction

   a. Determine the phase of the interferogram by first apodizing the double-sided interferogram and then applying the Fourier transform to the double-sided portion of the interferogram.

   b. Correct the measured interferograms based on a fit to the measured phase.

The *SPIRE_spectrometer_apodization.py* script performs all these processing steps by executing all commands up to the following line:
```
# Finished reprocessing up to the apodization step.
```

3.3.1.2. Apply different apodizing functions to the interferograms

The next processing step – apodization – can be performed while setting the task control parameter to different values. This particular demonstration applies several apodizing functions which are given below. The number refers to the line shape broadening introduced by the apodization:

1. No Apodization (i.e. skip the apodization step)

2. Norton-Beer 1.2

3. Norton-Beer 1.5

4. Gaussian 1.9

The *SPIRE_spectrometer_apodization.py* script will apply these different apodizing functions by executing all commands up to the following line:
```
# Finished applying the selected apodizing functions.
```

The *SPIRE_spectrometer_apodization.py* script deletes all but the detector of interest from the reprocessed interferogram in order to lower the memory requirements and reduce the execution time. The script makes multiple copies of the reprocessed interferogram product, one for each of the selected apodizing functions. This is required in this particular instance because, by default, the processing tasks overwrite their input. The following command creates a deep copy of the level-1 interferogram product:
```
new_ifgm = SpectrometerDetectorInterferogram(interferogram)
```

3.3.1.3. Process the apodized interferograms to level-1 spectra

The consequences of applying the different apodizing functions are best evaluated by comparing the instrumental lines shapes n the level-1 spectra. To that end, the different interferogram products are processed to level-1 spectra by applying:

1. Fourier Transform, to transform interferograms into spectra.

2. Flux Conversion, to apply absolute flux calibration.

3. Spectral Averaging, to average spectra across all performed scans.

The *SPIRE_spectrometer_apodization.py* script will apply these processing steps by executing all commands up to the following line:

```
# End of reprocessing
```

3.3.1.4. Compare the resulting spectra

The remainder of the script extracts and plots the wave and flux columns from the reprocessed spectra, leading to the following plot:
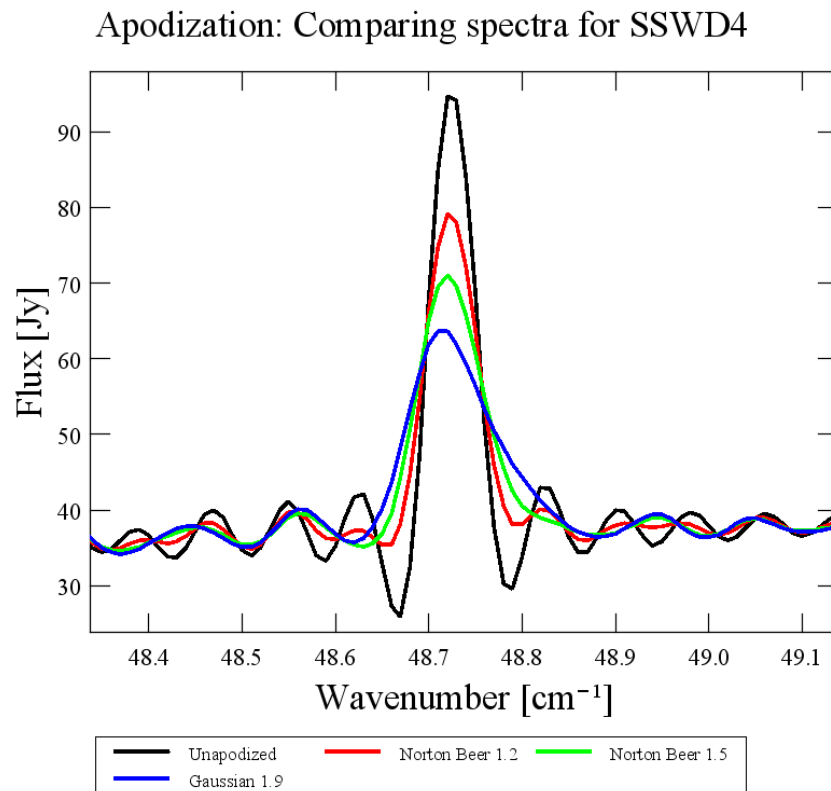

Figure 3.14. Comparing four spectra from the SSWD4 detector

The plot shows that apodizing functions reduce the amplitude of the side-lobes relative to the center burst as desired. However, they also deprecate the center peak amplitude and more aggressively widen the instrumental line shape the stronger they are. This example also illustrates that apodization can lead to a shift in the line center if there is significant slope in the spectral shape or, as in this case, in the calibration data.

3.3.2. Additional reading

Additional and more detailed information regarding the data processing modules and the data at the various levels of processing can be found in the Spectrometer Pipeline Description.