

# **The HIFI Data Reduction Guide**

**Editor: Sylvie F Beaulieu**

**co-Editor: Carolyn McCoey**

**HIFI ICC**

# The HIFI Data Reduction Guide

Editor: Sylvie F Beaulieu  
 co-Editor: Carolyn McCoey  
 HIFI ICC

<b>Revision History</b>		
Revision 0	Prior to 3 May 2011	
No revision history.		
Revision 1	3 May 2011	
Added information about convolution table from doGridding.		
Revision 2	5 Aug 2011	
Name changed from "HIFI Users' Manual" to "HIFI Data Reduction Guide", updated and clarified text regarding re-processing data with new calibration.		
Revision 3	8 Oct 2011	
Addition of HIFI Launch Pad, Point Mode DBS cookbook, and a chapter about flagging HIFI data.		
Revision 4	17 Oct 2011	
Updates to pipeline chapters, quality flag section, flagging data chapter, addition of unit conversion chapter.		
Revision 5	9 Nov 2011	
Updates to deconvolution and standing wave chapters.		
Revision 6	14 Nov 2011	
Update to the "What was done to my data?" chapter to make clear that lines are at SSB scale while continuum is at DSB scale.		
Revision 7	23 Nov 2011	
Addition of documentation for new flagging task, FlagTool. Documentation of various small known issues and work arounds for HIPE 8.		
Revision 8	20 Jan 2012	
Updates to the "How to make a spectral cube" chapter for updates in HIPE 8 and updates to the fitBaseline chapter for updates to fitBaseline in 8.1.		
Revision 9	21 May 2012	
Updates for HIPE 9.0, significant updates to the "Running the HIFI pipeline", "Tour of a HIFI ObservationContext", "Viewing Spectra" chapters, updates to the "Flags in HIFI data", "How to the add and remove flags" and "How to create a spectral cube" chapters. Bug-fixes and typo corrections elsewhere.		
Revision 10	23 June 2012	
Further updates for HIPE 9.0, updates to Baseline Removal and Standing Wave Removal chapters.		
Revision 11	10 August 2012	
Addition of a chapter describing how to send HIFI spectra to VO tools for HIPE 9.1.		
Revision 12	16 October 2012	
Addition of a chapter describing how to convert positions in data to offsets, improvements to "How to add and remove flags" and "Running the HIFI pipeline" chapters, various typos and bug fixes.		
Revision 13	24 October 2012	

Major update of "Setting flags interactively for many spectra" section in the "How to add and remove flags" chapter, removal of references to the old CubeSpectrumAnalysisToolbox, update to mapping browse product description in "Tour of a HIFI observation context" chapter, improvements to style consistency.		
Revision 14	26 November 2012	
Updates to "Sideband Deconvolution" chapter explaining how to correct baseline issues in the ssb result and how to export the ssb result to Class; minor updates to "Running the HIFI pipeline", "Tour of and Observation Context", and "What was done to my data" chapters to reflect changes in the pipeline, particularly the addition of mkDbsReference; further improvements to style consistency.		
Revision 15	12 December 2012	
Fix broken links and missing figures for HIPE 10 release.		
Revision	25 January 2013	
Addition of On The Fly (OTF) Mapping cookbook.		
Revision 16	20 February 2013	
Updates to "How to add and remove flags" chapter, "Setting flags interactively for many spectra" section. The task "flagTool" allows for the editing of rowflags.		
Revision 17	15 April 2013	
Updates to "Flags in HIFI data" chapter, "Quality Flags" section.		
Revision 18	23 April 2013	
Updates to "HIFI Baseline Removal" chapter, "FitBaseline Options" section.		
Revision 19	26 April 2013	
Updates to "Exporting HIFI data to CLASS" chapters.		
Revision 20	29 April 2013	
Updates to "Standing Wave Removal" chapter.		
Revision 21	9 July 2013	
Updates to "How to make a spectral cube" and "Exporting HIFI data to CLASS" chapters.		
Revision 22	25 September 2013	
Minor updates to sections 1.1 (added a figure), 4.2 (doPointing instructions), 5.4 (repipelining with new Auxiliary products), 12.3 (doDeconvolution INFO tables).		
Revision 23	9 October 2013	
doGridding: added a warning on flux conservation in spectral cubes ; fitBaseline: added a caveat on negative flux.		
Revision 24	31 October 2013	
Added one new flag in Quality Flag chapter. Added an extra detail in definition of "spur_rejection" in the Deconvolution chapter.		
Revision 25	7 November 2013	
In chapter 9. How to add and remove flags, section 9.5, changed the GUI figure for flagTool. In chapter 10, Standing Wave Removal: added the GUI figure for FitHifiFringe + changed input param to "data". In chapter 11, HIFI Baseline Removal: changed the GUI figure for FitBaseline + changed input param to "data". In chapter 20, Exporting HIFI data to CLASS, section 20.1, added a note for HIPE 12 - right-click on variable open hiClass GUI.		
Revision 26	21 November 2013	
Added details to the "HIFI Auxiliary Data" section of the "Tour of a HIFI ObservationContext" chapter.		
Revision 27	27 November 2013	

Added Spectral Scan Cookbook in Chapter 2: HIFI Cookbooks.		
Revision 28	28 November 2013	
Added Point Mode Frequency Switch Cookbook in Chapter 2: HIFI Cookbooks.		
Revision 29	29 November 2013	
Added Point Mode Load Chop Cookbook in Chapter 2: HIFI Cookbooks.		
Revision 30	12 December 2013	
Updated chapter "Standing Wave Removal", section "Electrical Standing Wave correction in HEB bands".		
Revision 31	16 January 2014	
New updates to chapter "Standing Wave Removal", section "Electrical Standing Wave correction in HEB bands".		
Revision 32	21 January 2014	
Added Point Mode Position Switch Cookbook in Chapter 2: HIFI Cookbooks.		
Revision 33	21 February 2014	
Updates to: Chapter 1: update method to import Herschel Data to HIPE. Chapter 11: update method to create a mask. Chapter 13: new syntax to command line for doGridding. Section 20.4: clarify text to create a FITS file.		
Revision 34	27 February 2014	
Added a note to Chapter 4, Level 1, doOffSubtract. Additional updates to Chapter 13: new syntax to command line for doGridding.		
Revision 35	12 March 2014	
Updates: Chapter 2, Position Switch cookbook: improved script for checking OFFs. Chapter 16: Units Conversion - included LoFrequency description.		
Revision 36	4 April 2014	
Updates: Chapter 2, extension of OTF mapping cookbook to a general mapping cookbook including DBS raster maps		
Revision 37	26 April 2014	
Updates: Chapter 17, update to polarPair documentation; Chapter 2, improvements to mapping cookbook.		
Revision 38	15 Auguts 2014	
Updated "Running the HIFI Pipeline" for HIPE 13 and updated flagTool documentation.		
Revision 39	15 September 2014	
Update to "Setting flags interactively for many spectra" section in the "How to add and remove flags" chapter - maskTables: Linemasks and Rowmasks files.		
Revision 40	9 October 2014	
Added a new chapter "The HIFI line identification tool (identifyLines task)".		
Revision 41	12 November 2014	
Inclusion of all HIFI-specific mathematical tasks in, and general update of, "Mathematical Operations on Spectra" chapter and update to the "Sideband Deconvolution" chapter.		
Revision 42	25 November 2014	
Updates to "Creating a Spectral Cube", "Standing Wave Removal", and "Unit Conversions" chapters for HIPE 13 release.		
Revision 43	10 December 2014	

Addition of "Understanding and using HIFI beam information in your data" chapter, further updates to "Unit Conversions" chapter, instructions for use of pipeline to calculate rms noise in data added to "How to run the HIFI pipeline" chapter.		
Revision 44	21 January 2015	
Changes in parameters syntax for fitHifiFringe, chapter 'Standing Wave Removal': new syntax - startPeriod, endPeriod, typicalPeriod, and subBase.		
Revision 45	6 February 2015	
HIPE 13 release: small updates to "What was Done to My Data", "Understanding and using HIFI beam information in your data", "Flags in HIFI data", "How to add and remove flags", "Standing Wave Removal", "Sideband Deconvolutions", "Exporting HIFI data to CLASS", and "How to Make a Spectral Cube" chapters.		
Revision 46	6 March 2015	
HIPE 13 release: updates to "How to run the HIFI Pipeline" chapter for the usage of the mkRms task and algorithm.		
Revision 47	1 June 2015	
Updates to "Flags in HIFI data" chapter, "Quality Flags" section.		
Revision 48	24 July 2015	
Updates to "Flags in HIFI data" chapter: "Quality Flags" section, "Sideband Deconvolution" chapter, and "How to make a spectral cube" chapter.		
Revision 49	27 July 2015	
Updates to the cookbook 'Spectral Scan Mode', section on Deconvolution.		
Revision 50	28 July 2015	
Updates to 'Sending HIFI spectra to VO tools' chapter.		
Revision 51	31 August 2015	
Updates to "Flags in HIFI data" chapter: "Quality Flags" section ; How to make a spectral cube chapter; Standing wave Removal chapter; HIFI Baseline Removal chapter.		
Revision 52	15 September 2015	
Updates to Standing wave Removal and HIFI Baseline Removal chapters - documenting new parameter 'addMedianContinuum' and updating the Electric Standing Wave Correction section.		
Revision 53	21 September 2015	
Addition of a new chapter: "Understanding the uncertainty table information in your data".		
Revision 54	26 October 2015	
Updating the pipeline flow diagrams, and minor fixes to some texts and tables.		
Revision 55	3 November 2015	
Updates to "Flags in HIFI data" chapter: "Quality Flags" section.		
Revision 56	18 November 2015	
Updates to "Understanding the uncertainty table information in your data", and to "Running the HIFI Pipeline" chapters.		
Revision 57	20 November 2015	
Updates to "The HIFI line identification tool" chapter.		
Revision 58	24 November 2015	
Small update to "The HIFI line identification tool" chapter.		
Revision 59	4 December 2015	

---

Added a warning to "Running the HIFI pipeline" chapter, section "How to run the HIFI pipeline".		
Revision 60	4 February 2016	
Minor updates to chapter "Tour of a HIFI ObservationContext", section "HIFI Calibration Data".		
Revision 61	29 February 2016	
On the behaviour of the SPUR_WARNING flag: minor updates to the chapter "Flags in the HIFI data", section "Channel flags" ; and to the chapter "HIFI Baseline Removal", section "Running FitBaseline".		

---

# Table of Contents

1. HIFI Launch Pad .....	1
1.1. Introduction .....	1
1.1.1. How to get your data .....	1
1.1.2. Looking at your data .....	3
1.1.3. Re-pipelining Observations .....	4
1.1.4. Common data reduction steps .....	5
1.1.5. Dealing with observing mode specific issues .....	6
1.1.6. Exporting data .....	6
2. HIFI Cookbooks .....	8
2.1. Introduction .....	8
2.2. Single Point Mode: Dual Beam Switch .....	8
2.2.1. Introduction .....	8
2.2.2. How Single Point Mode DBS observations are taken .....	9
2.2.3. Inspecting Single Point Mode DBS data .....	9
2.2.4. Single Point Mode DBS Data Reduction .....	11
2.3. Single Point Mode: Position Switch .....	16
2.3.1. Introduction .....	16
2.3.2. How Single Point Mode Position Switch observations are taken .....	17
2.3.3. Inspecting Single Point Mode Position Switch data .....	17
2.3.4. Single Point Mode Position Switch Data Reduction .....	19
2.4. Single Point Mode: Frequency Switch .....	27
2.4.1. Introduction .....	27
2.4.2. How Single Point Mode Frequency Switch observations are taken .....	27
2.4.3. Inspecting Single Point Mode Frequency Switch data .....	30
2.4.4. Single Point Mode Frequency Switch Data Reduction .....	30
2.5. Single Point Mode: Load Chop .....	37
2.5.1. Introduction .....	37
2.5.2. How Single Point Mode Load Chop observations are taken .....	38
2.5.3. Inspecting Single Point Mode Load Chop data .....	38
2.5.4. Single Point Mode Load Chop Data Reduction .....	41
2.6. Spectral Map Mode .....	50
2.6.1. Introduction .....	51
2.6.2. How Spectral Map Mode observations are taken .....	51
2.6.3. Inspecting Spectral Map Mode data .....	54
2.6.4. Spectral Map Mode Data Reduction .....	61
2.7. Spectral Scan Mode .....	64
2.7.1. Introduction .....	64
2.7.2. How Spectral Scan Mode observations are taken .....	64
2.7.3. Inspecting Spectral Scan Mode data .....	67
2.7.4. Spectral Scan Mode Data Reduction .....	69
3. Tour of a HIFI ObservationContext .....	76
3.1. Data Primer .....	76
3.1.1. Data frames .....	76
3.1.2. Data Products .....	76
3.1.3. Contexts .....	76
3.2. HIFI Science Data .....	77
3.3. HIFI Calibration Data .....	78
3.4. HIFI Browse Products .....	82
3.5. HIFI Auxiliary Data .....	83
3.6. HIFI Quality Context .....	87
3.7. HIFI TrendAnalysis Context .....	87
4. What was done to my data? .....	89
4.1. Introduction .....	89
4.2. Level 0 .....	90
4.3. Level 0.5 .....	93

4.4. Level 1 .....	94
4.5. Level 2 .....	96
4.6. Level 2.5 .....	98
5. Running the HIFI pipeline .....	100
5.1. Introduction to the Pipeline .....	100
5.2. How to run the HIFI Pipeline .....	101
5.3. How to process with new (or different) calibration data .....	105
5.4. Modifying the pipeline .....	108
5.4.1. Using the interactive Level 2.5 pipeline .....	110
5.4.2. Customising the Level 1 and 2 pipelines .....	113
5.4.3. Editing the pipeline algorithms .....	114
5.4.4. Running the Pipeline step by step .....	114
6. Viewing Spectra .....	116
6.1. Introduction .....	116
6.2. How to look at HIFI spectral data .....	116
6.2.1. Spectra .....	116
6.2.2. Spectral Cubes .....	118
6.2.3. HifiTimelineProducts (HTP) .....	119
6.3. Scripted plotting of spectral data with PlotXY .....	122
6.4. Scripted plotting of spectral data with SpectrumPlot .....	123
7. Converting positions in data to offsets .....	126
7.1. Introduction .....	126
7.2. Using the doOffset task .....	126
8. Understanding and using HIFI beam information in your data .....	128
8.1. Beam Metadata .....	128
8.2. Tools to obtain and use the HIFI beam model .....	129
9. Understanding the uncertainty table information in your data .....	130
9.1. Uncertainty model .....	130
9.2. Flux calibration uncertainty budget .....	130
10. Flags in HIFI data .....	132
10.1. Introduction to flags .....	132
10.2. Channel flags .....	132
10.3. Column rowflags .....	133
10.4. Quality Flags .....	135
11. How to add and remove flags .....	144
11.1. Introduction .....	144
11.2. How to understand what flags are in your data .....	144
11.3. Safe Usage of Flags .....	146
11.4. Setting and Clearing Flags with SpectrumExplorer .....	146
11.5. Setting flags interactively for many spectra .....	148
11.6. Setting and Clearing Flags with command line tools .....	157
11.7. Scripting Techniques for bulk clearing of flags .....	159
11.8. Scripting techniques for setting row flags .....	160
12. Standing Wave Removal .....	162
12.1. Introduction to Standing Wave Removal .....	162
12.2. Modified Passband Calibration Method .....	162
12.3. Sine Wave Fitting Method (fitHifiFringe) .....	162
12.3.1. Introduction to fitHifiFringe .....	162
12.3.2. Running fitHifiFringe .....	162
12.3.3. Example of using fitHifiFringe .....	167
12.4. Electrical Standing Wave correction in HEB bands .....	171
12.4.1. Introduction .....	171
12.4.2. Catalogue .....	171
12.4.3. Spline Model .....	172
12.4.4. Removal .....	174
12.4.5. Demonstration .....	177
13. HIFI Baseline Removal .....	182
13.1. Introduction .....	182



13.2. The FitHifiFringe Task .....	182
13.3. The FitBaseline Task .....	182
13.3.1. Introduction to FitBaseline .....	182
13.3.2. Running FitBaseline .....	183
13.3.3. Re-running FitBaseline .....	184
13.3.4. FitBaseline Options .....	185
13.3.5. Caveats .....	186
14. Sideband Deconvolution .....	188
14.1. Introduction to doDeconvolution .....	188
14.2. Basic strategy for running the deconvolution tool .....	190
14.3. Viewing deconvolution results .....	195
14.4. Exporting deconvolution results to Class .....	197
14.5. Advanced settings and diagnostic functions .....	197
14.5.1. Advanced methods .....	197
14.5.2. Diagnostic functions .....	199
15. How to make a spectral cube .....	204
15.1. Introduction to doGridding .....	204
15.2. doGridding Summary .....	204
15.3. Using doGridding... ..	209
15.3.1. ...to change beam, pixel, and map size .....	209
15.3.2. ...to make cubes of combined H- and V- polarisation .....	212
15.3.3. ...to make cubes for Solar System Objects .....	212
15.3.4. ...to make cubes more efficiently (limiting data input) .....	213
15.3.5. ...to make a rotated map or use a different WCS .....	214
15.3.6. ...with a different convolution .....	214
15.3.7. ...to specify the map centre .....	216
15.3.8. ...with selected data types .....	217
15.3.9. ...to deal with NaNs .....	218
15.4. doGridding in Detail .....	218
15.4.1. Particulars of Convolution .....	218
15.4.2. Using the Gridding task with the Spectrum Toolbox .....	220
16. Undoing the application of sideband gains .....	222
16.1. Introduction .....	222
16.2. Using undoSidebandGain .....	222
17. Mathematical Operations on Spectra .....	223
17.1. Introduction .....	223
17.2. Spectrum Toolbox HIFI Primer .....	223
18. Unit conversions .....	227
18.1. Converting to velocity and other frequency scales or frames .....	227
18.1.1. Changing frequency scale to USB, LSB, IF or velocity .....	227
18.1.2. Use of the Local Oscillator (LO) Frequency .....	228
18.1.3. Changing frequency rest frame with doVelocityCorrection .....	230
18.1.4. The meaning of velocities found in data and metadata .....	230
18.2. Flux conversions .....	231
19. Combining H- and V-polarisation Spectra .....	237
19.1. Introduction .....	237
19.2. Using the polarPairs task .....	237
20. Fitting Spectra .....	239
21. The HIFI line identification tool .....	240
21.1. Introduction .....	240
21.2. Basic Usage .....	240
21.3. Advanced Usage .....	244
21.4. A guided tour .....	245
21.5. The exportLines Task .....	250
21.6. Line assignment: the identifyLinesCatalog task .....	250
22. Making Publication quality plots .....	252
23. Exporting HIFI data to CLASS .....	256
23.1. Processing version from HIPE 12 onwards: direct FITS reading .....	256

---

23.2. Processing version earlier than HIPE 12: the hiClass task .....	257
23.2.1. Introduction to hiClass .....	257
23.2.2. hiClass examples .....	259
23.2.3. How to read HIFI data in CLASS .....	262
23.2.4. Exporting the results of deconvolution to Class .....	263
24. Sending HIFI spectra to VO tools .....	265
25. Reference Frames in HIFI data .....	268
25.1. Introduction .....	268
25.1.1. HSO Frame .....	268
25.1.2. SSBC Frame .....	268
25.1.3. LSR Frame .....	269
25.1.4. Source (nonSSO) Frame .....	269
25.1.5. Source (SSO) Frame .....	270
26. Relative performance of the HIFI spectrometers .....	271
27. Dealing with memory issues and slow performance. ....	272

## List of Figures

1.1. HSA Log-in and Herschel Science Archive User Interface from menu .....	1
1.2. Herschel Science Archive User Interface from button .....	2
1.3. Importing data into HIPE from a tar file .....	3
2.1. HIPE menu of useful scripts .....	8
2.2. DBS observation .....	9
2.3. Getting to the Level 2 spectra .....	10
2.4. The summary table .....	12
2.5. Emission in chop position .....	14
2.6. Emission correction for 1342190183 WBS-H-USB: (ON in green, OFF in red, and ON_corrected is blue) .....	15
2.7. Correcting for standing waves .....	16
2.8. Position Switch observation timeline .....	17
2.9. Summary .....	18
2.10. BrowseImageProduct .....	18
2.11. Level 1 data summary for HRS .....	20
2.12. Level 1 data summary for WBS .....	20
2.13. Level 1 spectra for HRS. The selected spectrum is from the last OFF of the first sequence of OFF-ON-ON-OFF. ....	21
2.14. Loading the Level 1 WBS product into the Spectrum Explorer .....	21
2.15. The default view of the Level 1 WBS product in Spectrum Explorer .....	22
2.16. A short tour of what is seen in the HIFI "selection panel". Note the time flow where subbands can be individually viewed. Double clicking the buttons toggles the spectrum to be visible or not. Double clicking "ALL" will show all the spectra (which at this stage will look quite the mess). ....	22
2.17. Pressing the "Grid" button will show thumbnail plots of all the spectra. Choosing the "Location" pull-down will place a "+" at the position of each integration, the "Raster" will show thumbnails at each position taken. ....	23
2.18. Level 2 WBS-H-USB SpectrumDataset. ....	23
2.19. Level 2 WBS-H-USB SpectrumDataset of 1342252113. Note the strong curvature within subbands. ....	25
2.20. Level 2 WBS-H-USB SpectrumDataset of 1342252113 after baseline correction. ....	25
2.21. Load Calibrated OFF for Obsid 1342252113 WBS-H-LSB. These will often display very strong standing waves. ....	26
2.22. Defringed OFF for Obsid 1342252113 WBS-H-LSB. No noticeable emission is present but platforming should be corrected. ....	27
2.23. Example of a FSW observation in Obsid 1342180473. Note the co-existence of positive and negative features belonging to the respective LO1 and LO2 tunings. Data are shown in an Upper Sideband (USB) scale. ....	28
2.24. Sketch illustrating the observing sequence considered in Frequency Switching with a Reference. Observing blocks are labeled as in the legend showed at the bottom right. ....	28
2.25. Same as Figure 2.23 for a Frequency Switching observation where no Reference position was taken - Obsid 1342200897. ....	29
2.26. Same as Figure 2.23 for a Frequency Switching observation where no Reference position was taken - Obsid 1342195094. ....	29
2.27. Same as Figure 2.23 for a Frequency Switching observation in band 6b (Obsid 1342180813), highlighting the presence of Electrical Standing Waves. ....	30
2.28. Observation context of a FSW observation, and summary table at Level 1 (only partial here). ....	31
2.29. Illustration of the relative positions of the respective LO1 (positive component) and LO2 (negative component) tuning lines for positive and negative frequency throws, and lines in either the LSB or USB. In all cases, the frequency scale assumed here is USB. Opposite direction will apply to data scaled in the LSB. ....	32
2.30. Line sideband assignment after the folding. Upper panel: USB scale spectrum. Lower panel: LSB scale spectrum. ....	33

2.31. Defringing correction in a FSW observation with no Reference (Obsid 1342248900). Four standing waves components are considered here. ....	34
2.32. Spectrum before and after correction. The relatively strong line (CO 5-4) here is masked in both its positive and negative phases (frequency throw of 94 MHz). ....	35
2.33. Baseline correction in Obsid 1342248900, after applying the defringing shown in Fig- ure 2.32. ....	36
2.34. Example of Reference Spectrum for Obsid 1342180473. No particular contamination is observed here. Note the poorer baseline quality compared to the ON-target double-difference spectrum from Figure 2.23. ....	37
2.35. Load Chop observations. ....	38
2.36. Summary ....	39
2.37. BrowseImageProduct ....	40
2.38. Level 0 MetaData ....	40
2.39. Level 2 HrsSpectrumDataset opened with Spectrum Explorer. ....	41
2.40. Level 1 data summary for HRS. ....	42
2.41. Level 1 data summary for WBS. ....	42
2.42. Level 1 HRS SpectrumDataset. ....	43
2.43. Level 2 WBS SpectrumDataset. ....	43
2.44. Smoothing width applied on the OFF spectra depending on the frequency. ....	44
2.45. Customise pipeline with Level 1 task mkOffSmooth. ....	45
2.46. OFF (sky reference) spectra. ....	45
2.47. FitHifiFringe task menu. ....	46
2.48. OFF (sky reference) spectrum output of the fitHifiFringe task. ....	46
2.49. Example of OFF contamination in 1342190778 (OFF spectrum in black, ON spectrum in red, ON-OFF in blue). ....	47
2.50. WBS-H-USB Level 2 spectrum. ....	48
2.51. Zoom on WBS-H-USB Level 2 spectrum. ....	49
2.52. FitHifiFringe task menu. ....	49
2.53. Zoom on WBS-H-USB Level 2 spectrum after fitHifiFringe. ....	49
2.54. WBS-H-USB Level 2 spectrum for a Load Chop observations with no reference. ....	50
2.55. Same spectrum after having applied fitHifiFringe with $n = 1$ . ....	50
2.56. Same spectrum after having applied fitHifiFringe with $n = 3$ . ....	50
2.57. Positions of read-outs of science data in an OTF (position switch) observation ....	53
2.58. Positions of read-outs of science data in an OTF (load chop) observation, taken at 90 degree position angle ....	53
2.59. OTF map 1342248770 viewed in Observation Viewer ....	55
2.60. A quick look at a spectral cube in the Spectrum Explorer ....	56
2.61. The spectra in the Level 2 HTP WBS-H-USB in the OTF observation plotted in Spec- trum Explorer, the extent of baseline drift in the observation can be seen. ....	58
2.62. The summary table for the WBS-H at Level 1 in the OTF observation. ....	59
2.63. The location option in Spectrum Explorer's raster mode for the WBS-H at Level 1. Hov- ering the mouse cursor over the three points to the bottom right (see inset), we see that these are the spectra in dataset 1 (rectangled in red). ....	59
2.64. The location option in Spectrum Explorer's raster mode for the WBS-H at Level 1, zooming in (twice) on the map region. ....	60
2.65. The summary table for the WBS-H at Level 1 in the DBS Raster observation. ....	61
2.66. Contamination due to emission in chop position in the DBS Raster observation 1342205481. ....	63
2.67. Example of WBS spectra collected over a mini-scan in band 1a between LO=492.7 GHz and LO=493.9 GHz (8 settings, Obsid 1342191505). Each colour corresponds to a different LO tuning. The sky frequency scale used here is the LSB one. Those lines falling at the same sky frequency at each tuning belong to the LSB (e.g. at 489.75 GHz), while those falling at different frequency at various tuning belong to the USB. ....	65
2.68. Sketch illustrating the observing sequence considered in Spectral Scans combined with DBS. Observing blocks are labeled as in the legend showed at the bottom right. ....	66
2.69. Same as in Figure 2.68 for Spectral Scans with a frequency grouping of 3 combined with DBS. The three shades of green are used here to represent the three different LO tunings com- bined within one single calibration block. ....	67

2.70. Observation context of a Spectral Scan, and summary table at Level 1 (here only partial) ...	67
2.71. Observation context of a Spectral Data, and example spectra from the Level 2 products. Note the residual standing wave in those data. ....	68
2.72. Level 2.5 deconvolved spectra for the WBS-H data in Obsid 1342191505. Note the residual standing wave resulting from imperfect data quality at the Level 2. Note also the gap between the lower and upper side band sky frequency ranges due to the limited LO tuning coverage. ....	69
2.73. Illustration of an Electrical Standing Wave in a Spectral Scan (Obsid 1342244537) at two different LO tunings. The data are here shown at Level 1 on an IF scale. As can be seen a continuum offset is usually also associated to the data distortion. ....	70
2.74. Output of the deconvolution at three levels of data cleaning in Obsid 1342190099. The top panel uses un-corrected data at Level 2, where spurs were still present and un-flagged (note the presence of negative ghosts as well). The middle panel uses data with spurs flagged but residual baseline structure still present. The lower panel uses data with the fringes and baselines corrected. ....	71
2.75. Example of LO settling time issues in part of a Spectral Scan in band 1a (Obsid 1342232978). This plot shows the collection of all WBS-H subband data collected at Level 2 on an USB scale. The settling time issues occur here at USB frequency around 542 GHz. ....	72
2.76. Spectral Ghost artefacts from the deconvolution of a strong line (standard Level 2.5 products from WBS-H in Obsid 1342215923). The strong 12CO line lies at 576.5 GHz (70 K) and injects negative signal in the deconvolution in the range 561-564 GHz. ....	73
2.77. See Figure 2.78 caption	74
2.78. Spectral Scan FSW data in Obsid 1342190186. The upper panel (Figure 2.77) shows part of the Level 2 spectra from the WBS-H in the USB scale. The water line at 557 GHz (from the LSB) can be seen on the lower end of the spectrum, together with the 12CO (5-4) seen both in the USB and LSB (upper end of the spectrum). The lower panel () shows the outcome of the deconvolution algorithm (on baseline-corrected Level 2 data) around the water and 12CO lines. Note the ghost features associated with the negative phase of the FSW and separated by the frequency throw. ....	74
2.79. Example of Reference Spectrum at a given LO tuning for Obsid 1342191505. Since the spectrum is made of a single difference of two OFF spectra taken at different chopper positions, the optical standing waves are not as optimally corrected as in a double-difference calibration. ....	75
4.1. Level 0 pipeline flow diagram	90
4.2. HRS pipeline flow diagram	93
4.3. WBS pipeline flow diagram	93
4.4. Level 1 pipeline flow diagram	94
4.5. Level 2 pipeline flow diagram	96
5.1. HIFI pipeline task: default view	101
5.2. Checking the version of the pipeline used at the HSA	105
5.3. Checking the calibration version used prior to HIPE 8 (top) and from HIPE 8 (bottom). ....	106
5.4. Using the Level 2.5 Interactive Pipeline GUI	111
6.1. Opening the Spectrum Explorer on a HIFI Level 2 spectrum	117
6.2. Opening the HifiTool in Spectrum Explorer on a HIFI Level 2 spectrum	118
6.3. Opening the Spectrum Explorer on a HIFI spectral cube	119
6.4. Opening the Spectrum Explorer on a HTP: comparing the Data Tree and Data Selection Panel	120
6.5. Using the location option of the Spectrum Explorer mosaic to see the positions of spectra in a map	122
6.6. Plot produced with SpectrumPlot	125
7.1. Absolute positions	127
7.2. Relative positions, in a coordinate system co-moving with the Solar-System target	127
9.1. Table containing the uncertainty model (values are in percentages) (for the V polarisation in this example)	130
9.2. Table containing the uncertainty budget (values are in percentages) (for the V polarisation in this example)	131
10.1. Row flags in a HIFI spectrum	135
10.2. Example of a Quality flag Report	136

11.1. To view channel flags in your spectrum, click on the <i>blue flag</i> icon (highlighted by the red oval) .....	144
11.2. To view channel flags in your spectrum by using a pointing device such as a touchpad or a mouse .....	145
11.3. The region flagged is colour-coded .....	145
11.4. Data point selection from the Spectrum Explorer button bar (highlighted in the red circle) .....	146
11.5. Selecting a region of the spectrum .....	147
11.6. Results from flagging the selected region .....	147
11.7. FlagTool task GUI .....	149
11.8. FlagTool datasets table and plot showing two flagged regions (coloured 'curtains') using two different flags .....	150
11.9. FlagTool messages in the console .....	151
11.10. FlagTool Linemasks table .....	152
11.11. flagTool using fitHifiFringe as an option .....	156
11.12. flagTool using fitBaseline as an option .....	156
11.13. The flagged subband is colour-coded in this example .....	159
12.1. FitHifiFringe GUI .....	163
12.2. Fitted sine waves, baseline, line mask, channel flags, and output data .....	164
12.3. Table containing information resulting from the fitting .....	165
12.4. Example of a typical period (at about 100 MHz) in a Load Chop observation in band 3A (taken without a sky reference) .....	167
12.5. Sine wave fitted using only one sine wave (i.e. nfringes=1) .....	168
12.6. Resulting fit showing the before and after spectra .....	169
12.7. Resulting fit using the optimum fitHifiFringe parameters .....	170
12.8. Example of a WBS HEB Catalogue .....	172
12.9. WBS Band 6a and 6 b (H and V) .....	173
12.10. WBS Band 7a and 7 b (H and V) .....	174
12.11. RDor Band 6a V-polarisation .....	174
12.12. Example of a spectrum with ESW (left panel), and a spectrum with no ESW (right panel) .....	175
12.13. Results after correction .....	175
12.14. Example of a spectrum with significant ESW (upper spectrum) and a spectrum with doubtful ESW (lower spectrum) .....	176
12.15. Results of the spectra going through the procedure .....	176
12.16. Results after correction .....	177
12.17. Uncorrected pipeline output .....	179
12.18. Final spectra after running the script .....	179
12.19. HebCorrection output table with <i>default</i> input parameters .....	180
12.20. HebCorrect output table with <i>exclude</i> parameter .....	180
13.1. FitBaseline GUI .....	183
13.2. Original and Residual plots produced by fitBaseline .....	183
13.3. Example of a Linemasks table .....	184
14.1. Folding of the upper and lower sidebands .....	188
14.2. Example of three different DSB LO settings being deconvolved .....	189
14.3. Dataset before cleanup .....	191
14.4. SSB result after deconvolution .....	191
14.5. Bad scans and channels stored in the SSB metadata .....	192
14.6. Deconvolution GUI .....	195
14.7. HIPE screenshot .....	196
14.8. Redundancy Histogram Plot .....	197
14.9. Observed vs. Modelled .....	200
14.10. Iterations progression .....	201
14.11. Chi-square vs. Iteration .....	202
15.1. The doGridding GUI .....	206
15.2. Gaussian Filter .....	219
18.1. Beam coupling computation. ....	234
21.1. IdentifyLines task GUI .....	241

21.2. The spectrum1d of Orion South in band 1a .....	241
21.3. Plot example for identifyLines .....	247
21.4. Plot example for identifyLines with specifying an index number .....	248
22.1. Plot example produced by the provided script .....	252
22.2. Same result as above but if plotting with Spectrum Explorer .....	255
23.1. HiClass task GUI .....	259

# Chapter 1. HIFI Launch Pad

Last updated 3 February, 2014.

## 1.1. Introduction

Welcome to the HIFI Launch Pad!

The Launch Pad is intended to get you off the ground and into HIFI data reduction quickly. We summarise how you get your data, quickly inspect it, and re-pipeline it (if needed) and point you toward information in the rest of the *HIFI Data Reduction Guide* and the *Herschel Data Analysis Guide* about common aspects of HIFI data reduction and analysis. More detailed information regarding specific observing modes can be found in the cookbooks in the following chapters.

If you are new to HIPE it is recommended that you also look at the [Quick Start Guide](#) and the [HIPE Owners Guide](#).

### 1.1.1. How to get your data

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

- You will need to log into the HSA using the "HSA Log-in" button in the bottom status bar of HIPE (see [Figure 1.1](#)) (for more details, see [Logging into the HSA](#))
- Herschel data are identified with a unique number known as the Observation ID (ObsID) (e.g. 1342212115)
- You can query the HSA from within HIPE using the *Data Access perspective*, from the Window → Show Perspectives menu (see [Figure 1.1](#)).
- You can also access the Herschel Science Archive by selecting (left-click) the HSA User Interface button (see [Figure 1.2](#)).
- HIPE expects the data in the form of a [pool](#). A pool is like a database, with observations organised in an Observation Context, containing links to all data, calibration, and supplementary files.

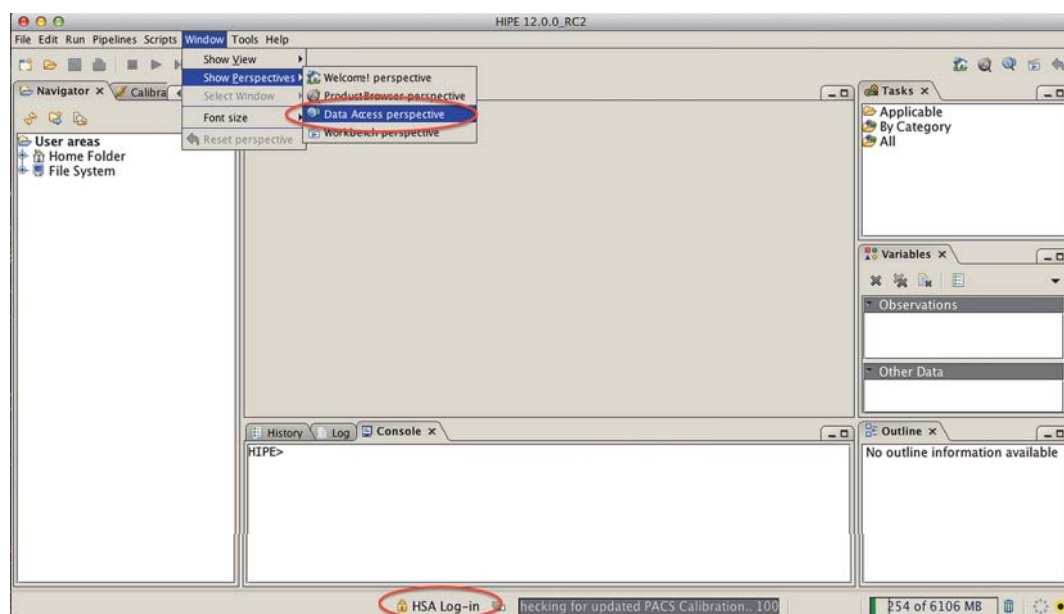


Figure 1.1. HSA Log-in and Herschel Science Archive User Interface from menu



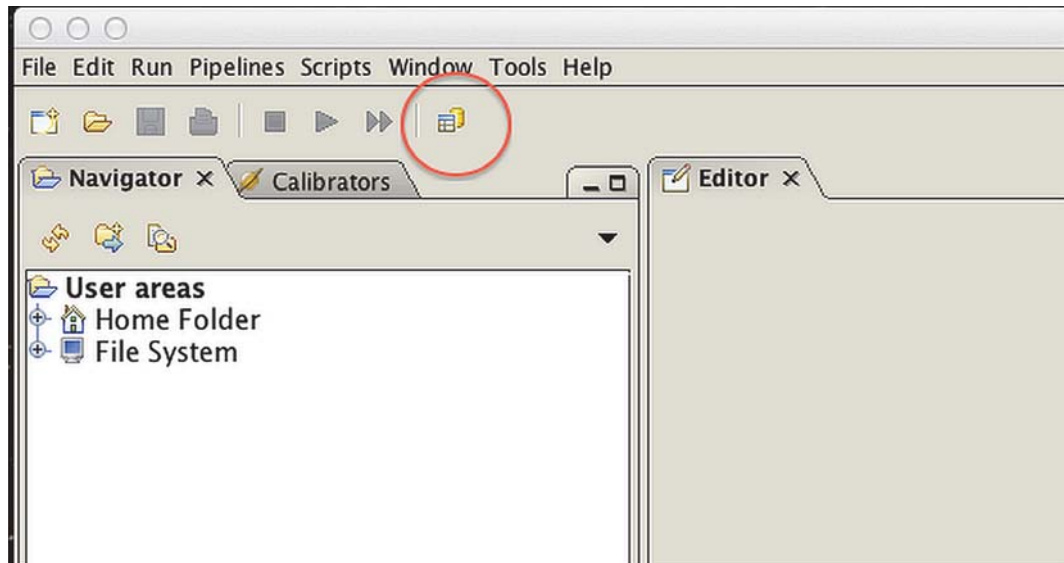


Figure 1.2. Herschel Science Archive User Interface from button

### ...from the HSA

There are several ways to get your data from the HSA, they are described in detail in the [first chapter](#) of the *Herschel Data Analysis Guide (DAG)* and summarised here.

1. To get multiple observations, download a tar file (not a pool) using the shopping basket (see details in the [DAG first chapter](#))
2. To get one observation you can directly access the HSA:
  - via the command line

```
obs=getObservation(ObsID, useHsa=True)
# example
obs=getObservation(1342212115, useHsa=True)
```

Note that if you are not logged into HSA, a login window will pop-up, allowing you to log in.

If you find that you have trouble obtaining the latest version of your observation it may be that you have it cached on your disk. In that case, try:

```
obs=getObservation(ObsID, useHsa=True, useCache=False)
```

- or you can use the “Send to external application” in the *HSA User Interface*.

If you directly access the HSA the data is not automatically saved to your disk and you should immediately save your observation with:

```
saveObservation(obs, poolName="myPool", saveCalTree=True)
```

### ...from disk

Once the data are saved on your hard disk, the Observation Context can be read into HIPE using:

```
myobs=getObservation(ObsID, poolName="myPool")
```

If you try to access this pool in the same HIPE session in which you saved it, you must update your list of local pools before HIPE will be able to find it. To do this, run the following command before running `getObservation`:

```
PoolManager.getInstance().propertyChange(new java.beans.PropertyChangeEvent \
(PoolCreatorFactory.getCreators().get("lstore"), \
LocalStoreFactory.LOCAL_POOL_DEFINITIONS_PROP, None, None))
```

Alternatively, you can press the *Refresh* button in *Storages and Pools* in the HIPE Preferences.

### ...from a tar file

If your data came from a tar file (multiple observations download), uncompress and untar the file, then, from HIPE view *Navigator* (see [HIPE Owners Guide](#) for more details), locate the folder on your disk, and open the content of the folder by a left-click on the plus sign. You will see folders associated to all your observations, an auxiliary folder, a calibration folder, and all your obsIDs with a small image on the left-side of the obsID's name. To import an obsID into HIPE, just double-left-click on the file with an image (see [Figure 1.3](#)). The *ObservationContext* of that obsID will then load in the HIPE view *Variables*, under *Observations*.

Note that it is also possible to uncompress and untar your file directly from the HIPE view *Navigator* - once you locate your tar file, just double-left-click on the *.tar.gz* file. A task called *decompress* will open and allow you to unpack your tar file.

Similarly to data accessed from HSA, you can save your observation in a *localPool* once it is unpacked:

```
# obsID located in the HIPE view Variables, under Observations
saveObservation(obsID, poolName="myPool")
```

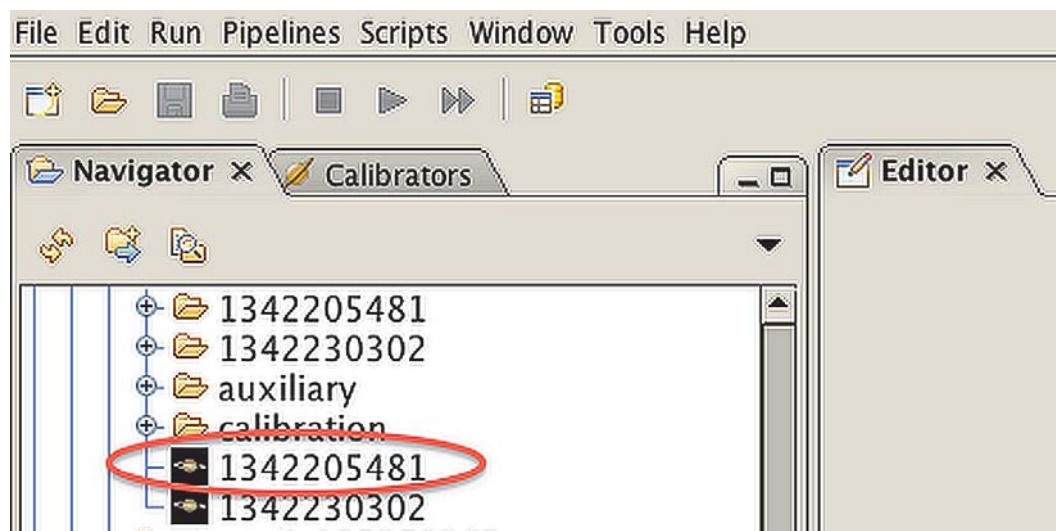


Figure 1.3. Importing data into HIPE from a tar file

## 1.1.2. Looking at your data

### ...A quick look

The best way to quickly inspect your data is to look at the *Browse Product* in the Observation Summary, which you can click on to enlarge. The browse products show:

- *Point Mode observations*: Two plots of Level 2 spectra with the H-polarisation to the left and the V-polarisation to the right. The upper and lower axes of the plots show the LSB and USB frequency

scale, respectively. The AOR label and observation number are used to title the plot. The observing mode, source name, requested RA and dec are below the plot title.

- *Mapping observations:* The map average spectra for the WBS-H (top) and WBS-V (bottom) is plotted per subband beside the integrated map of that subband. The upper and lower axes of the spectral plots show the LSB and USB frequency scale, respectively. The plots are arranged in order of increasing frequency. The integrated maps created with no correction done for any baseline issues. The x- and y- axes show the RA and declination, respectively, while the auxiliary axes show pixel coordinates. The colour scale used for the image is 'heat' and the intensity scale used is 'ramp' so the strongest emission in the map appears white.

The AOR label and observation number are used to title the browse product. The observing mode, source name, requested RA and dec are below the plot title.

- *Spectral Scans:* The single sideband solution after deconvolution of the Level 2 WBS spectra. No baseline correction has been done prior to deconvolution. The H-polarisation is shown to the left and the V-polarisation to the right. The AOR label and observation number are used to title the plot. The observing mode, source name, requested RA and dec are below the plot title.

### ...A deeper look

You can navigate through the Observation Context using the [ObservationViewer](#), see [Chapter 3](#) for more information about the contents of the Observation Context.

[SpectrumExplorer](#) is the main tool to plot and inspect spectra.

- It can be started by right mouse clicking on a [SpectrumDataset](#) or [product](#) (e.g. [HTP](#)) and selecting Open With → Spectrum Explorer. When viewing the contents of an HTP you must click in the box next to the variable name to plot the spectra.
- The selector panel appears at the bottom, with a row for each scan and a box (initially grey) for each subband. Clicking on each square will plot the spectrum for that subband, double-clicking will remove the plot.
- Tooltips describe the actions possible from the button bar above the plot.

## 1.1.3. Re-pipelining Observations

### When should I reprocess my data?

The HIFI ICC recommends that your data should ideally be processed using the HIPE version that you are using for data analysis to ensure that you do not run into data-software compatibility problems. However, it is not anticipated that you will have problems if you keep within one HIPE version, e.g. perform data analysis in HIPE 13 on data that was pipelined with HIPE 12. In some cases we may recommend that certain observing modes are processed with a particular HIPE version, see the [HIFI Instrument and Calibration page](#).

### How do I reprocess my data?

To reprocess (re-run the pipeline) data you can find `hifiPipeline` in the *Applicable Tasks* menu when you click on an Observation Context in the *Variables View*. Select the spectrometers and the levels between which you want to reprocess for and hit accept. Or, in the command line:

```
# Create a new Observation Context called newobs by reprocessing all levels of the
# Observation Context "obs"
newobs = hifiPipeline(obs = obs, save=False)
#
# To just reprocess from Level 1 to Level 2, and only for the HRS
newobs = hifiPipeline(obs = obs, fromLevel=1, upToLevel=2, apids = ['HRS-H', 'HRS-
V'], save=False)
```

See [Chapter 5](#) for more details.

### How do I use newer calibration data?

To re-pipeline data using the latest calibration version available from the HSA, rather than the calibration data that was shipped with your data, you must first configure the pipeline to go to the HSA for the calibration data. To do so, first run the `configureHifiPipeline` task:

```
calconfig = configureHifiPipeline(useHsa=True)
```

You can tell the pipeline to use the newer calibration by checking the `cal` box in the GUI and by passing the variable `calconfig` to the `palStore` bullet, before running the pipeline.

In the command line this is all done as follows:

```
calconfig = configureHifiPipeline(useHsa=True)
obs_1 = hifiPipeline(obs=obs, cal=1, palStore=calconfig)
```

See [Section 5.3](#) for more details.

## 1.1.4. Common data reduction steps

### 1.1.4.1. Correcting for Standing Waves

- Standing waves can be fitted with sine waves and subtracted using `FitHifiFringe`, which appears as an applicable task when one clicks on the Observation Context variable.
- On the command line, it is typically done as follows:

```
fitHifiFringe(data=obs, nfringes=2, product='WBS-V-LSB')
```

- See [Chapter 12](#) for more details.

### 1.1.4.2. Correcting for Baseline Drifts

- Baselines can be easily removed using the `fitBaseline` task. To create a new observation context with baseline corrected Level 2 products, use:

```
obs_fit = fitBaseline(data=obs)
```

and follow the instructions that appear in the console. For more information, see [Section 13.3](#)

### 1.1.4.3. Spectrum Arithmetics and Manipulation

When a spectrum is selected in the *Variables View*, HIPE becomes the *spectrum toolbox*; a range of tools for performing spectral arithmetics and manipulation become automatically available.

- View a spectrum in Spectrum Explorer and click on the crossed hammer and spanner icon in the toolbar above, the toolbox will appear to the right of the plot. Select the task you wish to perform from the drop-down menu and complete the GUI form.
- The spectrum tasks available in the toolbox can also be used in scripts or in the command line. The simplest way to find the syntax is to copy the command from the console after using the GUI. Here, we give a couple of examples.

```
#
# First extract a spectrum from the Observation Context, for example the Level 2
# WBS-H-USB
spectrum=obs.refs["level2"].product.refs["WBS-H-
USB"].product.refs["box_001"].product["0001"]
#
#Stitch subbands together:
StitchedSpectrum = stitch(ds=spectrum, variant="crossoverPoints",
    edgeTolerance=0.01, stepsize=0.0)
#
#extract a range in frequency (the ranges are given in the units in the spectrum)
ExtractedSpectrum = extract(ds=spectrum, ranges=[(555.0, 558.7)])
```

The use of all the tasks is described generally in [Spectrum Toolbox](#) of the *Herschel Data Analysis Guide* where you will find more information. Some HIFI-specific examples can also be found in [Chapter 17](#).

### 1.1.4.4. Fitting spectra

Spectral features can be fitted and removed using the [SpectrumFitterGUI](#), which is available from the Spectrum Explorer. The tool can be used to interactively fit a single spectrum, or in automatic mode on multiple spectra using a previously defined model.

- To interactively fit a spectrum, extract the region of interest or stitch all the subbands of the data together so that you have only one segment of data (this is one box in the Spectrum Explorer selector panel).

## 1.1.5. Dealing with observing mode specific issues

Interactive tasks and data issues that are particular to given observing modes are described in a series of cookbooks in [Chapter 2](#):

- [Section 2.2](#): Single Point Mode: Dual Beam Switch
- [Section 2.3](#): Single Point Mode: Position Switch (included are some discussion on Platforming and Off-Emission)
- [Section 2.4](#): Single Point Mode: Frequency Switch
- [Section 2.5](#): Single Point Mode: Load Chop
- [Section 2.6](#): Spectral Map Mode
- [Section 2.7](#): Spectral Scan Mode

## 1.1.6. Exporting data

### 1.1.6.1. Exporting Data to CLASS

- HIFI data can be exported to CLASS readable FITS files using the `HiClass` task. You may wish to stitch subbands together before exporting to CLASS.

```
# Export one dataset to a FITS file:
HiClassTask()(data = myspectra, fileName = 'myspectra.fits')
#
# Export one HIFI timeline product to a FITS file:
HiClassTask()(data = myhttp, fileName = 'myhttp.fits')
# Export the Level 2 spectra to a FITS file by supplying the ObservationContxt:
HiClassTask()(data = obs, fileName = 'myl2spectra.fits')
```

The FITS file is written in the directory you started HIPE from unless you specify a path.

- To read the FITS file into CLASS:

```
file out MyHIFISpectra.hifi mul
fits read MyHIFISpectra.fits
#
# Now you have a CLASS file named MyHIFISpectra.hifi (you can use whatever you
# want as an
# extension) that you can access like you always do in CLASS:
#
file in MyHIFISpectra.hifi
find
get first
set unit f i
device image white
plot
```

- Note that you need to use a recent version of CLASS, i.e. the version from April 2010 onwards.
- See [Chapter 23](#) for detailed information and examples on how to use the task.

### 1.1.6.2. Exporting Data to FITS

Use the [simpleFitsWriter](#) to save spectra to FITS file.

```
simpleFitsWriter(product=spectrum, file='spectrum.fits')
```

### 1.1.6.3. Exporting Data to ASCII

Use the [exportSpectrumToAscii](#) task available from the Spectrum Toolbox to save spectra to text file.

```
exportSpectrumToAscii(ds=spectrum, file='spectrum.txt')
```

# Chapter 2. HIFI Cookbooks

Last updated: 2 June, 2015

## 2.1. Introduction

The following cookbooks summarise important information on usage of all aspects of HIFI data acquisition and reduction for all observing modes. You will find a summary of how observations were carried out for a given mode, an overview of the Observation Context, from calibration to Level 2.5, and a description of the data reduction workflow including post-processing recipes to e.g. inspect emission in the OFF positions.

Although much care was taken to provide complete details for each mode, it is worth consulting all cookbooks in case some details were not repeated in every cookbooks.

To complement the cookbooks, you have access to some useful scripts (containing detailed comments) via HIPE that will guide you in writing your own scripts, and teach you more in-depth methods of data reduction for specific tasks e.g. artifacts cleaning, HEB correction etc... (see [Figure 2.1](#)). You can save these scripts, and adapt them to your need.

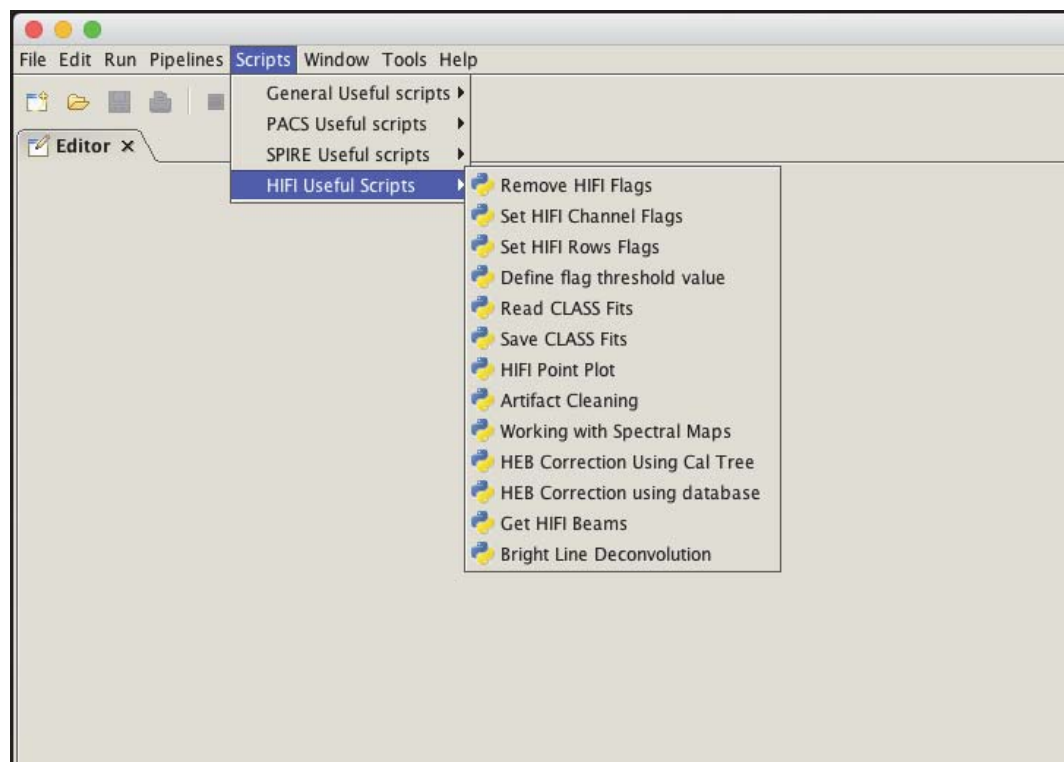


Figure 2.1. HIPE menu of useful scripts

## 2.2. Single Point Mode: Dual Beam Switch

Last updated: 1 April, 2015

### 2.2.1. Introduction

The most common problems faced when working with DBS mode observations are dealing with emission in chop positions and hard to remove (electronic) standing waves in band 6 and 7.

The observation used here (obsid=1342190183) is a FastChop DBS CO 5-4 observation of LDN 1157, therefore you can follow the script exactly with the data used in it. This cookbook provides more in depth information than that given in comments in the script and also screen shots to reassure you that you are doing the right thing.

To learn about removing baselines, fitting to lines, combining H and V spectra, and other data analysis steps that are not observing mode specific, you are referred to the remainder of the HIFI Data Reduction Guide.

## 2.2.2. How Single Point Mode DBS observations are taken

### 2.2.2.1. Observing Principles

Dual Beam Switch (DBS) observations are taken by repeatedly chopping 3' between the science target and a sky reference position (nod 1), then slewing the telescope 3' to the chop position just used and repeatedly chopping between the science target and a second chop position (nod 2). See the [Figure 2.2](#) for a schematic representation. This sequence is repeated as many times as needed to attain the requested noise.

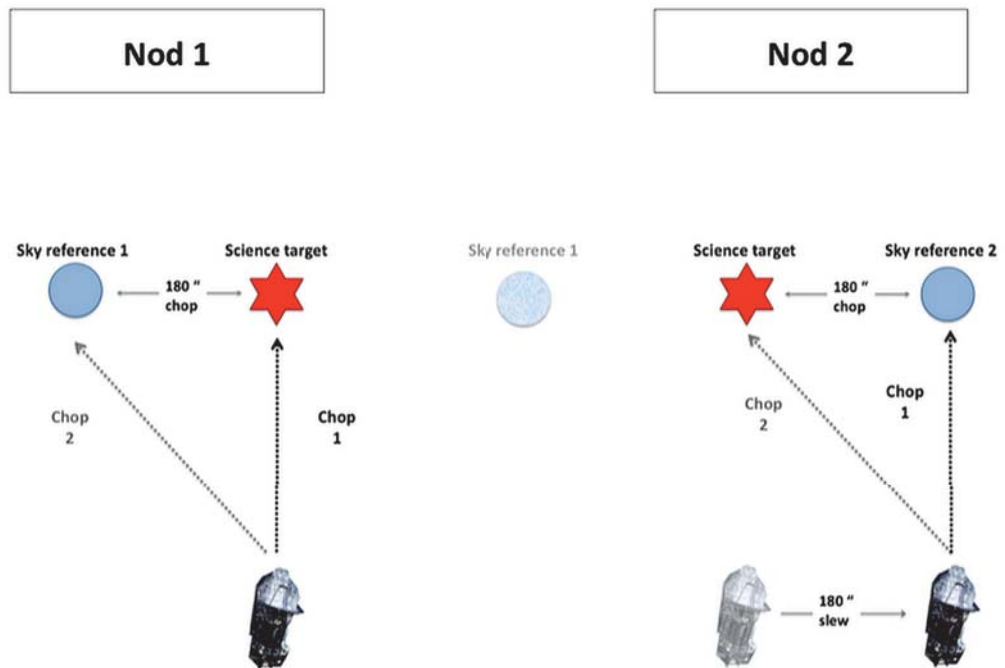


Figure 2.2. DBS observation

## 2.2.3. Inspecting Single Point Mode DBS data

Get your observation into HIPE, here we download a Fast Chop DBS observation from the HSA.

```
obs=getObservation(1342190183, useHsa=True)
```

The variable you just created using the `getObservation` command, `obs`, is not actually the data but a set of references to the data in the HSA. There is a reference for every product (e.g., Level



2.5, Level 2, Level 1, calibration. HifiTimelineProduct, SpectrumDatasets, see [Chapter 3](#) for more information about the contents of a HIFI ObservationContext) and the first time you access a product it will be downloaded from the HSA. This makes working slow and susceptible to lost connections; thus, it is strongly recommended to save the observation to a pool on your local disk and then work from that.

```
# Immediately save observation rather than work from HSA.
# Save in a pool with the obsid used as the pool name, save the
# calibration tree too, in case you want to re-pipeline the observation

saveObservation(obs, saveCalTree=True)
```

This command will save the observation into a pool called 1342190183 (the Observation Number).

### 2.2.3.1. Level 2 (and 2.5) Spectra

In the Level 2.5 of all point mode observations, HTP are stitched, folded (if Frequency Switch) and converted to simpleSpectrum format. HRS spectra are stitched together only if the subbands overlap in frequency. The mkRms task is run on the data to calculate the rms noise, the output is stored in the *Trend Analysis* product. In the remainder of this section, we concentrate on the Level 2 data.

The chop position spectra are subtracted from the science target spectra, using an averaged spectra of nod 1 and nod 2. The resulting target spectra are then averaged together and converted to sky frequency and to the  $T_A^*$  temperature scale to create the final Level 2 spectra.

In the Level 2 product there will be HifiTimelineProducts (HTP) for each of the spectrometers selected to be used in the observation. In most cases this will be both horizontal and vertical polarisations of the WBS and HRS, and there are separate products for the USB and LSB frequency scales.

To look at the Level 2 spectra, right click on the obs variable and open it with the *Observation Viewer* (unless you have previously selected a different viewer for an Observation Context in this HIPE session, the *Observation Viewer* will be the default viewer opened on a double click). Click your way through the tree until you reach the spectrum of interest.

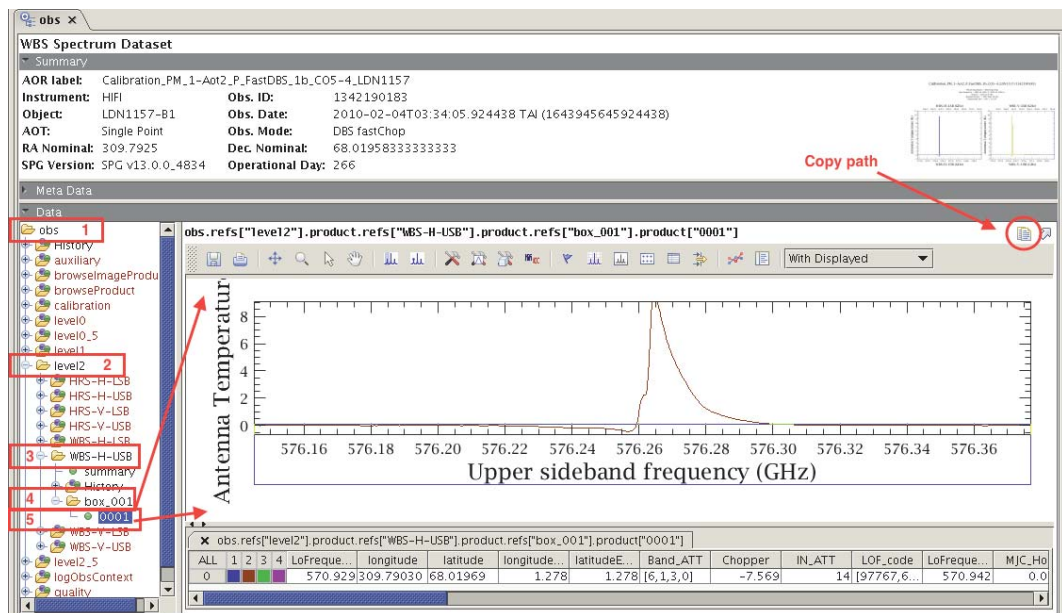


Figure 2.3. Getting to the Level 2 spectra

In [Figure 2.3](#) we illustrate opening the Level 2 WBS-H-USB spectrum. There are several points of note:

- Step four involves clicking open a box product which can contain up to 100 spectra (by default, but you can customise the `doCleanUp` step of the Level 2 pipeline to have different size boxes) that are grouped together to increase download time from the HSA.
- In step five we open a product called 0001, this is the Level 2 spectrum. Spectra in boxes are numbered from (000)1-(0)100, starting at 0001. DBS observations result in one Level 2 spectrum for each spectrometer-polarisation-sideband combination.

When you (single) click on the spectrum it will appear in the editor panel beside the Observation Context tree, with the reference to the product displayed above the plot. You can pan along the axes of this spectrum and zoom in on a box drawn with the mouse it after selecting Tools → Zoom upon right clicking on the plot. If you have Zoom selected as the initial tool in the Spectrum Explorer preferences Edit → Preferences → SpectrumExplorer you can zoom in directly. If you double click on the spectrum then Spectrum Explorer takes over the entire *Editor View*. You can navigate the spectrum as above but you also have access to all of the tasks in the Spectrum Toolbox.

To create a variable of the Level 2 spectrum that you can pass to tasks, you can drag the highlighted spectrum product (0001) in the Observation Context tree into the *Variables View*; in this example, a variable called `obs_level2_WBS_H_USB_box_001_0001` will be automatically be created, you can rename it by selecting the Rename option on right click on the variable name.

The reference can be copied using the *copy path* button (see [Figure 2.3](#)) and then pasted into a script or the console. For example, to extract the Level 2 WBS-H-USB spectrum from the ObservationContext in the command line you can use the following construct:

```
spectrum_wbs_h_u = obs.refs["level2"].product.refs["WBS-H-
USB"].product.refs["box_001"].product["0001"]
```

## 2.2.4. Single Point Mode DBS Data Reduction

### Can I trust the continuum?

DBS observations can produce very stable baselines, particularly in fast chop mode; however, unless you selected continuum optimisation in HSpot, the continuum is not guaranteed to be accurate.

### 2.2.4.1. Level 1 Spectra

Looking at your Level 1 data can help to identify problems with your observation that affect the quality of your Level 2 data. For example, in [Figure 2.3](#) above, an absorption feature can be seen at ~576.25-576.26 GHz; as we shall see, inspection of the Level 1 spectra shows that this is seen only in one nod of the observation and so is due to emission in one of the chop positions.

Level 1 data contains calibration observations in addition to science data, the science data are frequency calibrated but still in the IF scale and with  $T_A$  temperature scale. The best way to identify which are the science data is to use the *summary table*, which can be found in each HTP. You can double click on the summary table to open it with the *Dataset Viewer* and you can also extract it from the Observation Context in the same way you can a spectrum:

```
WBS_H_L1_SummaryTable=obs.refs["level1"].product.refs["WBS-H"].product["summary"]
```

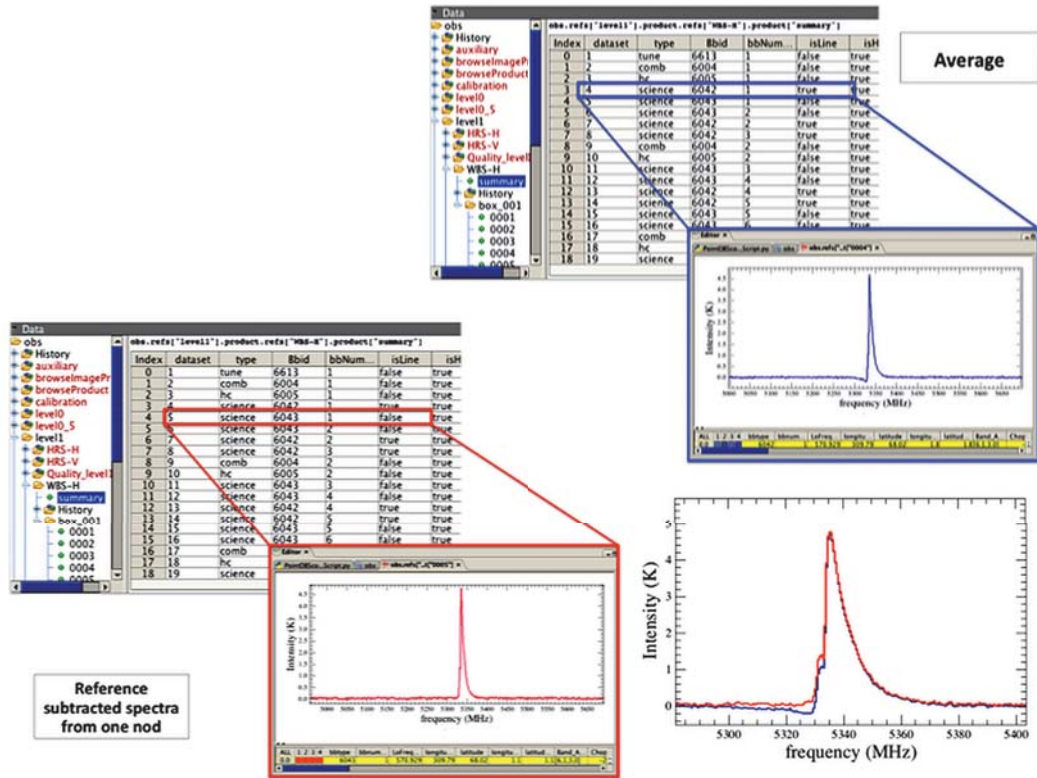


Figure 2.4. The summary table

The science data are labelled “science” in the *type* column of the summary table. At Level 1, we store the average of the reference subtracted chop positions in datasets identified by the column with *isLine* set to True. There is also science data with *isLine* “false”, and this is the reference subtracted spectra of one of the nod positions.

In [Figure 2.4](#), we show the summary table for the WBS-H where we can see that the first set of averaged nod positions are in dataset 4, while the first set of reference subtracted spectra for one of the nod positions is in dataset 5. We can also see, from the *bbNumber* in the fifth column, that the entire observation consists of six repeats of the nod cycles, and mid-way through the observation another set of calibrations – a comb and a hot-cold load measurement – were taken.

Clicking on one of the science datasets will open the spectrum up in the Editor view to the right as before, and we can see that there are seven sets of spectra in each dataset. It can be useful to look at each individual spectrum if you are concerned about drifts in the observation but here we will average all the spectra for each set of science data.

First, extract the HTP. This step is not necessary but makes the commands in the following steps easier to follow.

```
htp_wbs_h_L1 = obs.refs["level1"].product.refs["WBS-H"].product
```

At Level 1, the data are not all precisely on the same frequency grid, in particular the start and end frequencies of the spectra may differ slightly. This will cause the `selectHifi` task to give a warning that:

```
Not all the datasets have exactly the same shape - some need to be cropped so that they will fit all in one dataset.
```

This is not harmful but can be avoided by resampling the data in the HTP to a common grid. This is normally done by the `doFrequencyGrid` step of the Level 2 pipeline. A function can be written

to resample all of the WBS spectra to the same grid, a grid spacing of 0.5 MHz is used in the Level 2 pipeline and that is selected here.

```
def resampleHtp(htp):
    # This routine resamples all WBS spectra in a HTP to a common grid.
    # This makes it easier to perform spectral arithmetic on spectra calibrated
    # with different COMB measurements, since the frequencies will be now be aligned.
    #
    # Step 1: define a grid of WBS frequencies in MHz at 0.5 MHz spacing
    # note : WBS has 4 subbands, so we need 4 arrays.
    #       : the subbands start at roughly 4000, 5000, 6000, and 7000 MHz
    #       : the subbands are roughly a GHz wide
    sbgrid=0.5*Double1d.range(2000)
    grid = [4000.0 + sbgrid,5000.0+sbgrid, 6000.0+sbgrid,7000.0+sbgrid]
    #
    # Step 2: inspect summary table to get the index of all data, then loop through
    them
    for htpindex in htp["summary"]["dataset"].data :
        spectrum= htp.get(htpindex) # fetch the spectrum from the HTP
        spectrumType=spectrum.meta["sds_type"].value # determine type (COMB, HC,
        SCIENCE, TUNE)
        spectrum_resampled = resample(ds=spectrum, density=True, grid=grid) # resample
        htp.set(htpindex,spectrum_resampled,spectrumType) # reinsert spectra to the HTP
    #
    return htp # return the modified HTP
```

In order to resample the data in an HTP, it can now be passed to this function:

```
htp_wbs_h_L1 = resampleHtp(htp_wbs_h_L1)
```

The first time you run the above line, you will see the following message several times:

```
Resolution parameter could not be adjusted since the wave scale
unit is incompatible with the unit the resolution is expressed in.
```

It is harmless and can be ignored.

Now we want to find the averages of the Level 1 science spectra. This can be done by selecting the spectra based on their *Bbid* (Building block ID). For a DBS fast chop observation, nod 1 has a Bbid of 6042, and nod 2 a Bbid of 6043, while the science phases of a DBS slow chop observation are labelled with 6031 and 6032. By the time Level 1 of a DBS chop observation is reached, Bbid 6042 spectra have morphed into the average of the two reference subtracted chop spectra, while Bbid 6043 represent the reference subtracted spectra at one nod position.

We use the `selectHifi` task and set the `return_single_ds` option to true in order to return a `SpectrumDataset` that the `avg` task can be used on.

```
bbid_6042_select = \
selectHifi(htp=wbs_h_L1, selection = {"bbtype":[6042]}, return_single_ds=True)
bbid_6042_av = avg(ds=bbid_6042_select)

bbid_6043_select = \
selectHifi(htp=wbs_h_L1, selection = {"bbtype":[6043]}, return_single_ds=True)
bbid_6043_av = avg(ds=bbid_6043_select)
```

The average of the mean reference subtracted chop position spectra (in blue) is shown overlaid on the average of the reference subtracted spectra from one nod position (in red) in the lower right hand corner of [Figure 2.4](#). We can see that the averaged reference subtracted spectrum from one nod position is free from contamination, whilst the dip before the line is still seen in the averaged reference subtracted chop spectra. Therefore, the contamination must come from emission in only one chop position (the one not seen).

To overlay the two spectra, plot `nod1_av` in SpectrumExplorer and then, drag the `nod2_av` variable into the plot.

### 2.2.4.2. Comparing Chop Position Spectra

The chop position contamination we see with the observation we use in this example is quite clear-cut but there may be occasions when you wish to directly examine the spectra in the chop positions. Of course, the chop position spectra contain emission from the sky, which can make interpretation difficult, so instead we remove the contribution from the sky by looking at the difference in the chop spectra.

The difference in chop position (also called the OFF) is calculated by the pipeline for all modes. You can find the OFF positions for all spectrometers of the observation in the Calibration product in the Observation Context; choose calibration → pipeline-out → ReferenceSpectra.

The OFF data is processed up to an equivalent Level 2, and have USB and LSB products. On fixed and moving targets, there is one single OFF spectrum per backend/sideband (the sum of all available OFF from the Level 1).

The OFF positions spectra can be run through the standard tasks such as `fitHifiFringe`, `fitBaseline`, and `flagTool`, and will also be corrected from Electrical Standing Waves by default.

In [Figure 2.5](#) we show the differenced chop spectra for the WBS-H-USB in this observation as well as the location of this spectrum in the Observation Context. We note the presence 1) of possible contamination in the chop position, and 2) of optical standing waves that can be corrected (see [Section 2.2.4.4](#)).

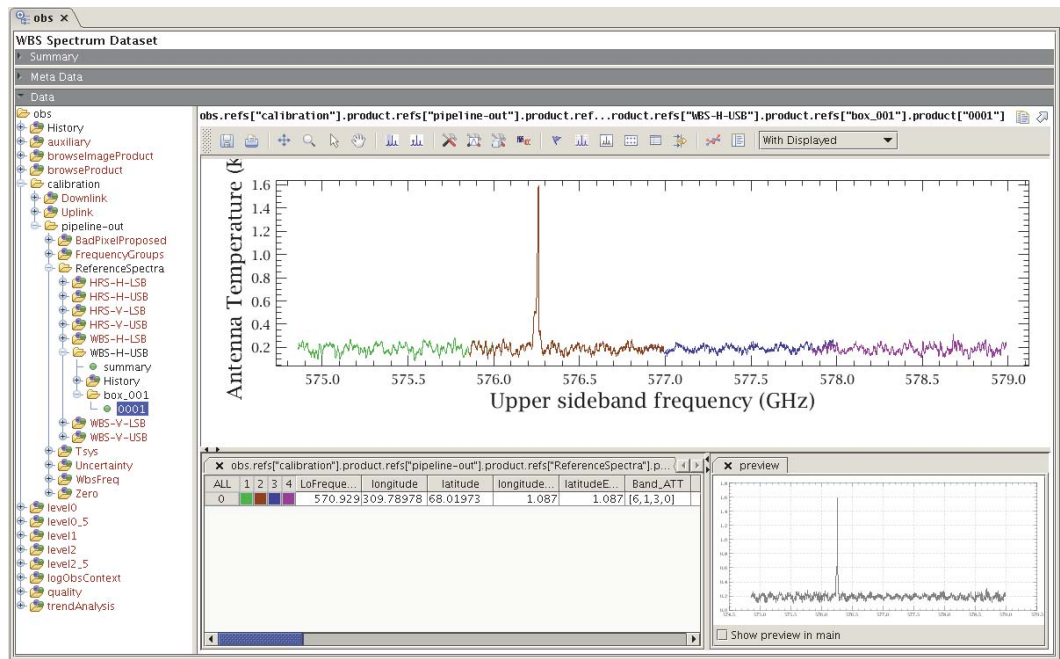


Figure 2.5. Emission in chop position

### 2.2.4.3. Correcting for emission in OFF positions

We can correct the contaminated chop position by subtracting (or adding) the average difference in the two chop positions from each of the affected chop spectra in the HTP.

You should apply an addition or a subtraction depending on whether the contamination in the processed OFF (which corresponds to OFF1-OFF2) was seen as a positive or negative contribution.

```
# In this example, we use obsid=1342190183 WBS-H-USB

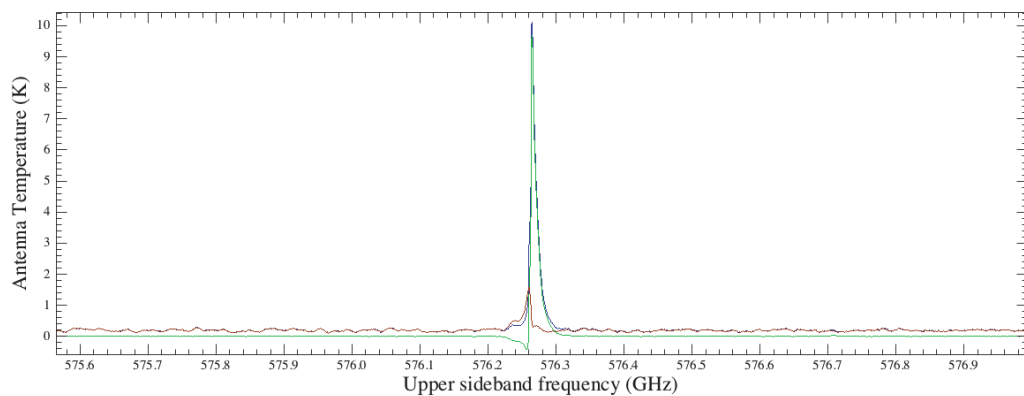
OFF = obs.refs["calibration"].product.refs["pipeline-
out"].product.refs["ReferenceSpectra"].product.refs["WBS-H-USB"] \
.product.refs["box_001"].product["0001"]

ON = obs.refs["level2"].product.refs["WBS-H-
USB"].product.refs["box_001"].product["0001"]

# To correct for contamination if the contamination is in the OFF1 (positive
contribution)

ON_corrected = add(ds1=ON,ds2=OFF)
```

Figure [Figure 2.6](#) shows the result of our correction: the ON spectrum is shown in green, the OFF spectrum is shown in red, and the ON<sub>corrected</sub> spectrum is shown in blue.



**Figure 2.6. Emission correction for 1342190183 WBS-H-USB: (ON in green, OFF in red, and ON<sub>corrected</sub> is blue)**

```
# To correct for contamination if the contamination is in the OFF2 (negative
contribution)

ON_corrected = subtract(ds1=ON,ds2=OFF)
```

#### 2.2.4.4. Correcting for standing waves

Standing waves are not a particular problem for DBS observations in bands 1-5. They can be corrected using the `fitHifiFringe` task (which is described in [Chapter 12](#)) to remove sine-waves from the data. Here, however, we briefly illustrate the importance of considering more than one sine wave fit to the data.

The observation in this cookbook shows a low-level standing wave that, upon first look, appears to have a period of about 150 MHz. The default settings of `fitHifiFringe` are to fit the WBS-H, and to fit one sine wave with a typical period of 150 MHz. Inspection of the Chi-squared plot produced when the task is run with default settings shows the presence of at least three standing waves, see [Figure 2.7](#). Accordingly, we correct the data in this way.

```
CorrectedData = fitHifiFringe(obs_or_htp=obs, nfringes=3)
```

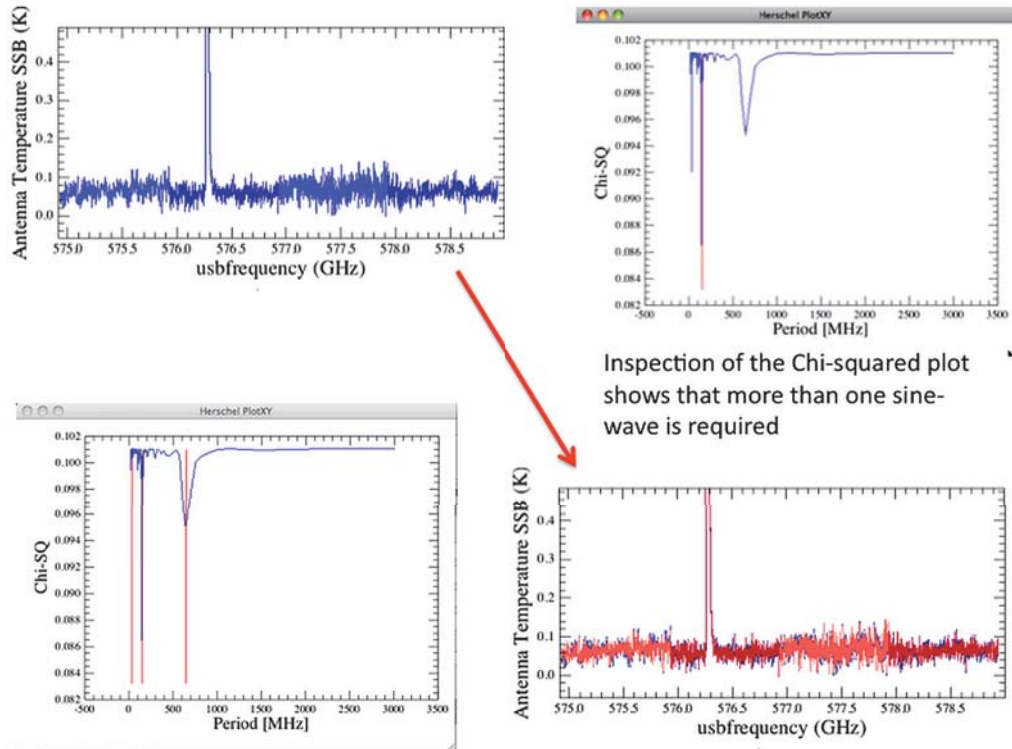


Figure 2.7. Correcting for standing waves

In the case of DBS observations in bands 6 and 7, electronic standing waves (ESWs) are found and can be difficult to satisfactorily correct by subtraction of sine-waves. If you have band 6 or 7 DBS data that are badly affected by standing waves, we recommend that you first consult [Section 12.4](#) to see how to treat them. The method presented in this section may allow you to correct the ESWs present in your data. If, you are still unable to correct your data, we recommended that you contact the Helpdesk to ask for assistance.

## 2.3. Single Point Mode: Position Switch

Last updated: 10 March, 2015

### 2.3.1. Introduction

This is the simplest observing mode for HIFI. This mode uses the internal loads for flux calibration and an external source reference to remove the emission from the telescope and any standing waves present. The reference position is within 2 degrees of the source and should be free from emission. The internal sources (hot and cold loads) are obtained via the chopping mechanism, the source and reference positions are obtained by slewing the telescope.

The observations used here are Position Switch observation of L1557-b2 (obsid=1342190836) in band 1b set to simultaneously observe the 557 (GHz) water line (in lower sideband) and the 572.49 GHz ammonia line (in the upper sideband). Also used in this chapter is another observation (obsid=1342270533) in band 1b of the ground state ortho water transition, and a band 2a observation of the CO 6-5 transition (obsid=1342252113).

## 2.3.2. How Single Point Mode Position Switch observations are taken

### 2.3.2.1. Observing Principles

Position Switch observations are taken by first integrating on the reference (OFF) position, then slewing the satellite to the source position (ON), and integrating. This sequence is repeated in reverse order (ON then OFF) to make a OFF-ON-ON-OFF pattern. The offset between the ON and the OFF can be at most 2 degrees. Aside from the integration time lost on the OFF position, slewing the satellite costs time, making Position Switch observations less time efficient than some of the other modes.

### 2.3.2.2. Observing Timeline

[Figure 2.8](#) illustrates the different observing blocks involved in a Position Switch observation. Before the first OFF position integrations, the LO is tuned to the desired frequency (cyan box), a comb and dark are taken for frequency calibration of WBS (yellow box), and cold and hot load sequences are taken (red and blue boxes). These internal calibrations may be repeated during the slews at later times during the observation. The green boxes indicate integrations on the sky either at the source position (ON), or on the reference position (OFF).

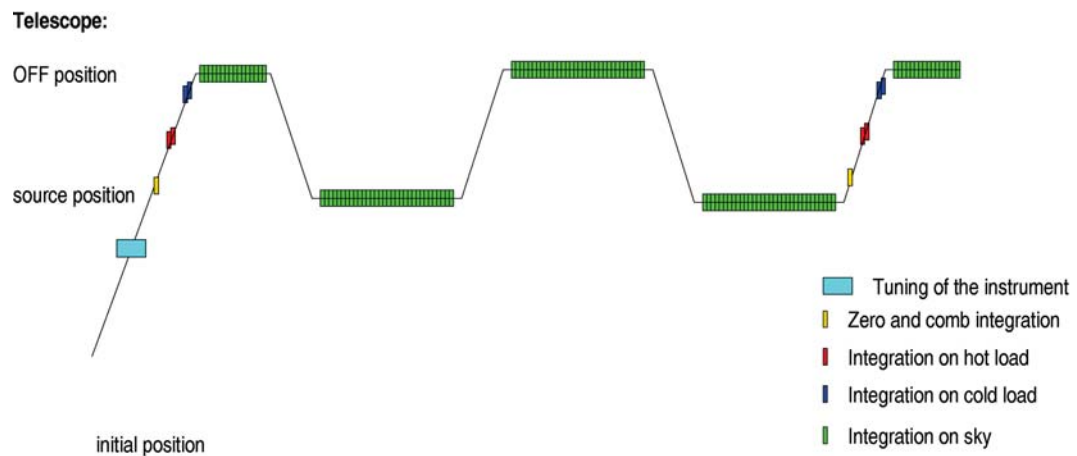


Figure 2.8. Position Switch observation timeline

## 2.3.3. Inspecting Single Point Mode Position Switch data

Get your data into HIPE, here we download the band 1b observation from the HSA.

```
obs=getObservation(1342190836, useHsa=True)
```

The variable you just created using the `getObservation` command, `obs`, is not actually the data but a set of references to the data in the HSA. There is a reference for every product (e.g., Level 2.5, Level 2, Level 1, calibration) which are displayed on the editor window when double-clicking on the `obs` variable. See [Chapter 3](#) for more information about the contents of a HIFI `ObservationContext`.

The first time you access a product it will be downloaded from the HSA. This makes working slow and susceptible to lost connections; thus, it is strongly recommended that you save the observation to a pool on your local disk, and then work from that.



```
# Immediately save observation rather than work from HSA. Save in a pool with the
# obsid used as
# the pool name, save the calibration tree too, in case you want to re-pipeline the
# observation.
#
saveObservation(obs,saveCalTree = True)
#
# This command will save the observation into a pool called 1342190836 (Obsid).
# To recall these data in any future HIPE sessions, use the getObservation command
# again:
#
obs=getObservation(1342190836)
#
#This reads the data directly from the disk where you saved it.
```

### 2.3.3.1. Summary and MetaData Tables

The *Summary* shows you the headlines of the observation, including the nominal coordinates and the HIPE version (*SPG*) that has been used to process these data:

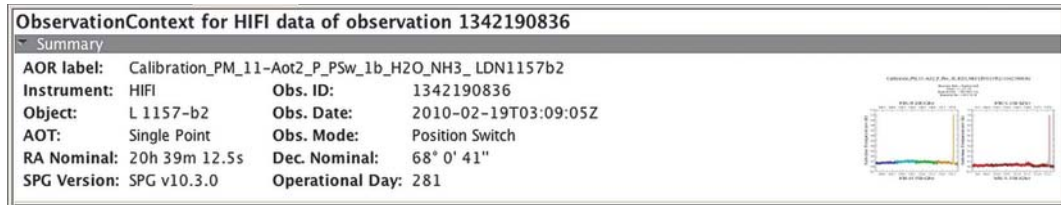


Figure 2.9. Summary

A browse product giving an overview of the WBS spectra in both polarisations is shown in the right-hand side. This image can also be seen by clicking on the *browseImageProduct* in the *Data* window (indicated here with a red oval), and corresponds to the post-card displayed in the HSA for this particular observation:

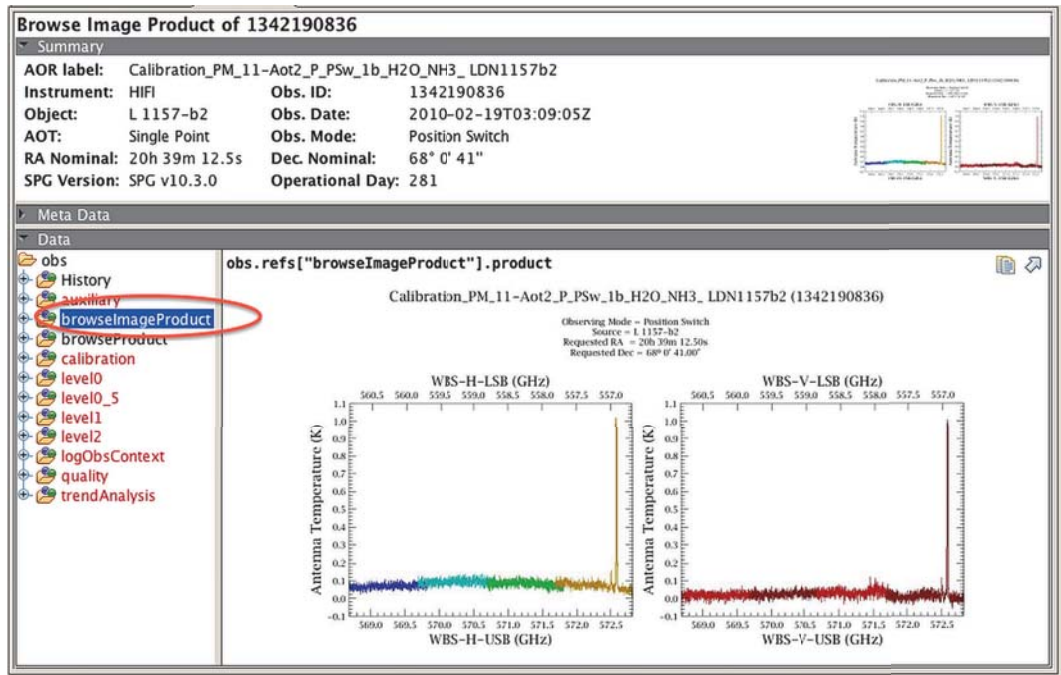


Figure 2.10. BrowseImageProduct

These pictures are automatically generated from the Level 2 product. You should then verify in the *Meta Data* that the observed coordinates (*ra*, *dec*) do not differ (within the pointing accuracy) from

the requested ones (*raNominal*, *decNominal*). The requested (RA,DEC) can be found in the metadata of any Level 0 product:

The small script below allows you to calculate the differences between requested RA and DEC and the actual RA and DEC.

```
# Requested Coordinates: put the metaData values into variable
#
raNominal=obs.getMeta()["raNominal"].value
decNominal=obs.getMeta()["decNominal"].value
#
# Observed Coordinates
#
ra=obs.getMeta()["ra"].value
dec=obs.getMeta()["dec"].value
#
print (decNominal-dec)*3600.0
print (raNominal-ra)*COS(decNominal/180.0*3.14159)*3600.0
```

Hence, the difference between observed and requested is (0.0489", -0.457") arcseconds. To assess how good that is, you should check the pointing accuracy of the satellite. The *Absolute Pointing Error (APE)* is not the same all over the mission - see this following [document](#) for more information.

The difference in positions in this observation are well within the APE for OD281 which had a pointing accuracy of 2".

## 2.3.4. Single Point Mode Position Switch Data Reduction

### 2.3.4.1. Data structure

#### Level 1 data

[Figure 2.11](#) and [Figure 2.12](#) illustrate the *summary* tables of this Position Switch observation at the Level 1, where calibration and OFF observations are still present. The *summary* table indicates the time flow of how an observation was taken. Each activity (tune, comb, hot/cold, OFF or ON integration) is given a unique **bbtype** number. For Position Switch the OFF integrations are labeled with bbtype 6021 and the ON integrations with 6022. Products are present for HRS and WBS backends for both polarisations.

The LO tuning (*tune*) takes place right before the calibration block, and has a fixed duration that depends on the frequency. Tuning indeed implies some thermal stabilisation of the LO chain so that a dead time, allowing for this, is allocated into the tuning block. Right after in this summary table, one can recognise the sequence illustrated in [Figure 2.8](#): the calibration block, composed of a *comb* (WBS frequency calibration) and a *hc* (code for “Hot-Cold”, for the bandpass and flux calibration), is followed by the OFF- and ON-target blocks in the order given in [Section 2.3.2.2](#). Note that Level 1 data for ON-target Bbid already have the OFF position integrations subtracted.

The sequence of the observation with HRS backend is listed in the table shown in [Figure 2.11](#). The structure is the same for the WBS data (see [Figure 2.12](#)) but of course without the *comb*.

**SummaryTable**

Summary

AOR label: Calibration\_PM\_11-Aot2\_P\_P5w\_1b\_H2O\_NH3\_LDN1157b2  
 Instrument: HIFI Obs. ID: 1342190836  
 Object: L 1157-b2 Obs. Date: 2010-02-19T03:09:05Z  
 AOT: Single Point Obs. Mode: Position Switch  
 RA Nominal: 20h 39m 12.5s Dec. Nominal: 68° 0' 41"  
 SPG Version: SPG v10.3.0 Operational Day: 281

Meta Data

Data

psw1b.refs["Level1"].product.refs["HRS-H"].product["summary"]

Index	dataset	type	Bbid	bbNumber	isLine	isHrs	isWbs	fullName	LoFrequency [GHz]	LO-Throw [GHz]	start	length
0	1	tune	6601	1	false	false	false	HRS_tune_block_aot	564.77025	0.0	0	2
1	2	hc	6005	1	false	true	true	HIFI_Calibrate_lot_cold	564.77025	0.0	2	2
2	3	science	6021	1	false	true	true	HIFIContOffIntegration	564.77025	0.0	4	12
3	4	science	6022	1	true	true	true	HIFIContOnIntegration	564.77025	0.0	16	12
4	5	science	6022	2	true	true	true	HIFIContOnIntegration	564.77025	0.0	28	12
5	6	science	6021	2	false	true	true	HIFIContOffIntegration	564.77025	0.0	40	12
6	7	science	6021	3	false	true	true	HIFIContOffIntegration	564.77025	0.0	52	12
7	8	science	6022	3	true	true	true	HIFIContOnIntegration	564.77025	0.0	64	12
8	9	science	6022	4	true	true	true	HIFIContOnIntegration	564.77025	0.0	76	12
9	10	science	6021	4	false	true	true	HIFIContOffIntegration	564.77025	0.0	88	12
10	11	science	6021	5	false	true	true	HIFIContOffIntegration	564.77025	0.0	100	12
11	12	science	6022	5	true	true	true	HIFIContOnIntegration	564.77025	0.0	112	12

Figure 2.11. Level 1 data summary for HRS

**SummaryTable**

Summary

AOR label: Calibration\_PM\_11-Aot2\_P\_P5w\_1t\_H2O\_NH3\_LDN1157b2  
 Instrument: HIFI Obs. ID: 1342190836  
 Object: L 1157-b2 Obs. Date: 2010-02-19T03:09:05Z  
 AOT: Single Point Obs. Mode: Position Switch  
 RA Nominal: 20h 39m 12.5s Dec. Nominal: 68° 0' 41"  
 SPG Version: SPG v10.3.0 Operational Day: 281

Meta Data

Data

psw1b.refs["Level1"].product.refs["WBS-H"].product["summary"]

Index	dataset	type	Bbid	bbNumber	isLine	isHrs	isWbs	fullName	LoFrequency [GHz]	LO-Throw [GHz]	start	length
0	1	tune	6613	1	false	true	true	WBS_attenuators_block	564.77025	0.0	0	3
1	2	comb	6004	1	false	true	true	WBS_Zero_Comb	564.77025	0.0	3	2
2	3	hc	6005	1	false	true	true	HIFI_Calibrate_hot_cold	564.77025	0.0	5	2
3	4	science	6021	1	false	true	true	HIFIContOffIntegration	564.77025	0.0	7	12
4	5	science	6022	1	true	true	true	HIFIContOnIntegration	564.77025	0.0	19	12
5	6	science	6022	2	true	true	true	HIFIContOnIntegration	564.77025	0.0	31	12
6	7	science	6021	2	false	true	true	HIFIContOffIntegration	564.77025	0.0	43	12
7	8	science	6021	3	false	true	true	HIFIContOffIntegration	564.77025	0.0	55	12
8	9	science	6022	3	true	true	true	HIFIContOnIntegration	564.77025	0.0	67	12
9	10	science	6022	4	true	true	true	HIFIContOnIntegration	564.77025	0.0	79	12
10	11	science	6021	4	false	true	true	HIFIContOffIntegration	564.77025	0.0	91	12
11	12	science	6021	5	false	true	true	HIFIContOffIntegration	564.77025	0.0	103	12
12	13	science	6022	5	true	true	true	HIFIContOnIntegration	564.77025	0.0	115	12

Figure 2.12. Level 1 data summary for WBS

Figure 2.13 shows the content of the Level 1 HRS-H Dataset. Each line represents a *dataset block* (from 1 to 11) which itself is made of several *SpectrumDataset*. Each dataset block corresponds to a type of event (tuning, hot-cold calibration, Off Integration, On Integration). For instance, the dataset 6 is a block of 12 *SpectrumDatasets* (from 0 to 11) corresponding to the OFF integration on the reference position.

Note that the frequency scale is 'Intermediate Frequency' (IF) at this stage.

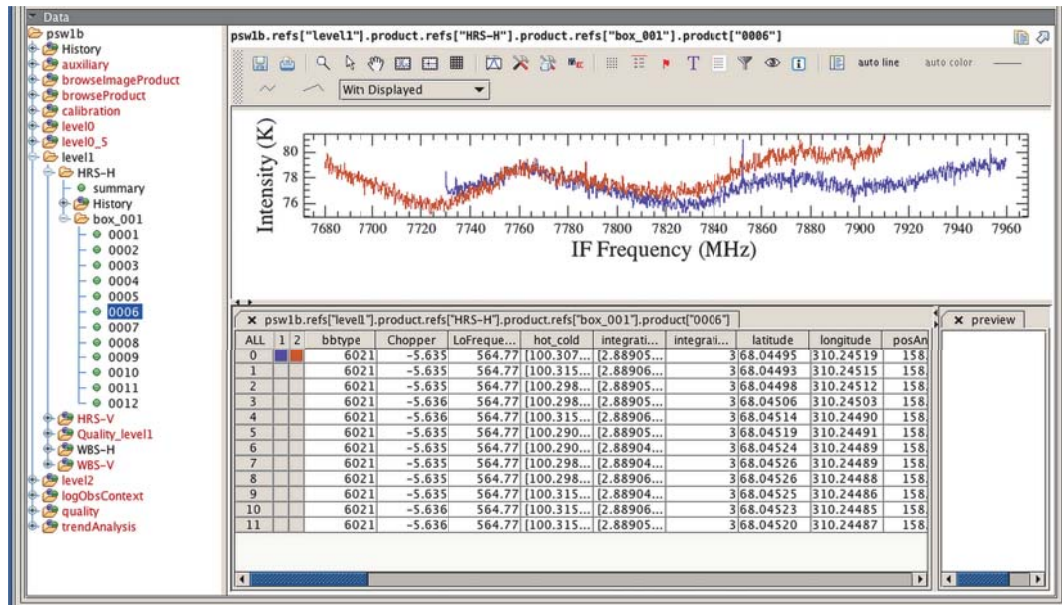


Figure 2.13. Level 1 spectra for HRS. The selected spectrum is from the last OFF of the first sequence of OFF-ON-ON-OFF.

Viewed from the Spectrum Explorer, it is possible to see the spectra of the various steps of the observation. For this section, the WBS-H product will be used, but everything that is shown here holds for the other backends and polarisations as well.

Figure 2.14 to Figure 2.17 demonstrate how to open the HIFI product in the Spectrum Explorer, and view various integrations as well as the position on the sky of the integrations.

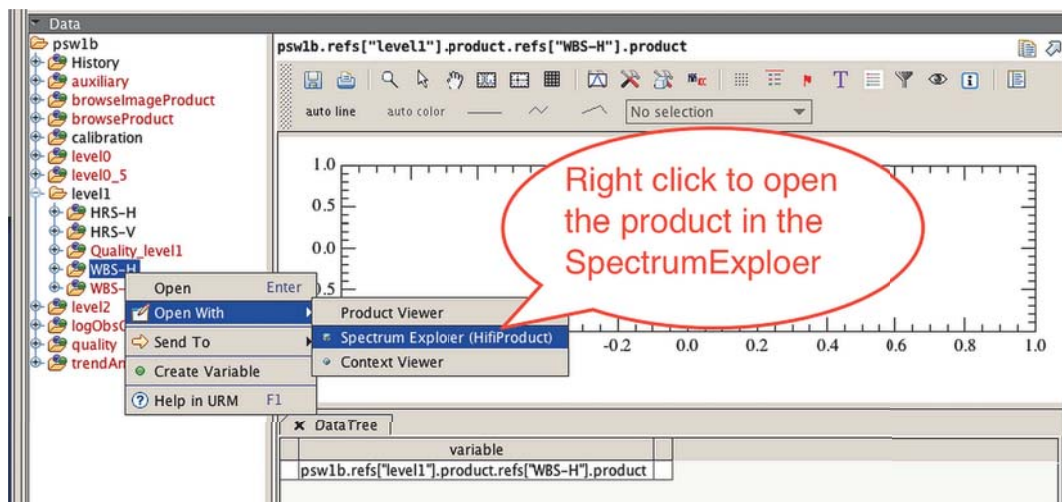


Figure 2.14. Loading the Level 1 WBS product into the Spectrum Explorer

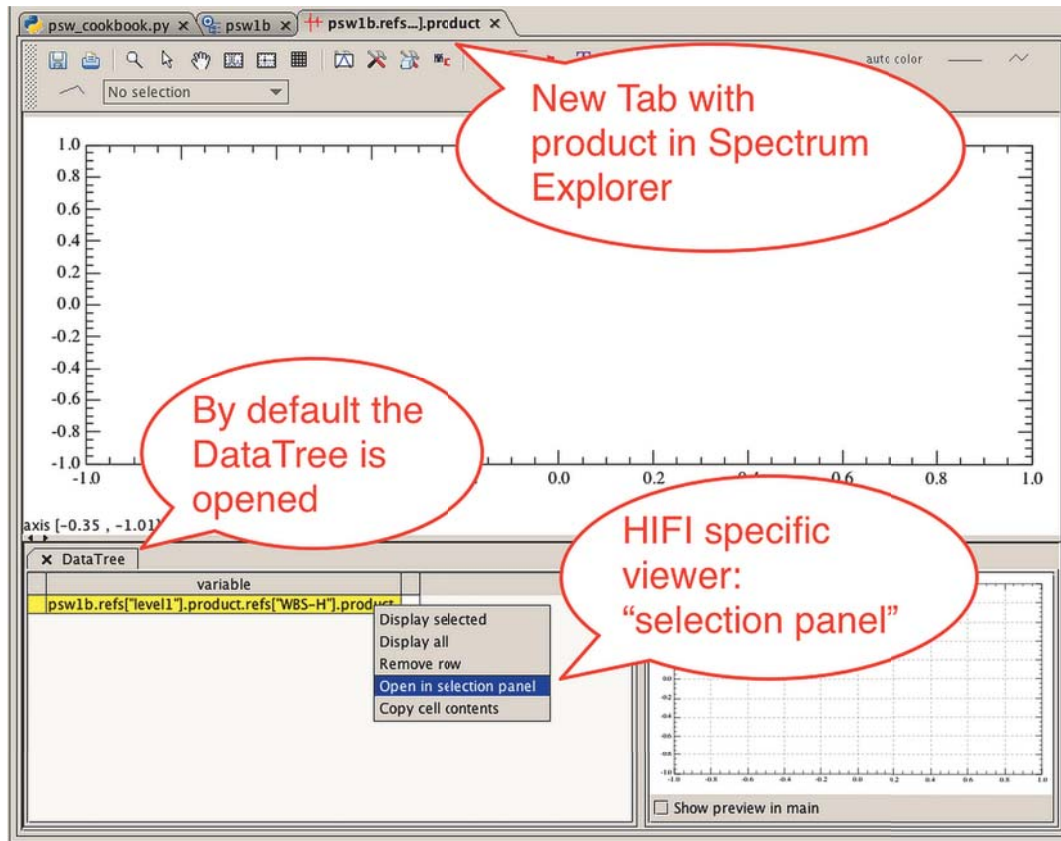


Figure 2.15. The default view of the Level 1 WBS product in Spectrum Explorer

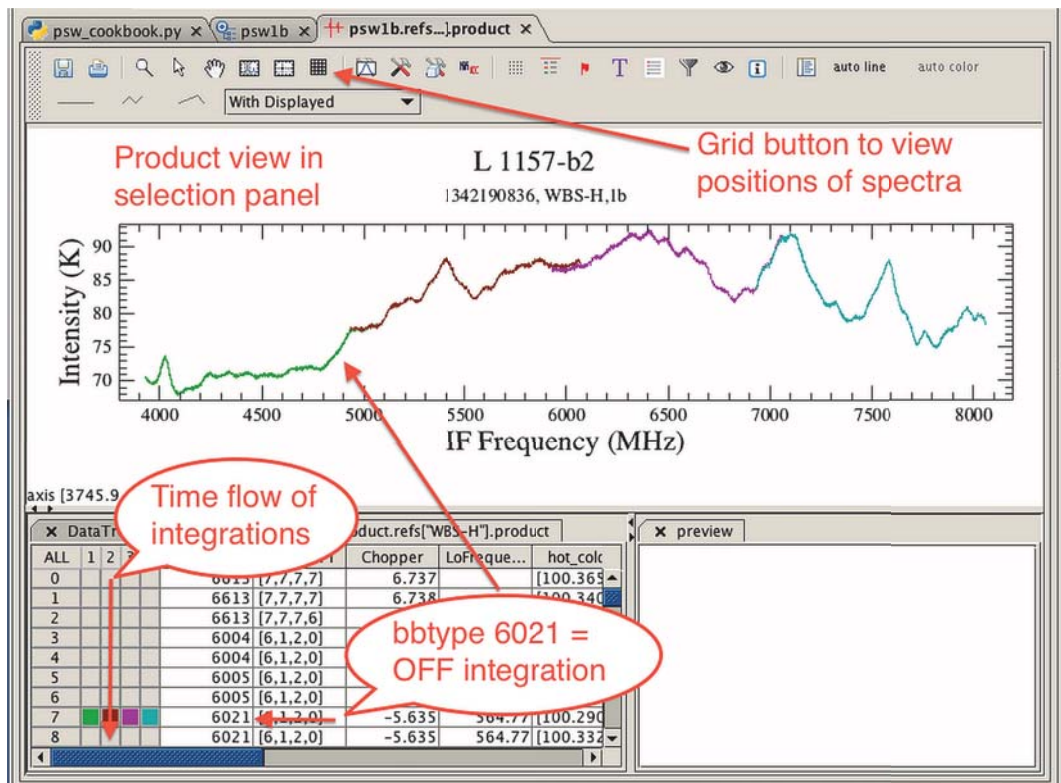


Figure 2.16. A short tour of what is seen in the HIFI "selection panel". Note the time flow where subbands can be individually viewed. Double clicking the buttons toggles the spectrum to be visible or not. Double clicking "ALL" will show all the spectra (which at this stage will look quite the mess).

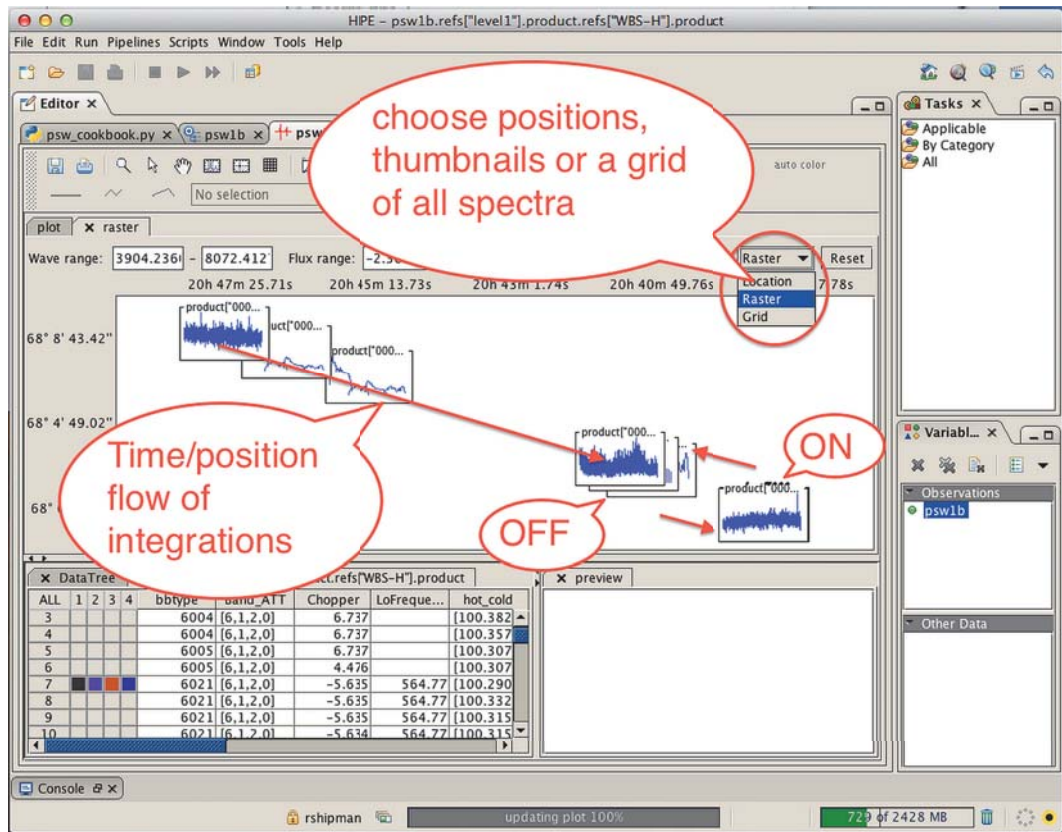


Figure 2.17. Pressing the "Grid" button will show thumbnail plots of all the spectra. Choosing the "Location" pulldown will place a "+" at the position of each integration, the "Raster" will show thumbnails at each position taken.

## Level 2

At Level 2, only ON-target spectra are maintained and there is one spectrum (*HifiTimelineProducts*, HTP) per backend/sideband/polarisation. Click your way through the tree until you reach the spectrum of interest.

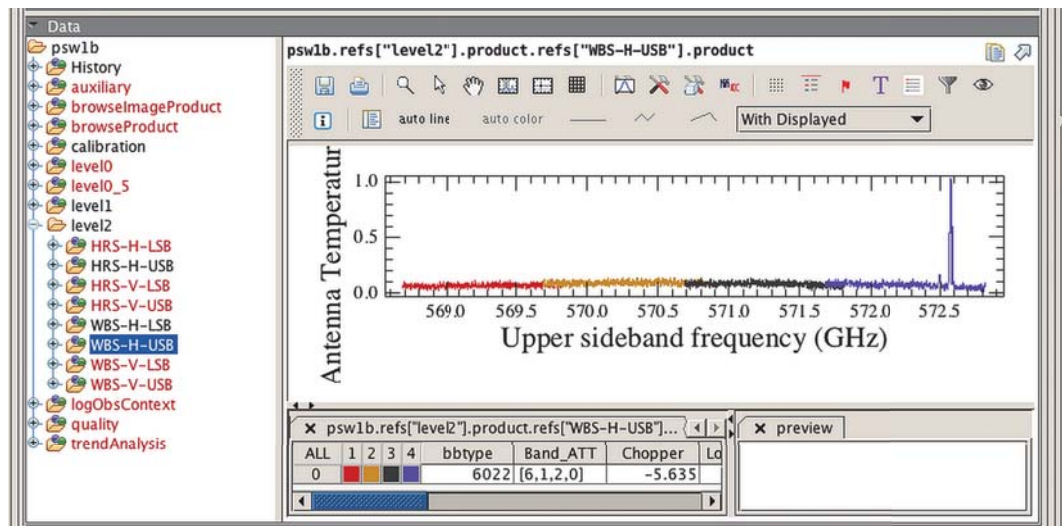


Figure 2.18. Level 2 WBS-H-USB SpectrumDataset.

Note that the spectrum is in  $T_A^*$  temperature scale. Careful inspection of Figure 2.18 shows a slight curvature and mismatches between WBS subbands. This is an example of "platforming" and the effect can be much worse, depending on the observation.

## 2.3.4.2. Data artifacts and data cleaning

The main data problems one can find in Position Switch observations are:

- **Baseline jumps and curvature (Platforming):**

- Platforming is seen as offsets between neighbouring WBS subbands and/or curvatures within subbands. Platforming can be addressed using the standard `fitBaseline` task (see [Chapter 13](#)) and by applying a low order polynomial to each WBS subband. Note that **platforming** is an artifact of the WBS. If you correct your spectra with `fitBaseline` specifically for platforming, you should not apply the same correction to the HRS data. Also be aware that if there was a significant continuum in the WBS data, it will be removed by `fitBaseline`.

- **Standing Waves:**

- Optical standing waves can be present in the data, with typical periods depending on the band used (see this [report](#) for more details). Those baseline modulations are usually enhanced in the presence of non-negligible continuum, and can become really severe on planet observations. For these standing waves, the `fitHifiFringe` task usually does a pretty good job (see [Chapter 12](#)).
- Additionally, bands 6 and 7 are affected by a peculiar Electrical Standing Wave (ESW) that forms behind the mixer in the IF (Intermediate Frequency) amplification chain. This modulation is not sinusoidal in nature and can only be dealt with by `fitHifiFringe` in case of very simple isolated lines (i.e. narrow, without wings). For more complex cases, a dedicated algorithm has been developed (so-called *matching technique*) and is offered in the `hebCorrection` task (details can be found here [hebCorrection](#)). We also recommend [Section 12.4](#). From HIPE 13.0 onwards, the ESW are automatically corrected in the pipeline by the task `doHebCorrection`, which applies prepared solutions stored in the HIFI calibration product. The same task can remove an already-applied correction if you wish to attempt a different correction yourself by using the task `HebCorrection`. Not every observation is perfectly corrected, unfortunately, though we are pursuing those judged in need of improvement. We invite you to consult the documentation cited above for more details.

- **Spurious spectral features:**

- There are two main sorts of spurious features in the HIFI data: narrow spur lines and saturation (probably a very broad form of the narrow spur). While the HIFI pipeline does its best to automatically detect and flag those, it is possible to fine-tune this data by flagging with the `flag-Tool` task (see [Section 11.5](#)).

### Platforming Example

Total power observations like Position Switch, often show abrupt offsets between WBS subbands and/or baseline distortions within subbands. The Obsid 1342252113 demonstrates a good example of this - see [Figure 2.19](#). The data are readily corrected with `fitBaseline` (see [Chapter 13](#)).



#### Note

The option `doglue = 0` will be needed to **not** join the WBS subbands.

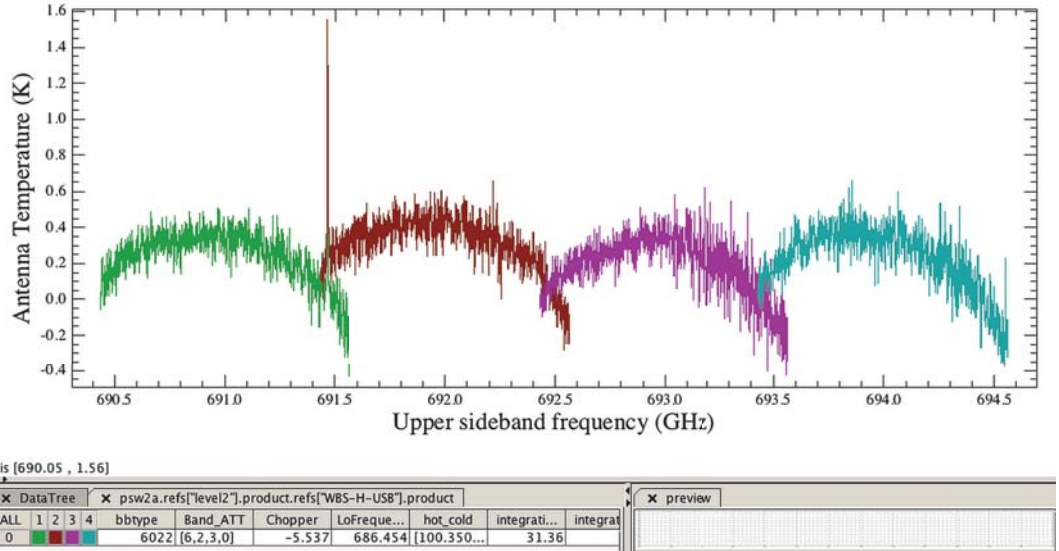


Figure 2.19. Level 2 WBS-H-USB SpectrumDataset of 1342252113. Note the strong curvature within subbands.

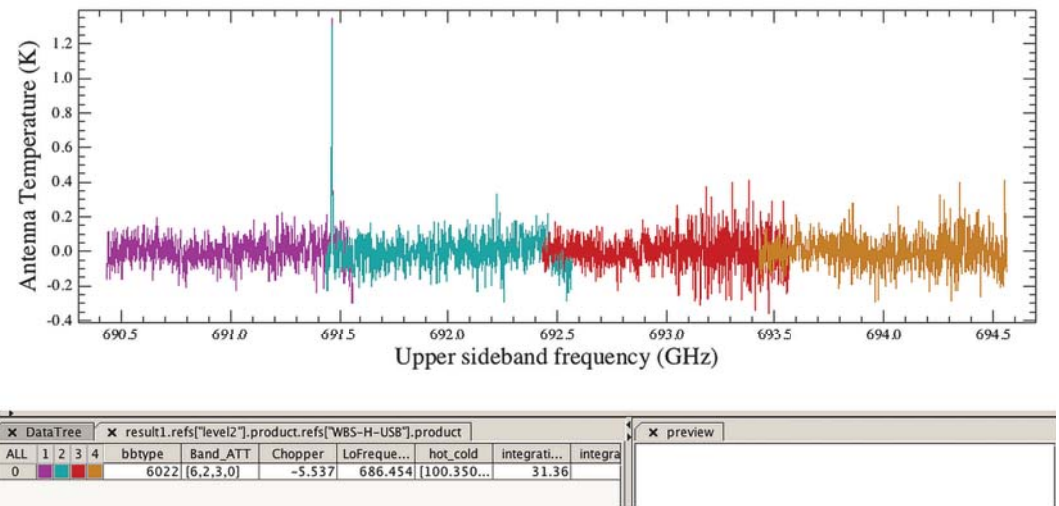


Figure 2.20. Level 2 WBS-H-USB SpectrumDataset of 1342252113 after baseline correction.

**Flagging bad data**

It is important to emphasise that Level 2 data are averaged data mixing potentially good and bad spectra. Hence, it is wise to carefully check the quality of the data at Level 1. When inspecting these Level 1 data, if you detect weird data (e.g. weird baseline, jump between WBS subbands, spurious spectral features) you can flag them (see [Section 11.5](#)), and then re-pipeline the observation.

**2.3.4.3. Inspecting OFF data for contamination**

Position Switch observations should be taken with an OFF position free of emission. To date, there are no known cases of contaminated OFF positions however, it is possible to investigate further. Note that the OFF position is synthetic and formed using the Cold load to mimic an empty sky. This leads to enhanced baseline instabilities.

You can find the OFF positions for all spectrometers of the observation in the Calibration product in the Observation Context; choose calibration → pipeline-out → ReferenceSpectra.

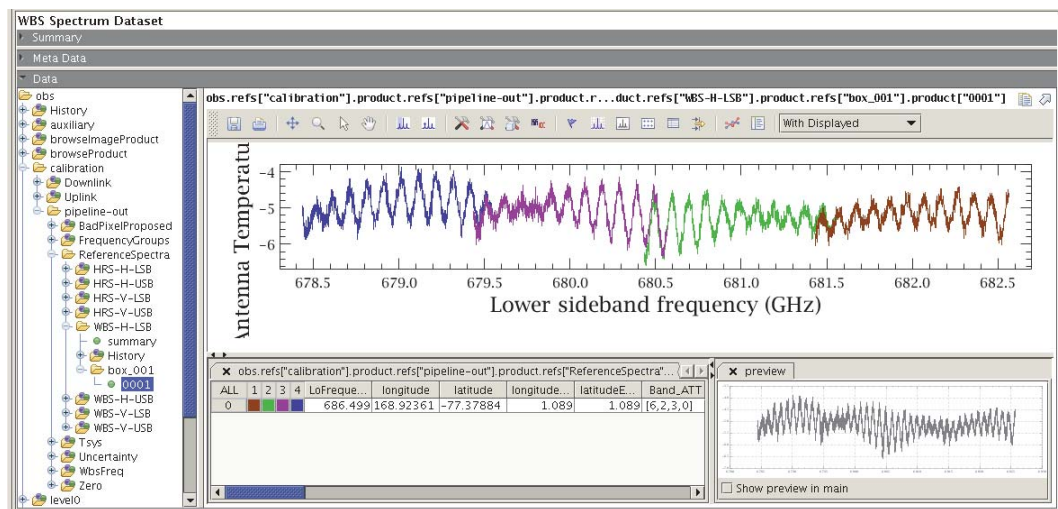


The OFF data is processed up to an equivalent Level 2, and have USB and LSB products. On fixed and moving targets, there is one single OFF spectrum per backend/sideband (the sum of all available OFF from the Level 1).

The OFF positions spectra can be run through the standard tasks such as `fitHifiFringe`, `fit-Baseline`, and `flagTool`, and will also be corrected from Electrical Standing Waves by default.

The OFF spectra in a Position Switch observation are key to the proper calibration of the observation. It is therefore recommended to examine all OFF spectra. You will be looking especially for emission contamination.

[Figure 2.21](#) shows an example of the 'load calibrated' OFF Spectrum for Obsid 1342252113 WBS-H-LSB. Like LoadChop observations without Refs, the standing waves can be very obvious. Since there is a significant difference between subbands, the defringing should not try to "glue" subbands together, i.e., `doglue=0`.



**Figure 2.21. Load Calibrated OFF for Obsid 1342252113 WBS-H-LSB. These will often display very strong standing waves.**

After running the task `fitHifiFringe` (using `nfringes=3` and `doglue=0`), the result can be inspected with the Spectrum Explorer (see [Figure 2.22](#)). For the data used in this example, we do not see any significant contamination, although cleaning the platforming should be done. Please note that the colour representing the subbands between e.g. [Figure 2.21](#) and [Figure 2.22](#) are unrelated to one another; Spectrum Explorer assigns random colours when plotting.

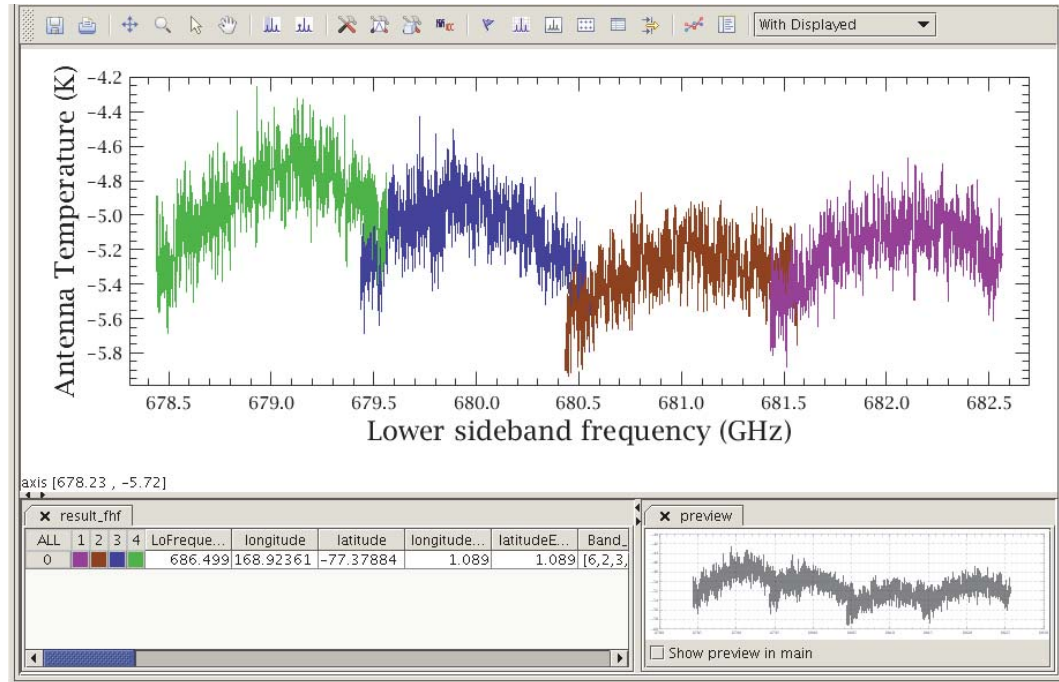


Figure 2.22. DeFRINGED OFF for Obsid 1342252113 WBS-H-LSB. No noticeable emission is present but platforming should be corrected.

## 2.4. Single Point Mode: Frequency Switch

Last updated: 6 March, 2015

### 2.4.1. Introduction

The Frequency Switching observing mode is one of the Single Point observing modes implemented in the HIFI instrument. This cookbook describes the strategy used to collect Frequency Switched data and addresses the main issues to take into account when working with such data from the archive.

### 2.4.2. How Single Point Mode Frequency Switch observations are taken

#### 2.4.2.1. Observing Principles

The Frequency Switching (FSW) observations consist in observing individual total power spectra with two slightly different Local Oscillator (LO) tunings (thereafter called LO1 and LO2), separated by the so-called frequency throw. If the frequency throw is small enough, the response of the instrument will not change much between LO1 and LO2. By forming the difference between the spectra taken at the respective LO tunings, the total power contributions from the instrument and sky can be cancelled out, while the line information is preserved, as lines will fall at a different Intermediate Frequencies (IF) due to the different tunings. As a consequence, an emission line will appear both as a positive and a negative feature, separated by the frequency throw. This is illustrated in [Figure 2.23](#).

The main advantage of this approach is that, unlike in the other HIFI modes, the line is observed both in the respective ON-target and Reference spectra (in effect both spectra are ON-target), optimising the observing time efficiency and therefore the achieved noise. With HIFI, two frequency throws were offered: a small one at around 100 MHz, and a larger one at around 300 MHz for SiS bands and 200 MHz for HEB bands (both either positive or negative).

Although the frequency throws were chosen to minimise the residual standing waves resulting from the different instrument response at the respective LO frequencies, the band-pass correction obtained

after performing this single difference is usually not sufficient. In order to remove this residual response, a Reference position is observed at a blank sky position. This position is also observed at both LO1 and LO2, the difference is taken and then subtracted from the ON-target data. Although it was recommended to always take FSW observations with a Reference position (and even more so in the HEB bands 6 and 7), some data have been observed without (so-called FSW NoRef), at the expense of a poorer baseline quality. We will illustrate later how to deal with these particular cases.

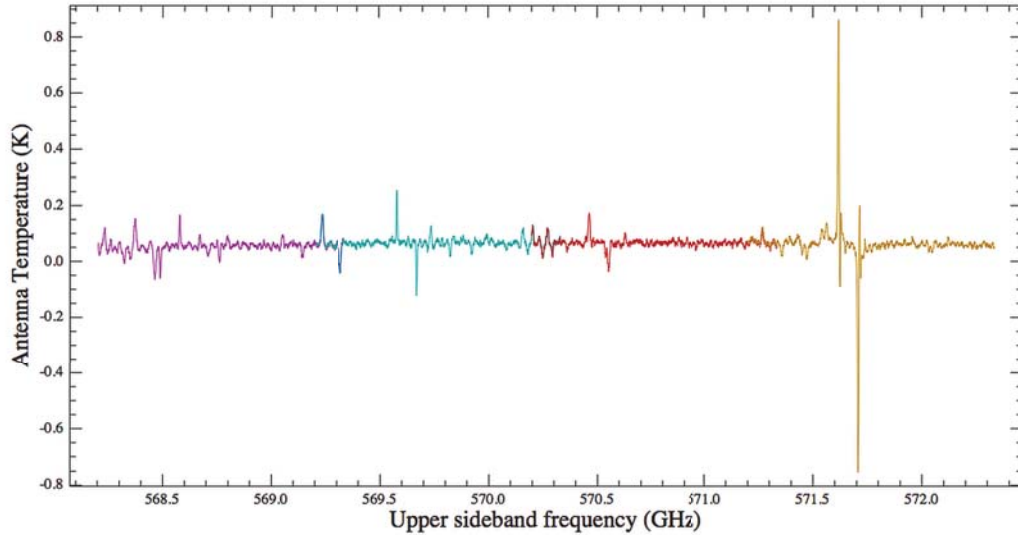


Figure 2.23. Example of a FSW observation in Obsid 1342180473. Note the co-existence of positive and negative features belonging to the respective LO1 and LO2 tunings. Data are shown in an Upper Sideband (USB) scale.

### Spectrometers in use

Both the Wide-Band (WBS) and High-Resolution Spectrometers (HRS) can be used simultaneously with this mode. Particular care is needed when observing with the HRS using a large throw at very high spectral resolution, as the line might fall outside of the spectrometer bandwidth (256 MHz) when tuning in the second phase LO2.

### 2.4.2.2. Observing Timeline

Figure 2.24 illustrates the different observing blocks involved in a Frequency Switching observation with a Reference. The different LO tunings are here labeled  $\nu_1$  and  $\nu_2$ . The switching between the two tunings is much faster (typically 100 msec) than any other change in e.g. telescope or chopper (the internal HIFI beam steering mirror) positions. In order to optimally calibrate the band-pass at each LO tuning, internal load measurements are also taken in FSW fashion, i.e. at each of the two tunings.

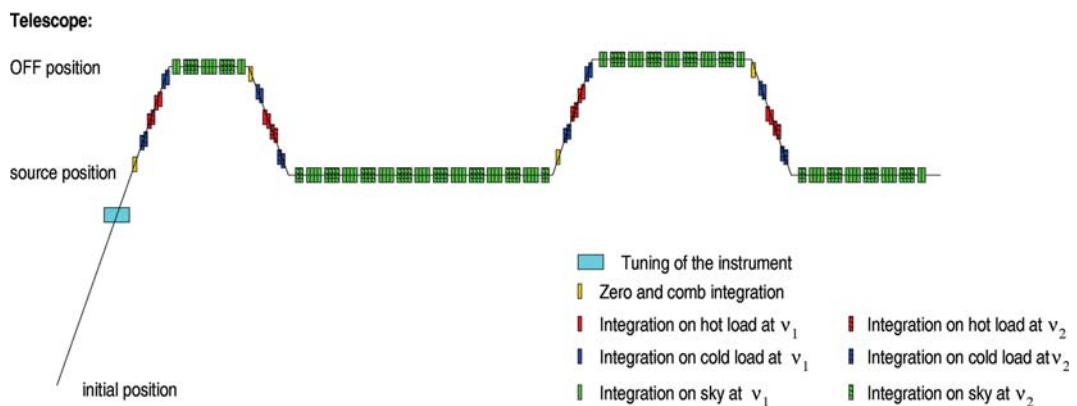


Figure 2.24. Sketch illustrating the observing sequence considered in Frequency Switching with a Reference. Observing blocks are labeled as in the legend showed at the bottom right.

### 2.4.2.3. Continuum information in Frequency Switching observations

The continuum from the observed source is lost in FSW observations. Indeed, because the respective LO1 and LO2 tunings are both performed on the target, all total power contributions are cancelled out in the subtraction process – only the lines are preserved as they shift in the IF scale.

### 2.4.2.4. Observation without a reference position

In order to avoid the overheads needed to observe a Reference position and make the best of the telescope time, some programmes have decided to use the *NoRef* option of the FSW mode. This option leads to relatively poorer baselines, as is illustrated in [Figure 2.25](#) and [Figure 2.26](#).

As shown in [Figure 2.25](#), having strong lines somehow mitigates the effect from the distorted baseline. When the lines are weak, as shown in [Figure 2.26](#), the analysis is more problematic. The detected lines can sometimes be recovered by applying a relatively heavy baseline correction, as will be shown in the following section. Typically this will only work for very narrow (and strong) lines, while broader lines or line wings will almost systematically be lost, or altered in the process.

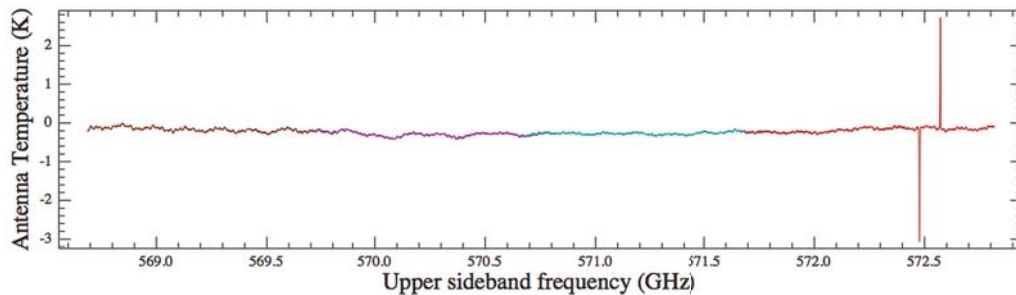


Figure 2.25. Same as [Figure 2.23](#) for a Frequency Switching observation where no Reference position was taken - Obsid 1342200897.

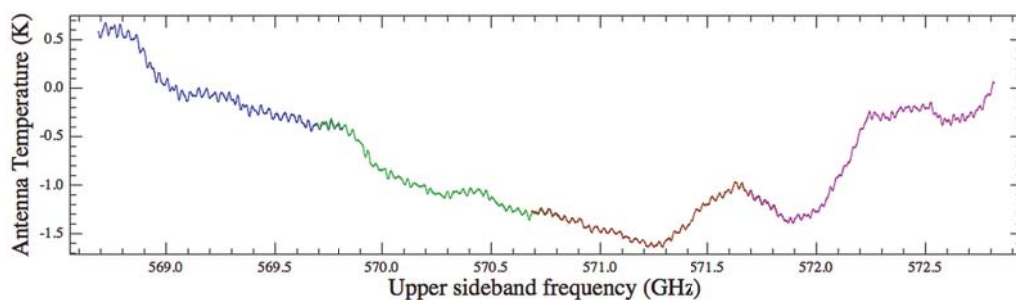


Figure 2.26. Same as [Figure 2.23](#) for a Frequency Switching observation where no Reference position was taken - Obsid 1342195094.

### 2.4.2.5. Frequency Switching in HEB bands

Using the FSW mode in HEB bands can be problematic due to the poor response stability of those bands. [Figure 2.27](#) illustrates the kind of spectra achieved in this fashion, dominated by very large baseline standing waves. In practice, almost no observation has been taken in this fashion so this remains a very isolated situation.

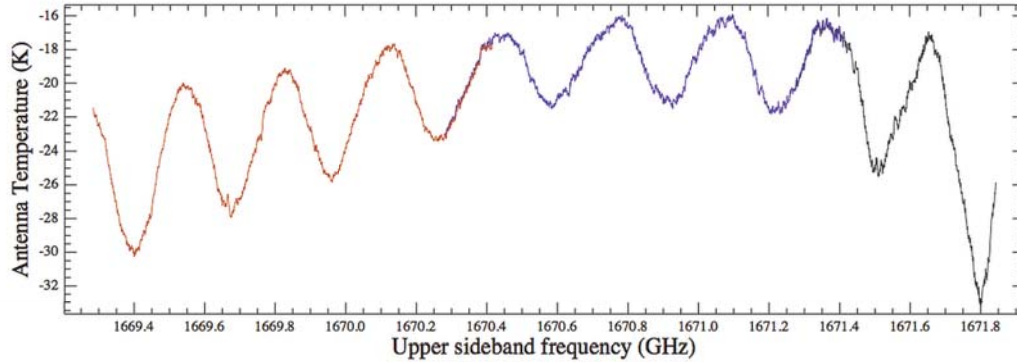


Figure 2.27. Same as [Figure 2.23](#) for a Frequency Switching observation in band 6b (Obsid 1342180813), highlighting the presence of Electrical Standing Waves.

## 2.4.3. Inspecting Single Point Mode Frequency Switch data

Get your observation into HIPE. Here, we use obsid=1342180473 as an example:

```
obs=getObservation(1342180473, useHsa=True)
```

The variable you just created using the `getObservation` command, `obs`, is not actually the data but a set of references to the data in the HSA. There is a reference for every product (e.g., Level 2.5, Level 2, Level 1, calibration. `HifiTimelineProduct`, `SpectrumDatasets`, see [Chapter 3](#) for more information about the contents of a HIFI `ObservationContext`) and the first time you access a product it will be downloaded from the HSA. This makes working slow and susceptible to lost connections; thus, it is strongly recommended to save the observation to a pool on your local disk and then work from that.

```
# Immediately save observation rather than work from HSA.
# Save in a pool with the obsid used as the pool name, save the
# calibration tree too, in case you want to re-pipeline the observation

saveObservation(obs, saveCalTree=True)
```

This command will save the observation into a pool called 1342180473 (the Observation Number).

## 2.4.4. Single Point Mode Frequency Switch Data Reduction

### 2.4.4.1. Data Structure

The Frequency Switching data structure shares many similarities with that of any of the Single Point modes. [Figure 2.28](#) illustrates the sequence of observation blocks present at the Level 1. After the initial calibration blocks, a series of OFF- and ON-target blocks is performed, in an “ABBA” pattern. At Level 1, the ON-target blocks (*HIFISwitchOnIntegration*) in fact already contain the subtraction of the Reference position spectra to that of the ON-target. The OFF-target blocks (*HIFISwitchOffIntegration*), however, solely correspond to observation at the Reference position. Note that less time is spent on the Reference position, so that these data are smoothed to a pre-defined channel width before being subtracted to the ON-target data.

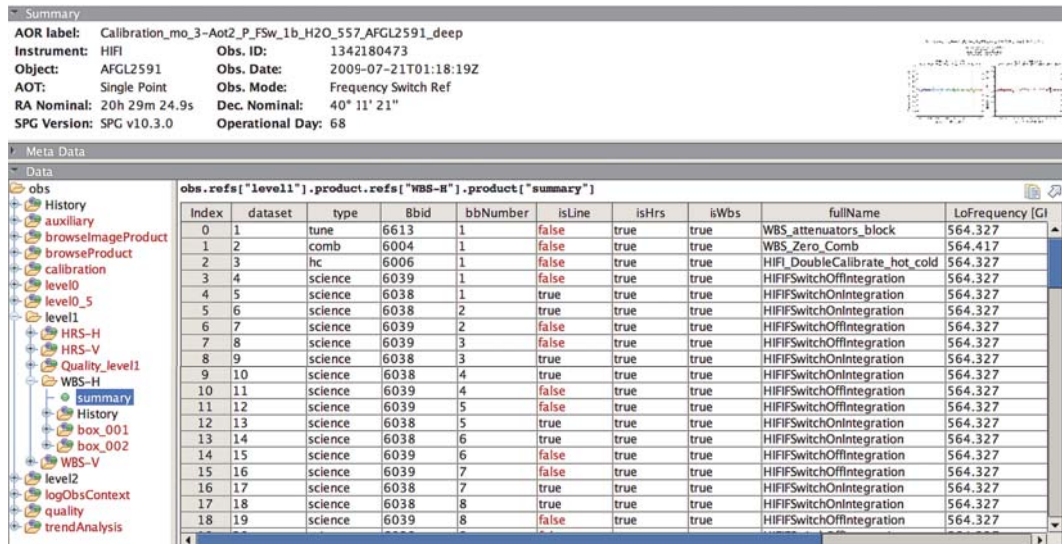


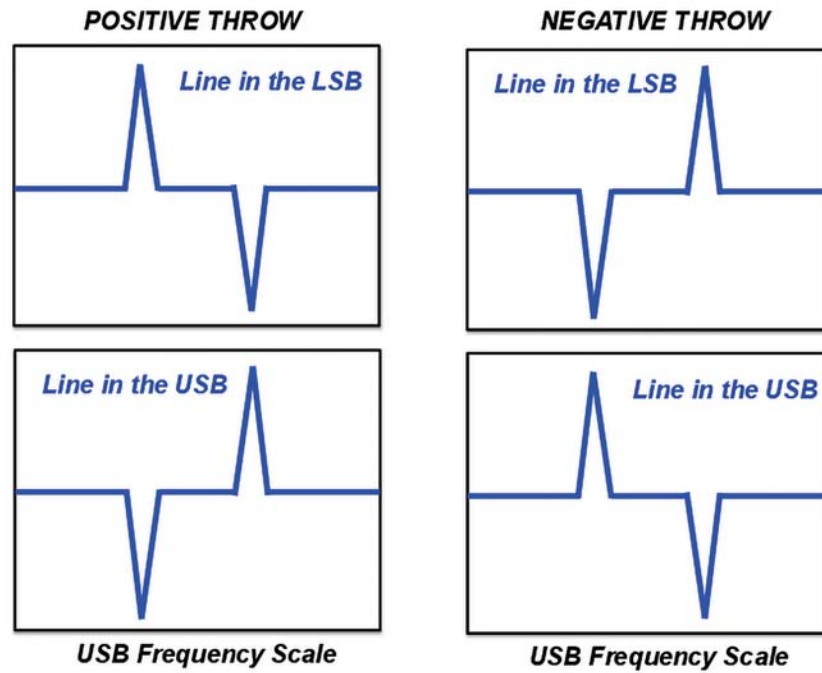
Figure 2.28. Observation context of a FSW observation, and summary table at Level 1 (only partial here).

At Level 2, only ON-target spectra are maintained and there will be only one averaged spectrum per spectrometer, scaled both in USB and LSB frequencies. The WBS spectra from this level are displayed together in the browse product shown at the upper right corner of the observation context visualiser. This picture is also used as preview post-stamp for the observation when running queries on the HSA.

### 2.4.4.2. Level 2: folded vs unfolded spectra

The data at Level 2 do not correspond to the last processing step of FSW data. Indeed, in order to make the most of the noise in the data, the spectra need to be folded. What this means is that the lines from both the LO1 and LO2 tuning should be combined, as they effectively correspond to two independent observations of the same line. The folding action basically works like a shift-and-add, except that this time the copied spectrum is shifted by the frequency throw, multiplied by -1 (hence making coincide in position and sign the two lines contributions), added to the original spectrum and finally divided by 2 (hence the noise improvement). The resulting line will consist of a central component bracketed by two negative ghosts of half the intensity (the balance between the intensity of the respective left and right ghost is also an indication of instrument stability).

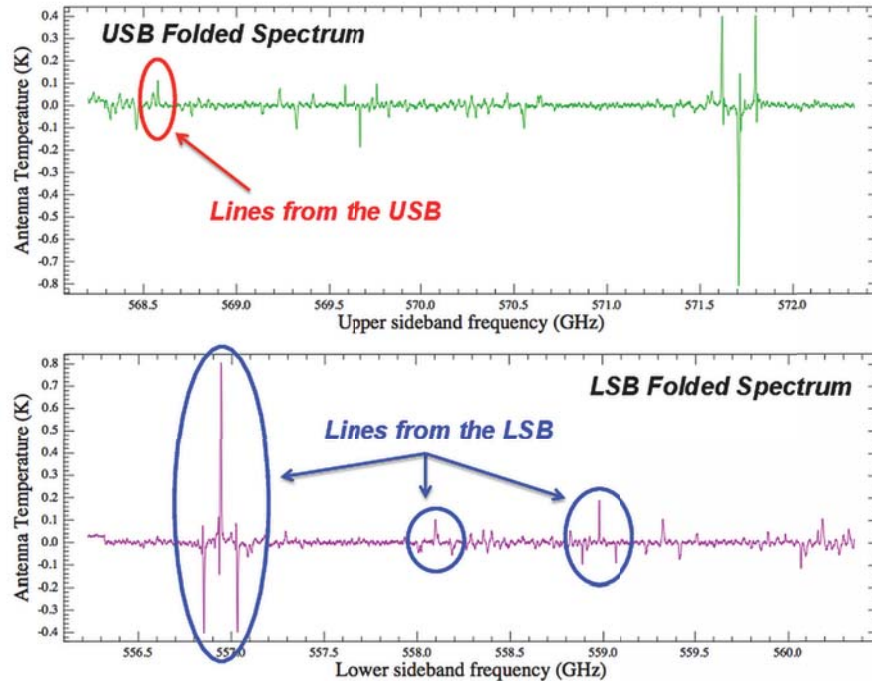
The sign of the shift however depends whether the spectrum is scaled in Lower Sideband (LSB) or Upper Sideband (USB). Indeed, for a given LO shift, lines from either the LSB or the USB will move in opposite direction. This can already be seen in [Figure 2.23](#) where some lines have their “negative” component at lower USB frequencies while others have it at higher USB frequencies. This trend is already sufficient to assign a sideband to the line if one knows the sign of the frequency throw. [Figure 2.29](#) summarises how the pair of LO1 and LO2 tuning lines would look like on an USB scale. The opposite would apply on a LSB scale.



**Figure 2.29.** Illustration of the relative positions of the respective LO1 (positive component) and LO2 (negative component) tuning lines for positive and negative frequency throws, and lines in either the LSB or USB. In all cases, the frequency scale assumed here is USB. Opposite direction will apply to data scaled in the LSB.

In order to perform the data folding, two tasks are made available: [fold](#) and [doFold](#). When calling those tasks, it is usually not necessary to indicate the value of the frequency throw, as it should be held in the data.

The output of the folding will inform directly about the instrument sideband to which a given line belongs. Indeed, the sign of the frequency shift applied in the folding will only be correct if the line belongs to the treated sideband. This is illustrated in [Figure 2.30](#). Lines belonging to a given sideband will appear as a positive feature bracketed with the two negative ghosts only on the applicable frequency scale. In the other scale, the central line will show up negative.



**Figure 2.30. Line sideband assignment after the folding. Upper panel: USB scale spectrum. Lower panel: LSB scale spectrum.**

Note that the Level 2 data are not folded by default. One of the reasons for this is that the baseline correction may be easier on unfolded data than on folded ones. Additionally, baseline subtraction in unfolded spectra involves two masks per lines, while three masks would be needed in folded data (to account for the two side-ghosts).

## Folding and Stitching

It is usually better to stitch your data before applying the folding. Indeed, when folding on non-stitched data, discontinuity will be created in the overlapping regions of the respective WBS or HRS subbands.

### 2.4.4.3. Data Artefacts and Cleaning

The typical data artefacts found in FSW observations are those associated to the imperfect band-pass correction inherent to the observing mode principles. As indicated earlier, these effects are enhanced when no Reference position was combined to the ON-target spectra. Those baseline distortions will mostly manifest in two forms:

## Standing Waves

Optical standing waves can be present in the data, with typical periods depending on the band used (see a detailed [report](#) here). Those baseline modulations are usually enhanced in the presence of non-negligible continuum and can become really severe on planet observations. For these standing waves, the `fitHifiFringe` task ([Chapter 12](#)) usually does a pretty good job. However, in FSW observations they often sit on top of complicated baseline structure (see below) so that baseline correction may be needed before. [Figure 2.32](#) illustrates a standing wave correction in an FSW observation without Reference. We have run here the correction per WBS subband (option `doglue = False`). This is usually more adequate so that irregular baseline structures are more easily taken into account by the task. Also, the `sub_base` option was used to remove a first order baseline level. In this particular case it is interesting to note that the main modulation occurred at a periods of  $\sim 36$  MHz and  $\sim 145$  MHz, which are not very common in HIFI data, but can occur in particular in such FSW observations.



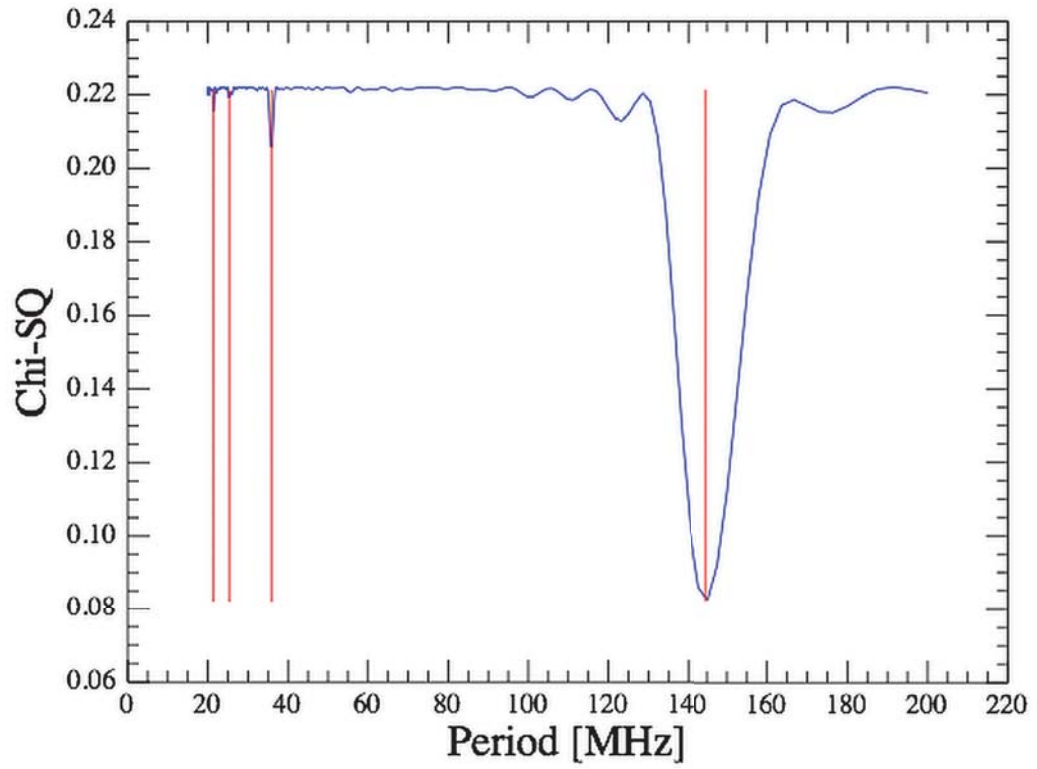
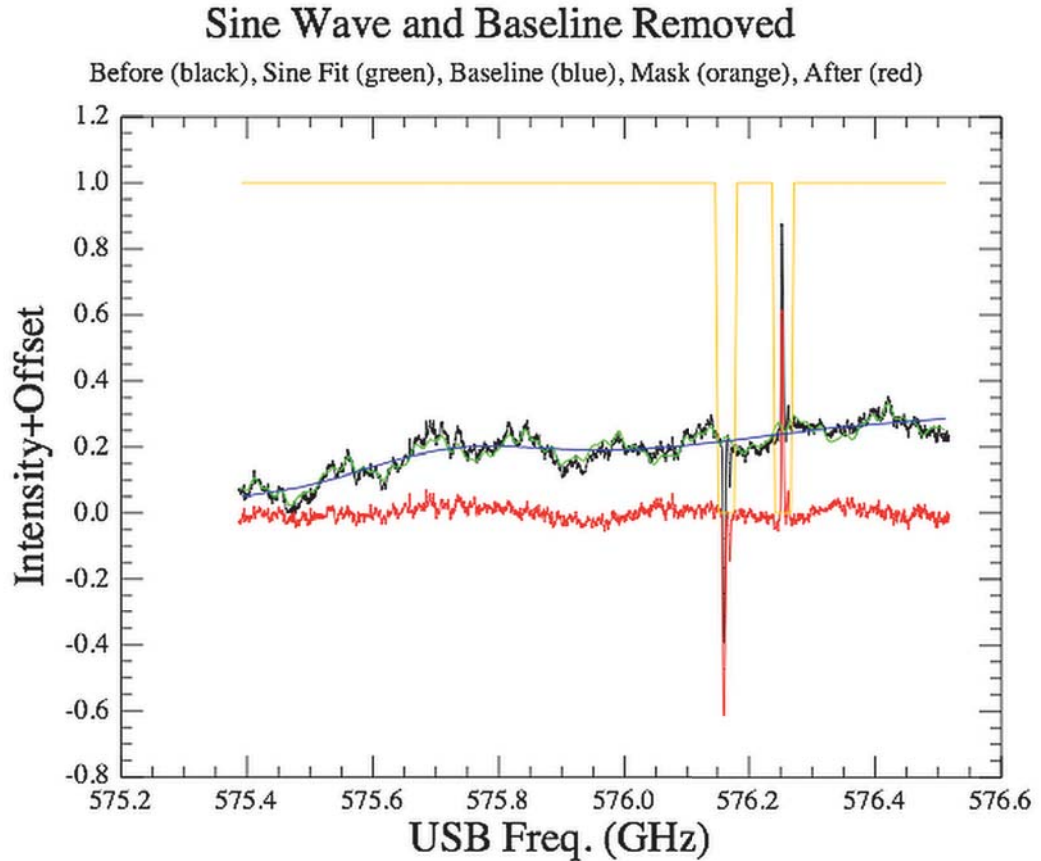


Figure 2.31. Defringing correction in a FSW observation with no Reference (Obsid 1342248900). Four standing waves components are considered here.



**Figure 2.32.** Spectrum before and after correction. The relatively strong line (CO 5-4) here is masked in both its positive and negative phases (frequency throw of 94 MHz).

Additionally, bands 6 and 7 are affected by a peculiar Electrical Standing Wave that forms behind the mixer in the IF amplification chain. An illustration of such a standing wave can be seen in [Figure 2.27](#). Since very little FSW observations were taken in those bands, we are not giving more details about this artefact here. Those Electrical Standing Waves can however be tackled with a dedicated task called `hebCorrection` (see [hebCorrection](#)). We also recommend [Section 12.4](#).

## Baseline structures

Because of the imperfect band-pass calibration, it will be common to have residual baseline structure, usually in the form of a slope or a more complex shape. These residual can be corrected using polynomial fits to the baseline with the `fitBaseline` task ([Chapter 13](#)). [Figure 2.33](#) illustrates the baseline correction on the defringed spectrum from [Figure 2.32](#). Here a polynomial fit of order 19 was used.

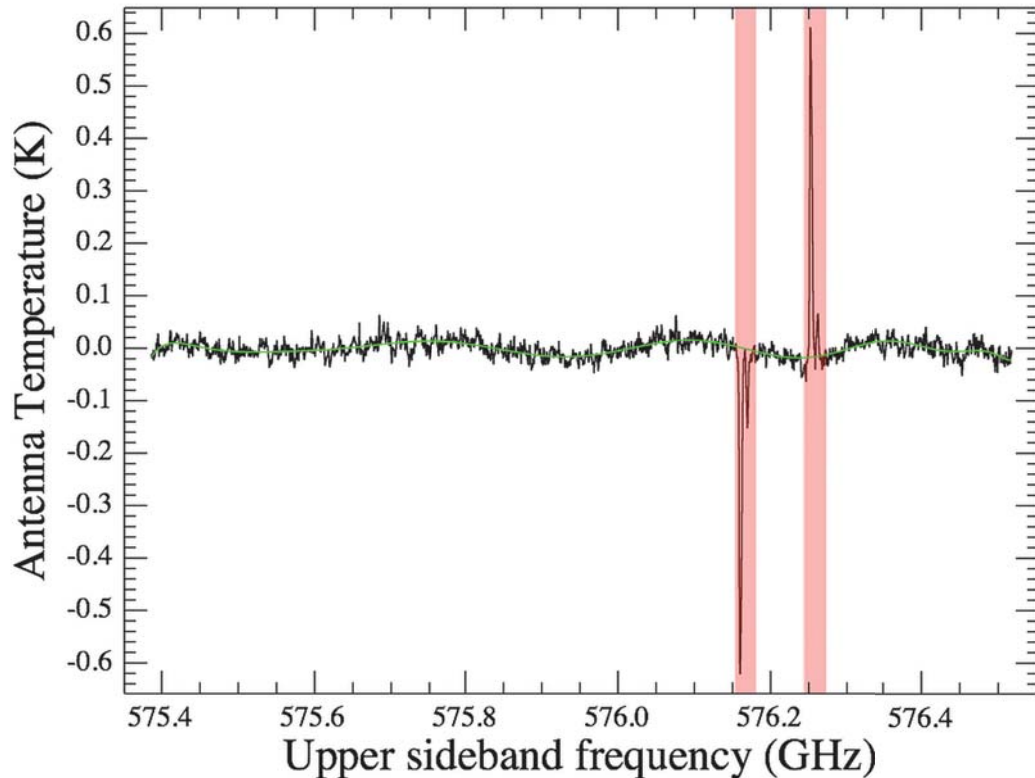


Figure 2.33. Baseline correction in Obsid 1342248900, after applying the defringing shown in [Figure 2.32](#).

### Artefact correction in HRS data

When HRS data have also been collected together with the WBS one, it may be interesting to perform the correction rather on those. Indeed, owing to their narrower instantaneous bandwidth, complicated baseline structure may be easier to fit (typically involving lower order polynomial fit) than when considered over the 1 GHz width of single WBS subbands. Of course, a higher noise will apply if a higher spectral resolution was used. However, data smoothing is possible to reconcile the noise level with those of the WBS.

### Other artefacts

On top of those, like any other HIFI observations, FSW data may suffer from spurious spectral features, either in the form of narrow spur lines, or saturation (probably a very broad form of the narrow spur). While the HIFI pipeline does its best to automatically detect and flag those, it is possible to fine-tune this data flagging with the `flagTool` task (see [Section 11.5](#)).

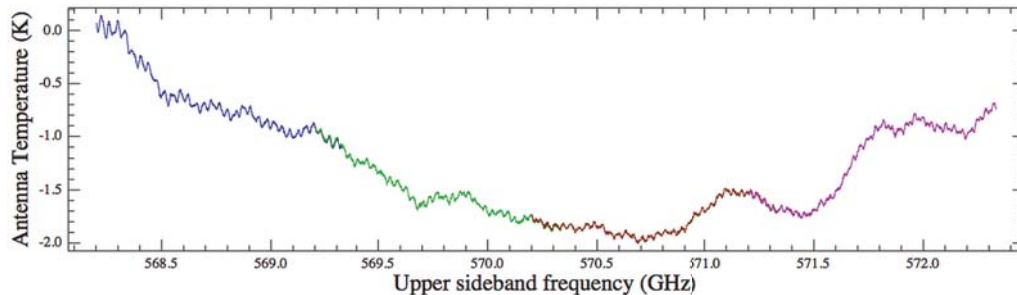
#### 2.4.4.4. Assessing emission in the OFF data

When taking FSW observations with a reference, it is possible to check whether the OFF data contain potential line contamination from a non-blank sky position. The OFF data is processed up to an equivalent Level 2 by the pipeline, and have USB and LSB products. You can find the OFF positions for all spectrometers in the Calibration product in the Observation Context; choose calibration → pipeline-out → ReferenceSpectra.

On fixed and moving targets, there is one single OFF spectrum per backend/sideband (the sum of all available OFF from the Level 1). The OFF positions spectra can be run through the standard tasks such as `fitHifiFringe`, `fitBaseline`, and `flagTool`, and will also be corrected from Electrical Standing Waves by default.

An example of an OFF position spectrum is shown in [Figure 2.34](#). It clearly shows the limitation of such checks. Indeed, the OFF spectrum formed here corresponds to a single difference and therefore

is equivalent to FSW observations taken without Reference. As a consequence, they will provide relatively poor baseline quality where the identification of any potential line contamination is more complex.



**Figure 2.34.** Example of Reference Spectrum for Obsid 1342180473. No particular contamination is observed here. Note the poorer baseline quality compared to the ON-target double-difference spectrum from [Figure 2.23](#).

#### 2.4.4.5. Combining data from both polarisations

Although the H and V mixers do not strictly point at the same position in the sky (the offset between the two is small compared to the beam at the applicable frequencies), it is often useful to combine the signal from the two polarisations to improve the signal-to-noise. Because the band-pass shape is different in the two mixers, it is probably best to perform this merging after the baseline correction. On the other hand, once the latter is done, combining H and V before or after the folding should not make any difference.

This combination can be done using the [polarPair](#) task (also available in a GUI).

```
specTotal = polarPair(ds1=specH, ds2=specV)
```

When weak lines are targeted in FSW mode, disentangling the contribution of a line from that of complicated baseline structure can make use of the distinct H and V data. Typically, in case of doubtful line identification, not only should you check whether the feature is seen in both polarisations, but you should also verify whether the negative component of the FSW also appears at the right frequency throw, and so in both polarisations with a similar intensity as the candidate positive component.

#### 2.4.4.6. Exporting spectra

The Level 2 spectra can be exported to Class at any stage (see [Chapter 23](#)). In case data are exported prior to the folding, Class will have no problem to recognise the frequency throw and apply it with its own fold command. It should be noted, however, that if you export a spectrum already folded with `hiClass`, Class will not know it a priori so you may still be able to run the fold command there, creating an erroneous spectrum.

## 2.5. Single Point Mode: Load Chop

Last updated: 24 March, 2015

### 2.5.1. Introduction

This mode uses the internal loads for reference with an optional OFF sky reference. The most common problems faced when working with Load Chop mode observations are dealing with standing waves

if the OFF reference option has not been chosen by the user. This applies most strongly to Bands 3, 4, 6, and 7.

The observations used here are a LoadChop CO 5-4 observation (band 1b) of  $\alpha$  Ceti (Obsid 1342190841) and a LoadChopNoRef CH+ observation (band 3a) of DR21 (Obsid 1342180551). Therefore, you can follow the script exactly with the data used in it. To learn about removing baselines, fitting to lines, combining H and V spectra, and other data analysis steps that are not observing mode specific, you can find information elsewhere in this Guide.

## 2.5.2. How Single Point Mode Load Chop observations are taken

### 2.5.2.1. Observing Principles

Load Chop observations are taken by alternately looking at the target on the sky and an internal source of radiation with a typical period of a few seconds. Hence, an internal cold calibration source (load, physical temperature around 10 K) is used as a reference to correct short term changes in instrument behaviour. Since the optical path differs between source and internal reference, a residual standing wave structure may remain. Even if the telescope by itself does not move during the process, such a scheme has relatively high dead times (typically only 1/3 to 1/4 of the total time spent on the actual science target). Additional calibration using an OFF sky reference position is optional but highly recommended whenever possible for standing wave correction in the pipeline. This applies most strongly to Bands 3, 4, 6, and 7. See [Figure 2.35](#) for a schematic representation. This sequence is repeated as many times as needed to attain the requested noise.

### 2.5.2.2. Observing Timeline

[Figure 2.35](#) illustrates the different observing blocks involved in a Load Chop observation for the case of the Load Chop with OFF sky reference option selected. For one LO tuning (cyan box), a calibration block will be taken (yellow, red, and blue boxes in [Figure 2.35](#)) in-between two target blocks (sequence of green and blue boxes). The target blocks are a sequence of integration on sky and on cold load. These target blocks are first on the OFF sky reference, then two times on source position, hereafter two times on the OFF sky reference, and finally on source. For the *noRef* mode, the sequence is the same but without the OFF sky reference target block. This sequence is repeated as many times as needed to attain the requested noise.

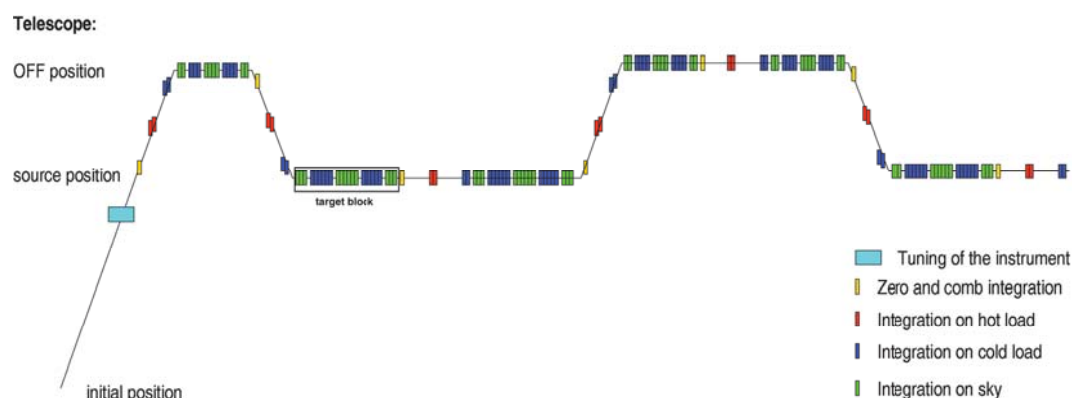


Figure 2.35. Load Chop observations.

## 2.5.3. Inspecting Single Point Mode Load Chop data

Get your observation into HIPE. Here we download a Load Chop with OFF reference observation from the Herschel Science Archive (HSA).

```
obs=getObservation(1342190841, useHsa = True)
```

The variable you just created using the `getObservation` command, `obs`, is not actually the data but a set of references to the data in the HSA. There is a reference for every product (e.g., Level 2.5, Level 2, Level 1, calibration) which are displayed on the editor window when double-clicking on the `obs` variable. See [Chapter 3](#) for more information about the contents of a HIFI `ObservationContext`.

The first time you access a product it will be downloaded from the HSA. This makes working slow and susceptible to lost connections; thus, it is strongly recommended to save the observation to a pool on your local disk, and then work from that.

```
# Immediately save observation rather than work from HSA.
# Save in a pool with the obsid used as the pool name, save the
# calibration tree too, in case you want to re-pipeline the observation
saveObservation(obs, saveCalTree = True)

#This command will save the observation into a pool called 1342190841 (Obsid).
```

### 2.5.3.1. Summary and MetaData tables

The *Summary* shows you the headlines of the observation, including the nominal coordinates and the HIPE version (*SPG*) that has been used to process these data:

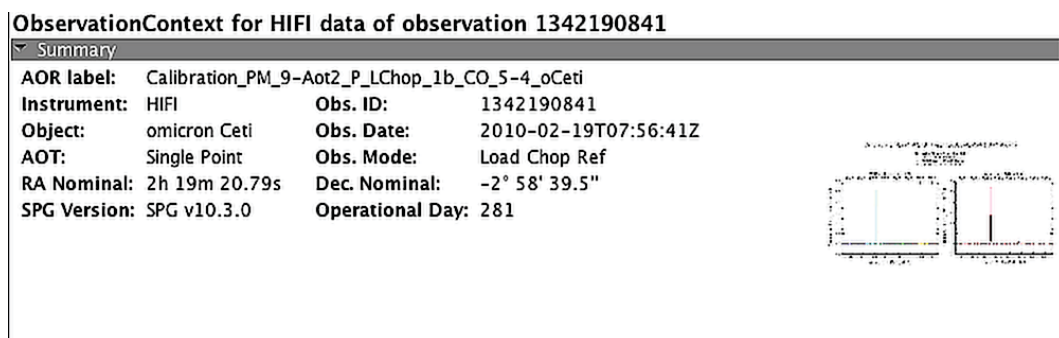


Figure 2.36. Summary

A browse product giving an overview of the WBS spectra in both polarisations is shown in the right-hand side. This image can also be seen by clicking on the `browseImageProduct` in the *Data* window and corresponds to the post-card displayed in the HSA for this particular observation:

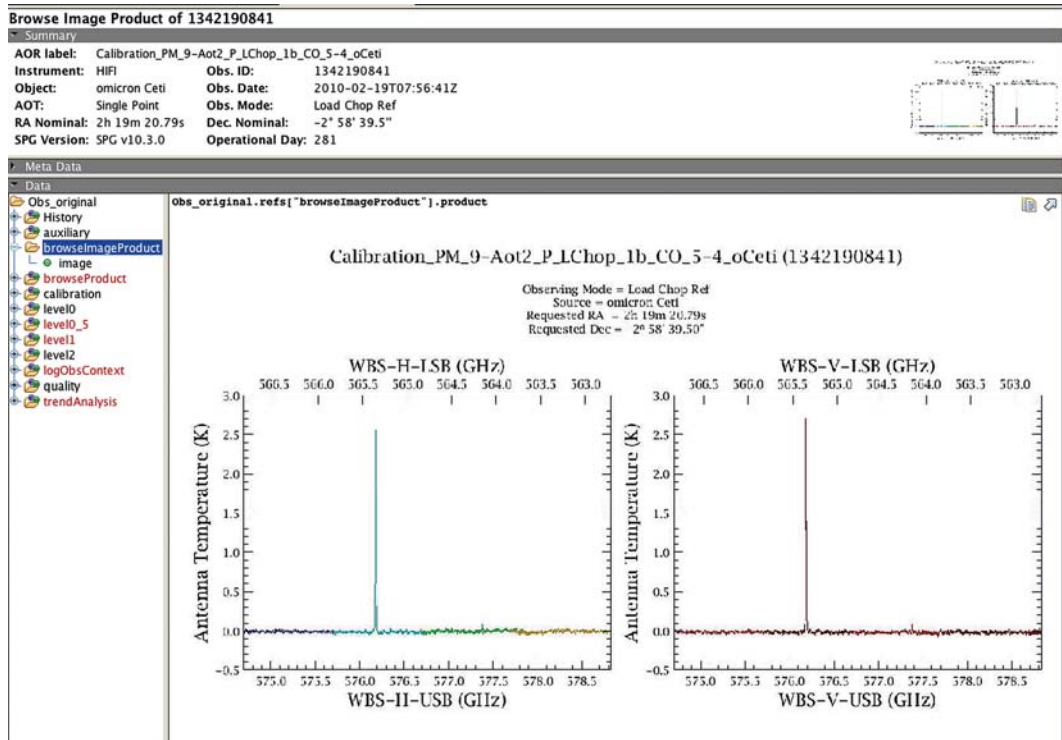


Figure 2.37. BrowseImageProduct

These pictures are automatically generated from the Level 2 product. Then, verify in the *MetaData* that the observed coordinates (*ra*, *dec*) do not differ (within the pointing accuracy) from the requested ones (*raNominal*, *decNominal*). The requested (RA,DEC) can be found in the MetaData of any Level 0 product:

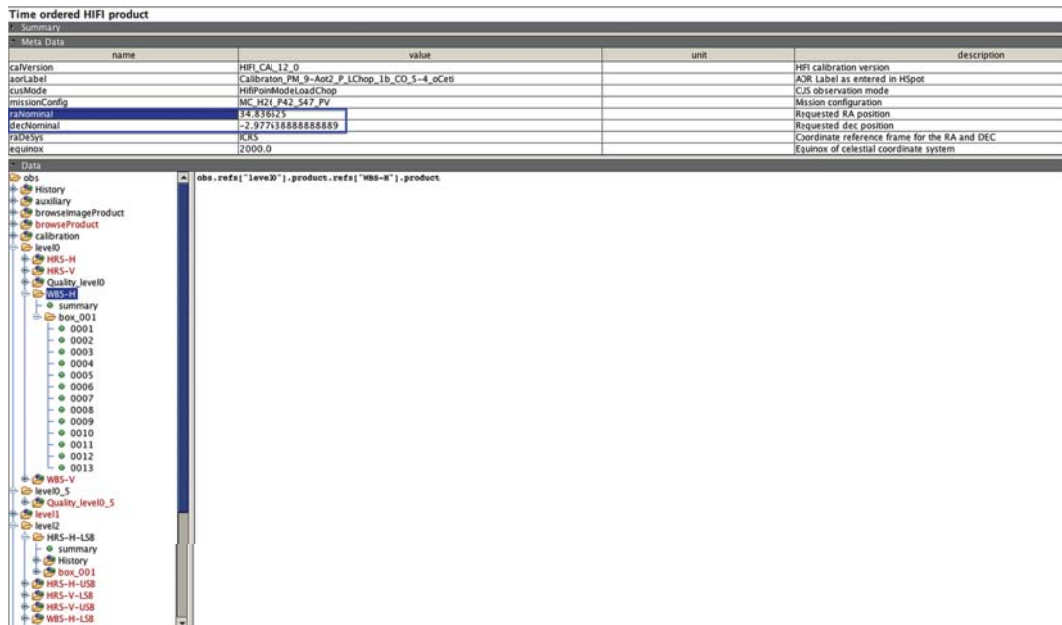


Figure 2.38. Level 0 MetaData

For this observation, RA= 34.836625 and DEC= -2.977638888888889. If you open any Level 2 *HrsSpectrumDataset* with the *Spectrum Explorer*, you will see the observed coordinates, e.g.:

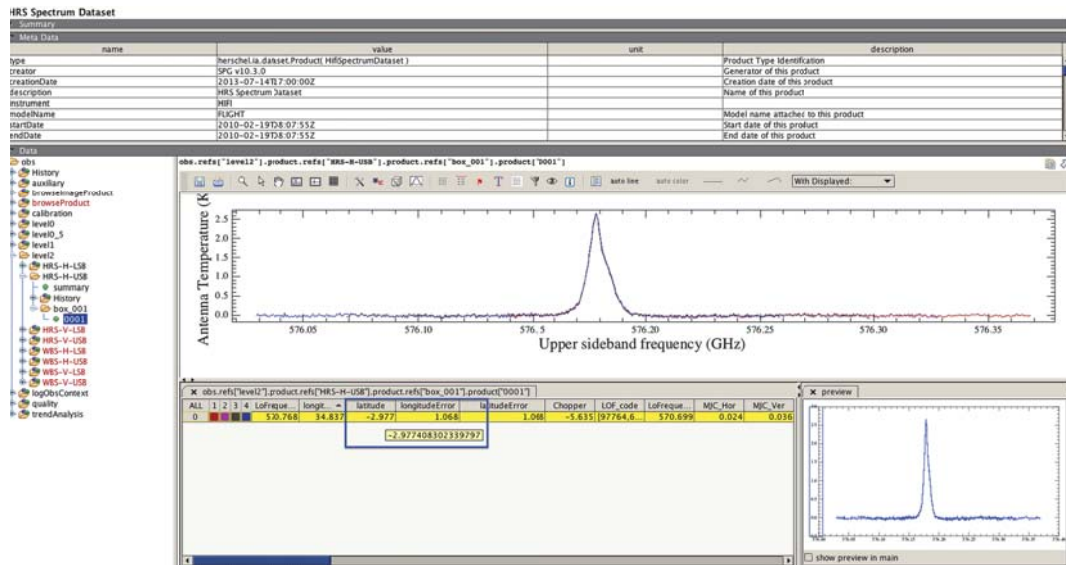


Figure 2.39. Level 2 HrsSpectrumDataset opened with Spectrum Explorer.

Place your mouse cursor on the longitude or latitude value, and the complete number will appear. Here we have RA= 34.83679065844212 and DEC= -2.977408302339797.

Hence, the difference between observed and requested is (1.6565e-4, 2.30586e-4) degrees, i.e. (0.6,0.83) arcseconds. To assess how good that is, you should check the pointing accuracy of the satellite. The *Absolute Pointing Error (APE)* is not the same all over the mission - see this following [document](#) for more information.

This observation has been performed on OD281 when the *APE* was 1.9-2.0'', so well beyond the difference measured here.

## 2.5.4. Single Point Mode Load Chop Data Reduction

### 2.5.4.1. Data structure

#### Level 1 data

[Figure 2.41](#) and [Figure 2.40](#) illustrate the *summary* tables of this Load Chop observation at the Level 1, where calibration and OFF observations are still present. Products are present for HRS and WBS backends for both polarisations.

The LO tuning (*tune*) takes place right before the calibration block and has a fixed duration that depends on the frequency. Tuning indeed implies some thermal stabilisation of the LO chain so that a dead time allowing for this is allocated into the tuning block. Right after in this summary table, one can recognise the sequence illustrated in [Figure 2.35](#): the calibration block, composed of a *comb* (WBS frequency calibration) and an *hc* (code for “Hot-Cold”, for the bandpass and flux calibration), is followed by the OFF- and ON-target blocks in the order given in [Section 2.5.2.2](#). Note that Level 1 data for ON-target Bbid are already made of a double difference.

The sequence of the observation with HRS backend is listed in the table shown in [Figure 2.40](#). The structure is the same for the WBS data (see [Figure 2.41](#)) but of course without the *comb*.



SummaryTable

Summary

Meta Data

Data

obs\_original.refs["level1"].product.refs["HRS-H"].product["summary"]

Index	dataset	type	Bbid	bbNumber	isLine	isHrs	isWbs	fullName	LoFrequency [GHz]	LO-Throw [GHz]	start	length
0	1	tune	6601	1	false	false	false	HRS_tune_block_aot	570.699	0.0	0	2
1	2	hc	6005	1	false	true	true	HIFI_Calibrate_hot_cold	570.699	0.0	2	2
2	3	science	6036	1	false	true	true	HIFILoadChopOffIntegration	570.699	0.0	4	13
3	4	science	6035	1	true	true	true	HIFILoadChopOnIntegration	570.699	0.0	17	22
4	5	science	6035	2	true	true	true	HIFILoadChopOnIntegration	570.699	0.0	39	22
5	6	science	6036	2	false	true	true	HIFILoadChopOffIntegration	570.699	0.0	61	13
6	7	science	6036	3	false	true	true	HIFILoadChopOffIntegration	570.699	0.0	74	13
7	8	science	6035	3	true	true	true	HIFILoadChopOnIntegration	570.699	0.0	87	22
8	9	science	6035	4	true	true	true	HIFILoadChopOnIntegration	570.699	0.0	109	22
9	10	hc	6005	2	false	true	true	HIFI_Calibrate_hot_cold	570.699	0.0	131	2
10	11	science	6036	4	false	true	true	HIFILoadChopOffIntegration	570.699	0.0	133	13

Figure 2.40. Level 1 data summary for HRS.

SummaryTable

Summary

AOR label: Calibration\_PM\_9-Aot2\_P\_LChop\_1b\_CO\_5-4\_oCeti

Instrument: HIFI Obs. ID: 1342190841

Object: omicron Ceti Obs. Date: 2010-02-19T07:56:41Z

AOT: Single Point Obs. Mode: Load Chop Ref

RA Nominal: 2h 19m 20.79s Dec. Nominal: -2° 58' 39.5"

SPG Version: SPG v10.3.0 Operational Day: 281

Meta Data

Data

obs.refs["level1"].product.refs["WBS-H"].product["summary"]

Index	dataset	type	Bbid	bbNumber	isLine	isHrs	isWbs	fullName	LoFrequency [GHz]	LO-Throw [GHz]	start	length
0	1	tune	6613	1	false	true	true	WBS_attenuators_block	570.699	0.0	0	3
1	2	comb	6004	1	false	true	true	WBS_Zero_Comb	570.699	0.0	3	2
2	3	hc	6005	1	false	true	true	HIFI_Calibrate_hot_cold	570.699	0.0	5	2
3	4	science	6036	1	false	true	true	HIFILoadChopOffIntegration	570.699	0.0	7	13
4	5	science	6035	1	true	true	true	HIFILoadChopOnIntegration	570.699	0.0	20	22
5	6	science	6035	2	true	true	true	HIFILoadChopOnIntegration	570.699	0.0	42	22
6	7	science	6036	2	false	true	true	HIFILoadChopOffIntegration	570.699	0.0	64	13
7	8	science	6036	3	false	true	true	HIFILoadChopOffIntegration	570.699	0.0	77	13
8	9	science	6035	3	true	true	true	HIFILoadChopOnIntegration	570.699	0.0	90	22
9	10	science	6035	4	true	true	true	HIFILoadChopOnIntegration	570.699	0.0	112	22
10	11	comb	6004	2	false	true	true	WBS_Zero_Comb	570.699	0.0	134	2
11	12	hc	6005	2	false	true	true	HIFI_Calibrate_hot_cold	570.699	0.0	136	2
12	13	science	6036	4	false	true	true	HIFILoadChopOffIntegration	570.699	0.0	138	13

Figure 2.41. Level 1 data summary for WBS.

Figure 2.42 shows the content of the Level 1 HRS-H Dataset. Each line represents a *dataset block* (from 1 to 11) which itself is made of several *SpectrumDataset*. Each dataset block corresponds to a type of event (tuning, hot-cold calibration, Off Integration, On Integration). For instance, the dataset 3 is a block of 13 *SpectrumDatasets* (from 0 to 12) corresponding to the OFF integration on the internal load.

Note that the frequency scale is Intermediate Frequency (IF) at this stage.

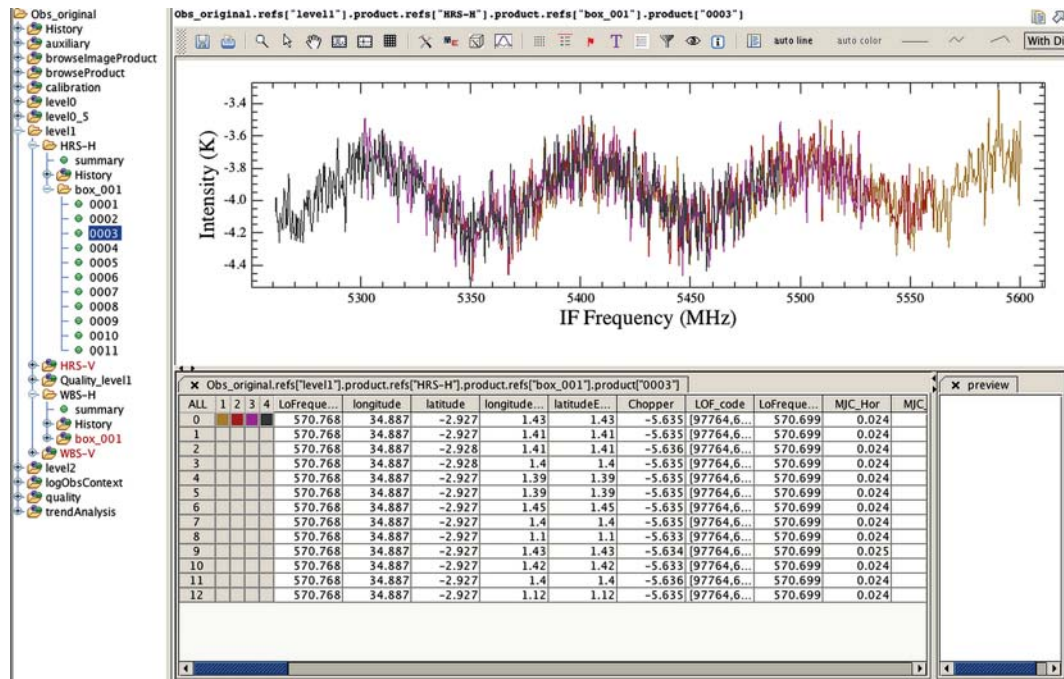


Figure 2.42. Level 1 HRS SpectrumDataset.

### Level 2 data

At Level 2, only ON-target spectra are maintained and there is one spectrum (*HifiTimelineProducts*, HTP) per backend/sideband/polarisation. Click your way through the tree until you reach the spectrum of interest.

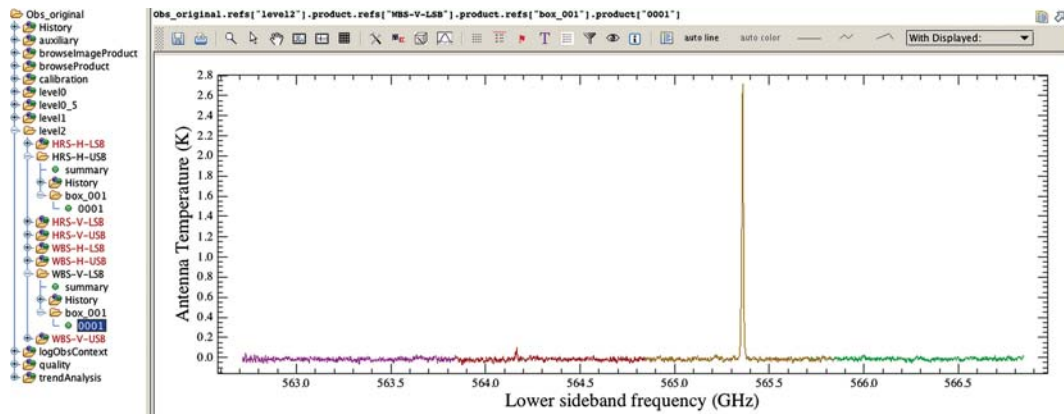


Figure 2.43. Level 2 WBS SpectrumDataset.

At a first glance, data are perfect, without any baseline issues. Note that the spectrum is in  $T_A^*$  temperature scale.

### 2.5.4.2. The OFF (sky reference) spectrum

#### How the OFF spectrum is used in the data reduction

For the FSW and Load Chop observing modes (both for pointed and mapping modes), it is highly recommended to have a reference spectrum measured at a reference position. This reference spectrum is observed with a shorter integration time than the ON source spectrum, and as a consequence has a higher noise for the same spectral resolution. In the double difference method applied in the pipeline,

the OFF spectrum is then smoothed to reduce the extra noise from that reference measurement. In the pipeline, what is thought to be the optimal smoothing (see [Figure 2.44](#)) is applied to the reference spectrum in order to improve the S/N in the final observation for the user. Note that since the OFF spectrum is the result from (OFF\_sky - COLD\_load), it is possible that the overall level of this spectrum is negative because the 'blank sky' will often have a lower temperature equivalent to that of the cold source.

band	frequency [GHz]	LoadChop [MHz]	FSwitch [MHz]
1a	480.0	9.0	11.0
1a	640.0	9.0	11.0
1b	480.0	9.0	11.0
1b	640.0	9.0	11.0
2a	640.0	9.0	11.0
2a	800.0	9.0	11.0
2b	640.0	9.0	11.0
2b	800.0	9.0	11.0
3a	800.0	9.0	11.0
3a	960.0	9.0	11.0
3b	800.0	9.0	11.0
3b	960.0	9.0	11.0
4a	960.0	9.0	11.0
4a	1120.0	9.0	11.0
4b	960.0	9.0	11.0
4b	1120.0	9.0	11.0
5a	1120.0	9.0	11.0
5a	1280.0	9.0	11.0
5b	1120.0	9.0	11.0
5b	1280.0	9.0	11.0
6a	1420.0	30.0	11.0
6a	1522.0	30.0	11.0
6a	1522.01	18.0	11.0
6a	1570.0	18.0	11.0
6b	1570.0	18.0	11.0
6b	1655.0	18.0	11.0
6b	1655.01	30.0	11.0
6b	1710.0	30.0	11.0
7a	1710.0	18.0	11.0
7a	1910.0	18.0	11.0
7b	1710.0	18.0	11.0
7b	1910.0	18.0	11.0

**Figure 2.44.** Smoothing width applied on the OFF spectra depending on the frequency.

Nevertheless, you might want to try different smoothing by tuning the Level 1 task [mkOffSmooth](#) in the *Customize Pipeline*:

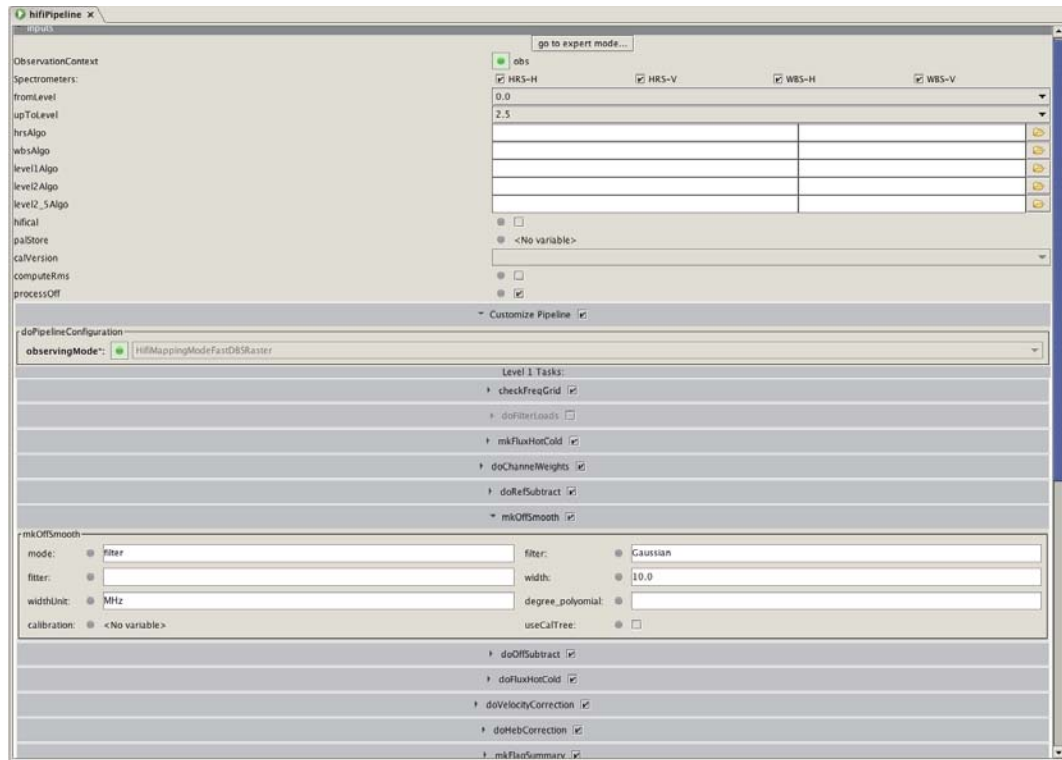


Figure 2.45. Customise pipeline with Level 1 task `mkOffSmooth`.

## Any contamination in the OFF spectrum ?

It is wise to verify that there is no signal (contamination) in the OFF sky reference spectrum (if any).

The OFF data is processed up to an equivalent Level 2 by the pipeline, and have USB and LSB products. You can find the OFF positions for all spectrometers in the Calibration product in the Observation Context; choose calibration → pipeline-out → ReferenceSpectra.

On fixed and moving targets, there is one single OFF spectrum per backend/sideband (the sum of all available OFF from the Level 1). Because the OFF spectrum corresponds to a single difference, the consequence is relatively poor baseline quality where the identification of any potential line contamination is more complex. Note that OFF positions spectra can be run through the standard tasks such as `fitHifiFringe`, `fitBaseline`, and `flagTool`, and will also be corrected from Electrical Standing Waves by default.

Example of an OFF spectrum (for the WBS-H-LSB):

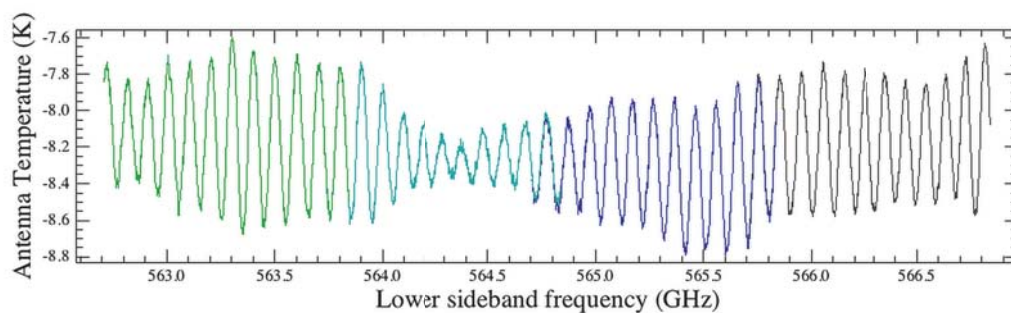


Figure 2.46. OFF (sky reference) spectra.

Unfortunately, because of the strong standing waves in this example, it is difficult to know if there is any contamination. You might try to apply the `fitHifiFringe` task (see [Chapter 12](#)) on it in

order to remove those waves. To do so, select the observation (*obs*) and double-click on the *fitHiFiFringe* task shown in the *Applicable Tasks* window of HIPE. A menu will appear in order to tune the task. With such waves, we recommend to select 3 fringes, with a typical period of 100 MHz and to apply it on individual WBS subbands (unselect *dogLue*) as follows:

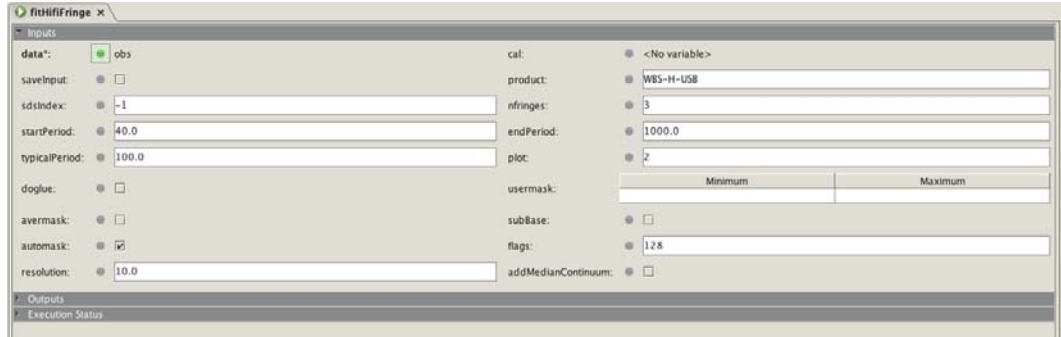


Figure 2.47. *fitHiFiFringe* task menu.

The result will be:

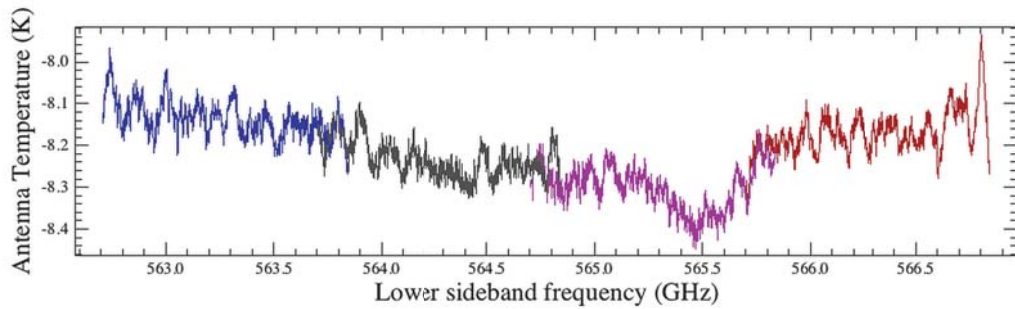


Figure 2.48. OFF (sky reference) spectrum output of the *fitHiFiFringe* task.

But, in this case, it is still difficult to check.

[Figure 2.49](#) is an example of a different observation (1342190778, band 7b) with a clear  $C^+$  contamination around 1900.65 GHz:

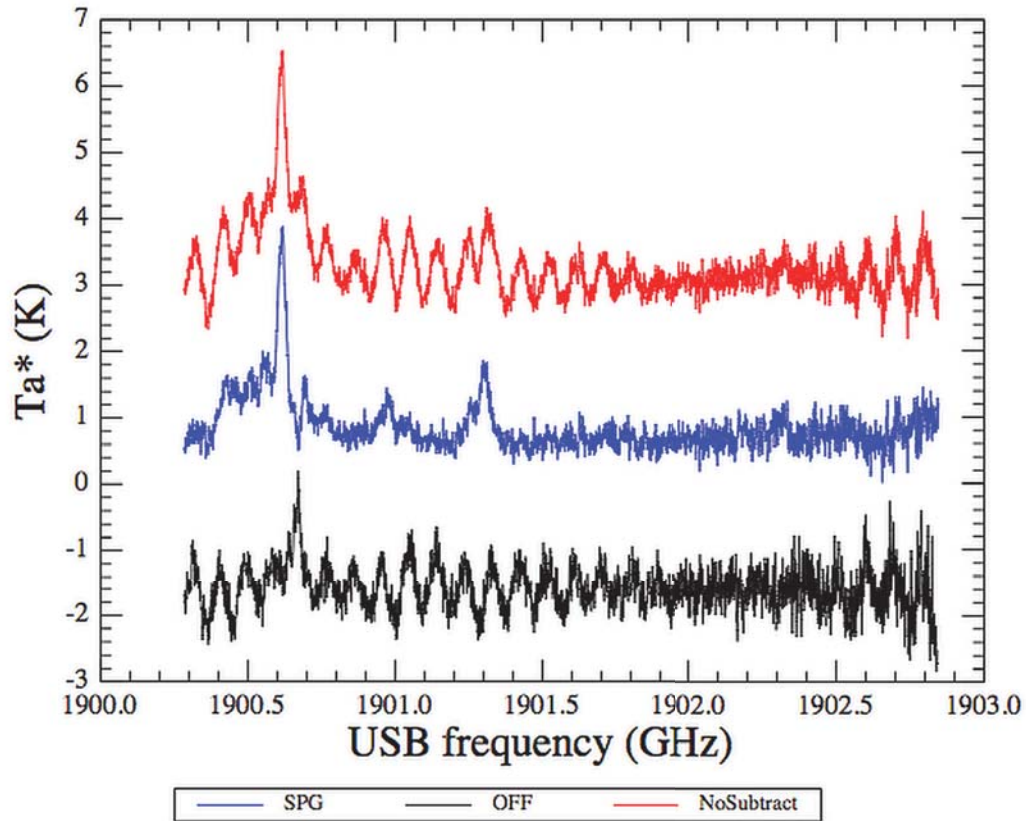


Figure 2.49. Example of OFF contamination in 1342190778 (OFF spectrum in black, ON spectrum in red, ON-OFF in blue).

In this example, the residual standing waves in the single difference spectra (ON and OFF respectively) is only properly cancelled out in the ON-OFF difference, albeit implying a distortion of the  $C^+$  line due to the contamination in the OFF. In order to use a non-contaminated ON spectrum, you should either apply some defringing to this data, or subtract the OFF only in spectral ranges sufficiently separated from the  $C^+$  line, at the expense of some residual standing wave modulation in the frequency range applying to this line.

### 2.5.4.3. Data artifacts and data cleaning

The main data problems one can find in Load Chop observations are:

- **Standing Waves:**

- Optical standing waves can be present in the data, with typical periods depending on the band used (see this [report](#) for more details). Those baseline modulations are usually enhanced in the presence of non-negligible continuum, and can become really severe on planet observations. For these standing waves, the `fitHifiFringe` task usually does a pretty good job (see [Chapter 12](#)).
- Additionally, bands 6 and 7 are affected by a peculiar Electrical Standing Wave (ESW) that forms behind the mixer in the IF (Intermediate Frequency) amplification chain. This modulation is not sinusoidal in nature and can only be dealt with by `fitHifiFringe` in case of very simple isolated lines (i.e. narrow, without wings). The task `hebCorrection` is available to correct spectra by fitting models of the ESW. Although not every instance of ESW is well-represented by the available models, the great majority of observations are much improved (details can be found here [hebCorrection](#)). We also recommend [Section 12.4](#). Since HIPE 13.0, prepared ESW solutions have been stored in the HIFI calibration, and are now applied automatically by the pipeline task `doHebCorrection`. The same task can remove an already-applied correction.

- **Spurious spectral features:**

- There are two main sorts of spurious features in the HIFI data: narrow spur lines and saturation (probably a very broad form of the narrow spur). While the HIFI pipeline does its best to automatically detect and flag those, it is possible to fine-tune this data flagging with the `flagTool` task (see [Section 11.5](#)).

- **Residual baseline slopes/structure:**

- In addition to the spurious features, it is still possible to have residual baseline structure, usually in the form of a slope or a more complex shape. These residual can be corrected using polynomial fits to the baseline with the `fitBaseline` task [see [Chapter 13](#)]. This kind of problems is particularly strong in Load Chop mode without a reference. Although all these options were strongly discouraged, it is possible to find such data in the archive (part of them belonging to the Performance Verification Phase) so very special care needs to be taken if you want to exploit such datasets (see [Section 2.5.4.4](#)).

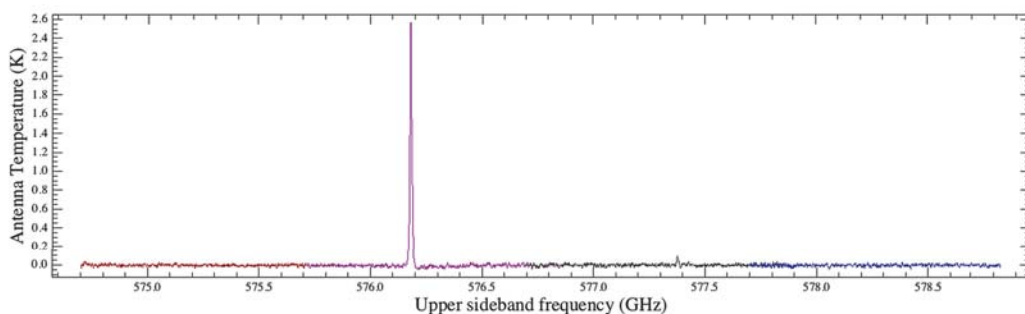
### Flagging bad data

It is important to underline that Level 2 data are averaged data mixing potentially good and bad spectra. Hence, it is wise to carefully check the quality of the data at Level 1. When inspecting these Level 1 data, if you detect weird data (e.g. weird baseline, jump between WBS subbands, spurious spectral features) you can flag them (see [Section 11.5](#)). After flagging the data, you will then need to re-pipeline the observation.

### Correcting for standing waves

Calibrated observations taken with Load Chop (with OFF reference) modes in beamsplitter Bands 1, 2, and 5 usually show clean spectra. If any residual waves are present in the Level 2 products, they have the shape of pure sine waves and can be subtracted using the `fitHifiFringe` task in HIPE (see [Chapter 12](#)). Observations in diplexer Bands 3 and 4 often show residual waves generated in the diplexer rooftop which are not pure sine waves. Amplitudes increase strongly toward the IF band edges. Although `fitHifiFringe` only fits sine waves, fits to the diplexer waves can be approximated using multiple sine waves, typically around 600 MHz. Note that the approximation is not as good at the IF band edges. Also, in the HEB bands 6 and 7 (especially short, non-DBS), observations show rather strong electrical standing wave residuals in Level 2 spectra. In HIPE 13.0 onwards, they are corrected automatically in the pipeline by the task `doHebCorrection` based on prepared solutions stored in the HIFI calibration product. Not every dataset is perfectly corrected, unfortunately. We're making an effort to improve the correction where judged necessary. It is possible for you to remove the applied correction with `doHebCorrection`, and attempt a correction yourself. For instructions on how to run the task yourself, see [Chapter 12](#).

Even if the observation we consider here is clean, let's apply `fitHifiFringe` on it. Select the observation `obs` and double-click on the `fitHifiFringe` task. Let's work on the WBS-H-USB product.



**Figure 2.50. WBS-H-USB Level 2 spectrum.**

Let's zoom a bit:

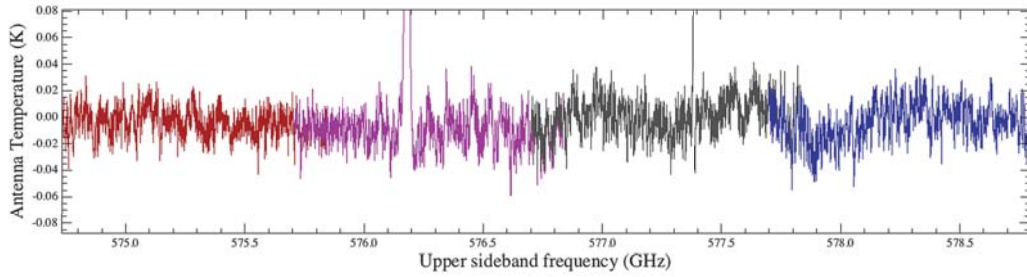


Figure 2.51. Zoom on WBS-H-USB Level 2 spectrum.

As a first guess, we might see at least weak standing waves with periods of 100 MHz and 1 GHz. Hence, let's tune the `fitHifiFringe` task as follows with 3 fringes and a typical period of 1000 MHz:

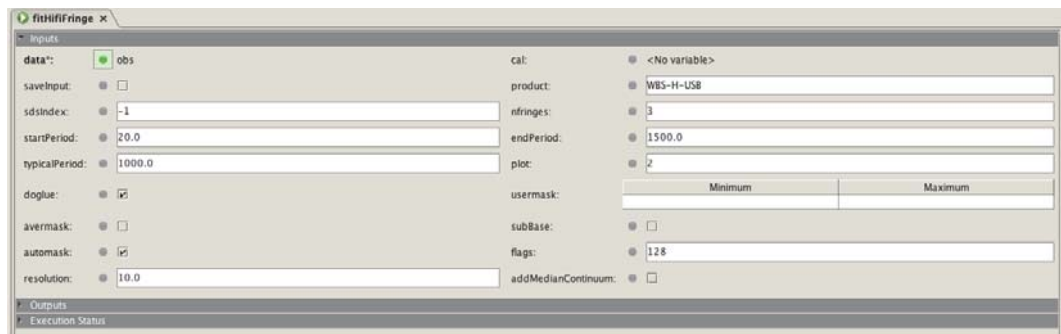


Figure 2.52. `fitHifiFringe` task menu.

The task finds 3 fringes:

period MHz	amplitude K	amplitude %baseline	phase degrees	chisq	delta_chisq
750.014	0.003	-85.777	169.014	0.068	0.056
564.747	0.003	-73.529	144.113	0.065	0.044
62.080	0.002	-52.492	334.688	0.063	0.024

And the output spectrum is:

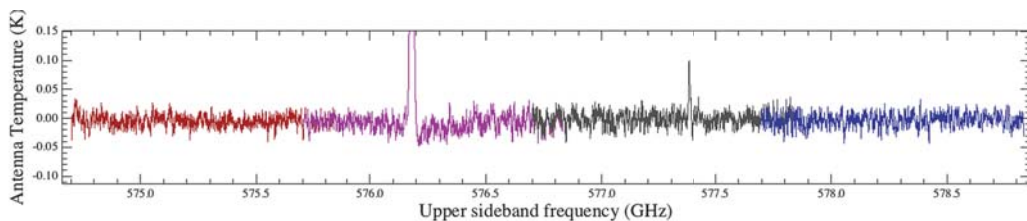


Figure 2.53. Zoom on WBS-H-USB Level 2 spectrum after `fitHifiFringe`.

Apply the same method, if necessary, to all products.

### 2.5.4.4. How to deal with Load Chop observations without sky reference

Observations performed **without any sky reference** very likely suffer from enhanced ripples, untreatable continuum, or at least a continuum modified by that of the cold load, etc. The impact can be



different for SiS and HEB bands: in HEB your data may be usable without any big trouble since data may be dominated by the ESW (see [Section 2.5.4.3](#)).

If your observation has been performed **without any sky reference** (e.g. LoadChopNoRef CH+ observation band 3a of DR21 (Obsid 1342180551), case shown hereafter), you very likely have strong ripples in Level 2:

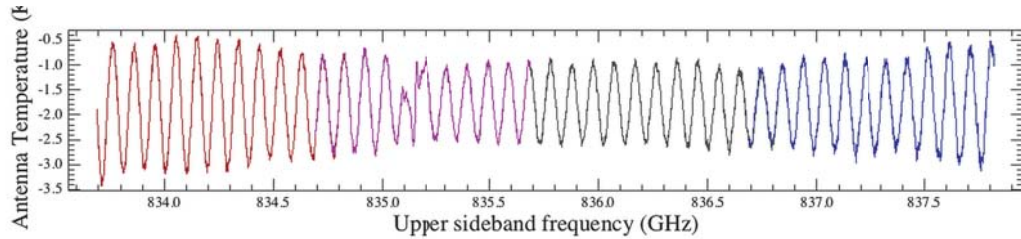


Figure 2.54. WBS-H-USB Level 2 spectrum for a Load Chop observations with no reference.

At least one 100 MHz fringe seems to dominate. In order to remove these ripples, you should apply the `fitHifiFringe` task with only one fringe in that case. One fringe of 96 MHz will be found, and the output of the task will be:

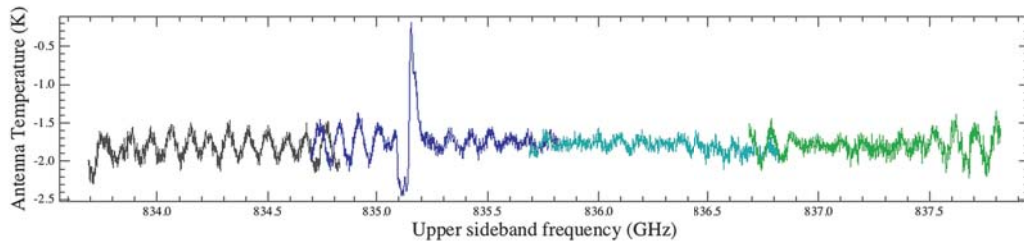


Figure 2.55. Same spectrum after having applied `fitHifiFringe` with `n = 1`.

Still some strong residuals are present. If 3 fringes are searched, the task will converge on 85, 95 and 150 MHz waves, but the result will not be satisfactory as the interesting line around 835.2 GHz will be spoiled:

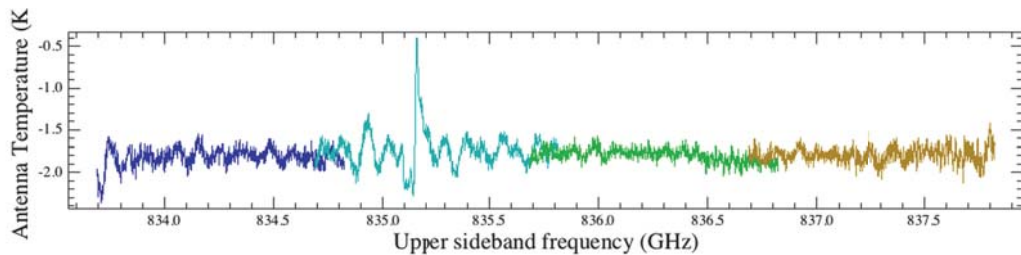


Figure 2.56. Same spectrum after having applied `fitHifiFringe` with `n = 3`.

You can still increase the number of fringes but you might completely remove the real lines ! In most cases, the final data will be of poor quality compared to the ones you could have obtained using the Load Chop mode with off reference.

## 2.6. Spectral Map Mode

Last updated: 1 April, 2015

## 2.6.1. Introduction

HIFI spectral mapping observations are performed in one of two modes: On The Fly (OTF) or Dual Beam Switch (DBS) Raster. OTF mapping involves taking data continuously as the telescope slews over the area to be mapped, while data is taken at fixed grid positions in DBS Raster maps. OTF maps are most commonly carried out using position switching as a reference scheme but frequency switching and load chop are also available.

The high efficiency of OTF mapping makes it the most commonly used HIFI mapping mode. However, the nature of OTF mapping, where the telescope can be slewing at speeds greater than the settling time of the instrument, makes it more prone to baseline issues than DBS raster mapping. DBS Raster maps tend to have more stable baselines, due to the double-differencing used in the DBS reference scheme, but require greater observation times. As a consequence, OTF maps are not recommended for continuum studies while DBS Raster maps are most suitable for small maps and observations in the HEB bands (6 and 7). Nonetheless, both mapping modes can be affected by standing waves and other baseline distortions in addition to spurs. Clean-up of the Level 2 data prior to re-gridding into spectral cubes is always recommended.

In this cookbook we use the observation `obsid= 1342248770`, which is an OTF map of S 140 IRS 1 taken in band 5a with position switching used for the OFF (sky) reference. The strongest line seen in the data is the CO (10-9) line and the Level 2 data shows signs of baseline drift in the WBS, which will require clean-up before re-gridding. We also use a DBS Raster map observation, also of S 140 IRS 1, with `obsid= 1342205481`. This is a 3-by-3 map taken in band 7 and targeted on the C+ line, which is contaminated by emission in a chop position.

## 2.6.2. How Spectral Map Mode observations are taken

### 2.6.2.1. Observing Principles

#### OTF maps

OTF maps were taken by slewing the telescope over the region to be mapped whilst continuously taking data. The telescope slews "up" one scan leg and "down" the next while data is dumped every four seconds, giving a grid of read-outs, as shown in [Figure 2.57](#), which is taken from the AOT release notes. The map is repeated as necessary to meet the noise requirements specified in HSpot starting again from the same position on the sky. The sky reference (if used) is taken at the end of a scan leg by a position switch, and at either fixed coordinates or an offset, as specified in HSpot, and is "shared" by several source measurements for calibration.

#### Dual Beam Switch (DBS) Raster maps

DBS Raster maps are created by taking measurements on a fixed grid, and using the DBS method of chopping 3' either side of the target position to observe a sky reference. DBS Raster maps produce a more accurate continuum than OTF maps, at the expense of a greater time overhead, and are useful in bands 6 and 7, which require fast chopping references to mitigate the longer settling times of the Hot Electron Bolometers (HEBs). DBS observations were carried out with chop speeds of 0.125 Hz (slow chop) or up to 4 Hz (fast chop). Fast chop observations produce a more accurate continuum, and more accurate measurements of broad lines. It has also been found that fast chop observations are easier to correct for electrical standing waves in bands 6 and 7.

#### Sky sampling in HIFI spectral maps

HIFI spectral maps could be carried out with sky samplings of 10", 20", 40", half-beam sampling or Nyquist sampling. For DBS Raster maps, this means that each grid point was separated by this spacing. For OTF maps, this means that the scan legs were separated by this spacing.

Until OD 419, "Nyquist sampling" was defined as  $HPBW/2$  (half-beam sampling). This resulted in a slight under-sampling in Nyquist terms. From OD 419, the Nyquist definition was corrected to give a

sampling between readouts of a factor approximately 1.2 less than the sampling used in the half-beam case. For more details see this [technical note](#).

You need to be aware of this because observations that were planned to have Nyquist beam spacing prior to OD 419 still have the HTP and cube metadata item *nyquistSampling* set to *true* even though the actual sampling is half-beam.

### Noise definition in HIFI spectral maps

When the observation was planned in HSpot, the optimum observing sequence (e.g., number of readouts, number of times the sky reference was observed, number of repetitions of the map) that fit the input setup was found and the resulting noise in the observation was calculated. The noise calculated in a regridded cell at the requested spatial sampling is the best value to compare with the noise predicted by HSpot.

## 2.6.2.2. Reference Schemes

### OTF maps

The reference schemes used to calibrate OTF maps are Position Switch, Frequency Switch and Load Chop, which are described in the point mode cookbooks, see [Section 2.3](#), [Section 2.4](#) and [Section 2.5](#), respectively. Position Switching, where a sky reference up to two degrees away from the map centre is used, is the most commonly used reference scheme for OTF maps as it is the most time-efficient. In the case that a line-free region cannot be found within 2 degrees of a target then Frequency Switching, in which the telescope remains on target but a different Local Oscillator (LO) setting is observed, typically would be the next choice of reference scheme. The Load Chop reference scheme, in which the telescope remains on target but internal Hot and Cold loads are used as reference, would be used in the case that the region is very extended and line-rich so that neither Position-Switching or Frequency Switching were possible. For bands 6 and 7, which have longer settling times than bands 1-5 and so require fast (chopping) references, Frequency Switching or Load Chop could be the reference scheme of choice.

Irrespective of the reference scheme used, OTF maps are prone to baseline issues as the slewing speed of the telescope can be faster than the settling time of the instrument. However, Frequency Switching and Load Chop OTF observations are particularly badly affected as the optical path between the source and references differ. During Performance Verification this was found to be such an issue that it was recommended to always perform Frequency Switch and Load Chop observations with an additional sky reference. Nonetheless, some OTF Frequency Switch and OTF Load Chop mapping observations have been carried out without a sky reference, and these observations can suffer from severe baseline distortions, which degrade the data quality. The baselines can be improved with much correction but the process typically only works for very strong and narrow lines as faint lines and broad line wings tend to be lost in the correction.

### DBS Raster maps

DBS raster maps are calibrated using the DBS reference scheme, which uses the chopper to observe fixed sky reference positions 3' either side of the target position. The reader is directed to [Section 2.2](#) for more detail. This method of double differencing generally gives good baseline stability. It is always a good idea to check for contamination in the reference spectra of DBS observations (see [Section 2.6.4.2](#)).

## 2.6.2.3. Observing Timeline

### OTF maps

OTF maps are performed continuously with data read out as the telescope scans "up" one scan leg and "down" the next. Data is dumped (or read) every four seconds but not in the turn between scan legs. In [Figure 2.57](#) you can see that the map read-outs (denoted by blue dots) form a zig-zag pattern. This is a

consequence of a timing mismatch between the commanding of the pointing of the telescope and the commanded times at which data is read out. In order to ensure that the requested sensitivity of the map is attained over the entire map, an extra read out is added to each scan leg in the map. In [Figure 2.57](#) the requested map area is shown shaded in grey and the approximate beam size is indicated by the blue circle.

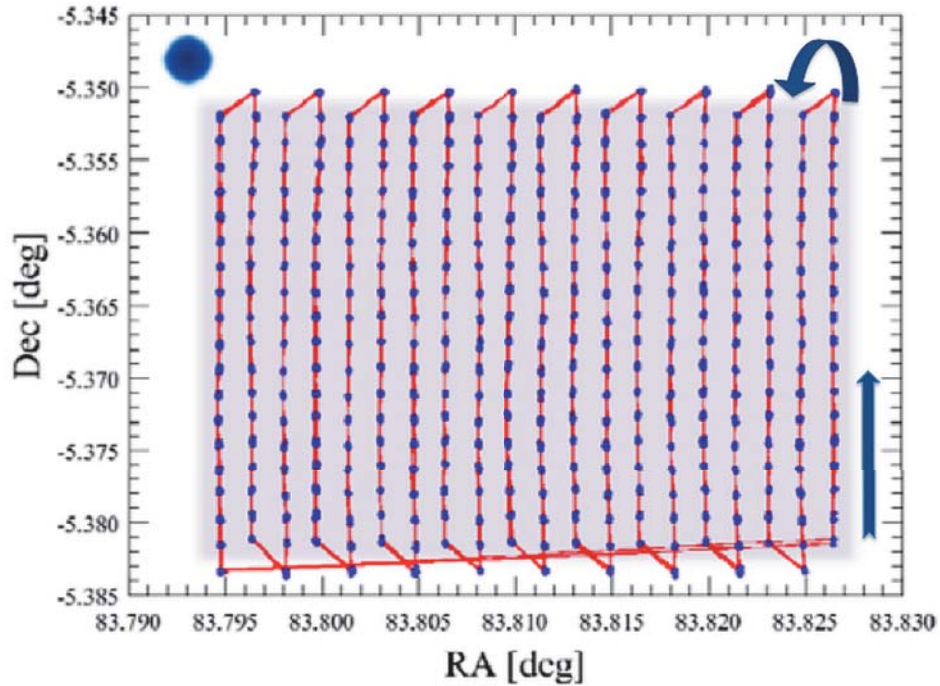


Figure 2.57. Positions of read-outs of science data in an OTF (position switch) observation

In the case of OTF maps taken using the frequency switching or load chop, reference schemes data is still read-out every four seconds. However, this includes the integrations used for the reference, i.e., the alternative frequency or the measurement of the loads. As a result, the sampling on the sky of scientific data does not form a regular grid of points as in the case of position switch OTF maps, and this is illustrated in [Figure 2.57](#) below (also taken from the AOT Release notes). Note, however, that the HIFI pipeline and the `doGridding` task used in HIPE to create spectral cubes convolve the read-outs onto a regular grid (weighted according to the distance of the position the data was taken at from the position taken as the centre of the pixel).

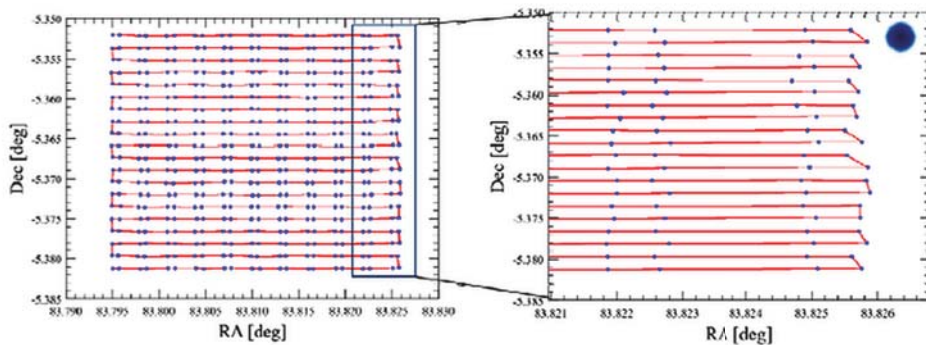


Figure 2.58. Positions of read-outs of science data in an OTF (load chop) observation, taken at 90 degree position angle

## DBS Raster maps

DBS raster maps are performed as a grid of DBS observations and the grid is repeated until the requested noise is expected to be achieved. The timeline for a single DBS observation is illustrated in [Figure 2.2](#) in the Point mode DBS cookbook. As the telescope does not continuously slew during DBS Raster map observations, the grid of points is regular and does not display the zig-zag seen in OTF maps.

## 2.6.3. Inspecting Spectral Map Mode data

### 2.6.3.1. Data Structure

The first step in any data reduction is to take a look at what you have. For all mapping observations the data structure is the same and contains:

- data including calibration observations at Level 1
- calibrated spectra at Level 2
- a copy of the Level 2 HTP and spectral cubes created from the Level 2 data at Level 2.5
- Browse Products to give a quick overview of the data at the top level
- OFF data processed up to an equivalent Level 2 by the pipeline

The Browse Products are created in the same way irrespective of the type of map. Level 2.5 spectral cubes are created using the default options of the `doGridding` task with no data clean-up and should also be considered as a type of quick look product. Differences in the data structure and content due to the type of map are seen below Level 2.5. We describe the contents of each Level in more details below. We use an OTF Position Switch map (obsid= 1342248770) and a DBS Raster map (obsid= 1342205481) to illustrate the structure of these types of data.

To get the observations from the HSA into HIPE:

```
# OTF map
obs=getObservation(1342248770, useHsa=True, save=True)
#
# Raster map
obs_raster = getObservation(1342205481, useHsa=True, save=True)
```

The observations are saved into your *MyHSA* pool.

You will see `obs_otf` and `obs_raster` listed as variables in the *Observations* section of the *Variables* panel in HIPE. Double-click on one of these to open it in the *Observation Viewer*. You can see the Observation Context for the OTF map viewed in this way in [Figure 2.59](#). The Observation Context summary is to the top left and contains identifying information about the observation, such as observation number (obsid), observation date, observing mode, etc.

### 2.6.3.2. Browse Product

The first thing you can do to get a sense of the data is to look at the Browse Product. This is the small image you see at the top right in the Observation Context summary, and is the same as the postcard that you can see in the Herschel Science Archive. To view the image, click on it. You can zoom in and out using the mouse scroll wheel (or the trackpad equivalent). Click again on the image to exit the viewer.



Figure 2.59. OTF map 1342248770 viewed in Observation Viewer

The mapping browse product shows a sets of map-averaged Level 2 spectra for each subband with the integrated map, also for that subband, to the right. The sets of images are arranged in order of increasing frequency, i.e., subbands 1-4 from top left to bottom right for bands 1-5, and subbands 4 to 2 from top left to bottom left for bands 6 and 7. The AOR label and observation number are used to title the plot. The observing mode, source name, requested RA and dec are below the plot title.

For each spectrum, the upper and lower axes of the plots show the LSB and USB frequency scale, respectively. The y-axis is scaled to a factor 1.2 times the peak value in the spectrum excluding data flagged as a spur. This allows you to see that the data is impacted by a spur but still see other features in the spectrum.

To the right of each spectrum are integrated maps that are created with no correction done for any baseline issues. In the cases that the baseline suffers from drifts and/or standing waves, the continuum will dominate the map. The right and bottom x- and y- axes show the RA and declination, respectively, while pixel coordinates are shown on the auxiliary axes. The colour scale used for the image is 'heat' and the intensity scale used is 'ramp' so the strongest emission in the map should appear white.

For more information about browse products, see [Section 3.4](#).

### 2.6.3.3. Level 2.5

The Level 2.5 mapping product contains a copy of the Level 2 HTPs (see [Section 2.6.3.4](#) below) and spectral cubes that have been created by the HIFI pipeline. The `doGridding` task is used by the pipeline to convolve the science data in the Level 2 `HifiTimelineProducts` (HTPs) onto a regular grid, giving spectral cubes with the grid spacing that was used in the observation; observations of Solar System Objects (SSOs) are gridded in a coordinate system comoving with the target. The spectra in the cubes are on the  $T_A^*$  temperature scale. Frequencies are Doppler-corrected to the LSR frame (i.e., source velocity relative to the LSR is *not* corrected for) except for SSOs, which are Doppler-corrected to their rest frame. See [Section 15.4](#) for more information about the details of how `doGridding` works.

Spectral cubes are created for each spectrometer, polarisation, and sidebands used in the observation and the products are organised as follows:

- first, we find a copy of the Level 2 HTPs
- then, we find a `cubesContext`, which is a special `Context` used to store, and allow you to easily explore cubes. The Level 2.5 `cubesContext` is named `cubesContext`. If the map was made with a non-zero position angle, as is the case for the `obsid` used in this cookbook, you will also find a `cubesContext` named `cubesContextRotated`. The `cubesContext` contains cubes gridded

assuming no rotation angle and so is displayed with the traditional North up and East to the left, while *cubesContextRotated* contains cubes gridded with the position angle that was entered into HSpot and thus with a mathematically correct convolution.

- Inside both *cubesContext* and *cubesContextRotated* are *cubeContexts* for each of the spectrometers and polarisations used in the observation for both lower and upper sidebands. Each context is labelled accordingly, e.g., the *cubeContext* for the lower sideband, horizontal polarisation of the WBS gridded without taking rotation into account is called *cubesContext\_WBS-H\_LSB*.
- The final level contains one spectral cube with the subbands stitched (for WBS) for a given spectrometer, polarisation and sideband. The cube is labelled by spectrometer, polarisation, and sideband and have an index numbered with 1, e.g., the spectral cube for the WBS-H USB is called *cube\_WB-S\_H\_USB\_1*. The HRS subbands are stitched **only** if they overlap in frequency (`doStitch` is called using `fillGaps=False`) in order to avoid NaNs in the cubes. The label will follow the same convention as with the WBS, e.g. *cube\_HRS\_H\_LSB\_1*. Otherwise, if the HRS subbands do not overlap (as it is the case with this observation), the pipeline will produce one cube per spectrometer, polarisation, and sideband, for each subband, e.g. *cube\_HRS\_H\_LSB\_2*, for subband 2.

We emphasise again that the cubes at Level 2.5 are created "as is" from the Level 2 data and should be considered as a type of quick look product. In order to obtain optimal results, Level 2 data should be "cleaned" by correcting baselines and flagging any remaining spurs prior to regridding into spectral cubes.

The spectral cubes are products of type *SimpleSpectralCube* and can be quickly viewed in the Spectrum Explorer by clicking on the cube name, or can be fully viewed and manipulated by opening the Spectrum Explorer completely with a double-click. If you previously opened a cube with something other than the Spectrum Explorer then you may need to right-click on the cube and select Open With → Spectrum Explorer.

The quick view option of the Spectrum Explorer window will appear beside the Observation Context tree and allows you to quickly navigate among the cubes in the Observation Context but has limited functionality. Opening the Spectrum Explorer fully (so that it takes over the entire *Editor View*) gives you full access to all of the display and toolbox options in the Spectrum Explorer.

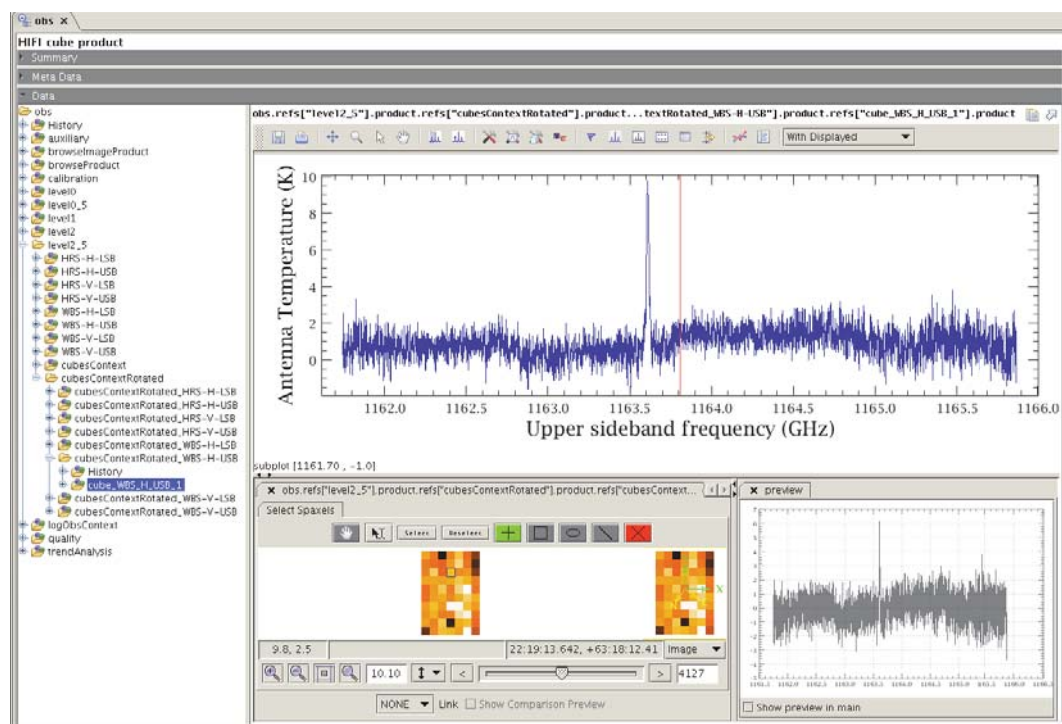


Figure 2.60. A quick look at a spectral cube in the Spectrum Explorer

When manipulating a cube in the Spectrum Explorer, it is recommended to create a variable to work on. The reason for this is that some of the tasks in the toolboxes associated with the Spectrum Explorer modify the data. But, to protect your original data from being overwritten, HIPE will not allow data still in an Observation Context to be modified. To create a variable of a cube you can drag the cube name from the Observation Context tree to the *Other Variable* section of the *Variables View*, where you will probably wish to rename the variable to something shorter by clicking on the variable name, deleting the default variable name, and typing your preferred choice. You can also right-click on the cube name, a menu open, then select *Create Variable*. In the command line the same is done with:

```
cube = obs.refs["level2_5"].product.refs["cubesContextRotated"].product.refs\
["cubesContextRotated_WBS-H-USB"].product.refs["cube_WBS_H_USB_1"].product
```

The use of the Spectrum Explorer with Spectral Cubes is described later in this manual in [Section 6.2.2](#) and more generally in [How to display the spectra in cubes](#) in the Herschel Data Analysis Guide.

### 2.6.3.4. Level 2

The Level 2 HifiTimelineProduct (HTP) contains only science data (no calibration or reference data) that has been converted to antenna temperature and sky frequency as described for cubes above. It is recommended to inspect the data at Level 2 to identify whether there exist any baseline issues in the data that could affect the quality of the Level 2.5 cube, necessitating some clean-up prior to re-gridding.

At Level 2, one finds HTPs for each spectrometer and polarisation used in the observation for both upper and lower sidebands. Inside each HTP is at least one *box*, which is a construct used to compactly store the datasets in the observation. For OTF observations, there is one dataset per scan leg, numbered from 0001. Each dataset contains one spectrum per read-out in that scan leg, where these are numbered from zero. For DBS Raster maps there is one dataset for each position in the map, and each dataset contains one spectrum, which is the average of all the data taken at that position.

The OTF Position Switch observation considered in this cookbook has seven datasets within the box, corresponding to the seven scan-legs in the map, see [Figure 2.60](#). Each dataset contains eighteen spectra, which are convolved by `doGridding` to produce the spectra in the nine pixels per scan-leg in the map.

The HTP is extracted from the Observation Context to work on by dragging the name of the HTP to the *Variables View* or, in the command line by:

```
htp = obs.refs["level2"].product.refs["WBS-H-USB"].product
```

The datasets are extracted in a similar way, here we extract the first dataset in the OTF observation:

```
ds1 = htp.refs["box_001"].product["0001"]
```

If desired, one can extract one of the spectra within the dataset to work on, say the tenth spectrum. In the command line this is done by:

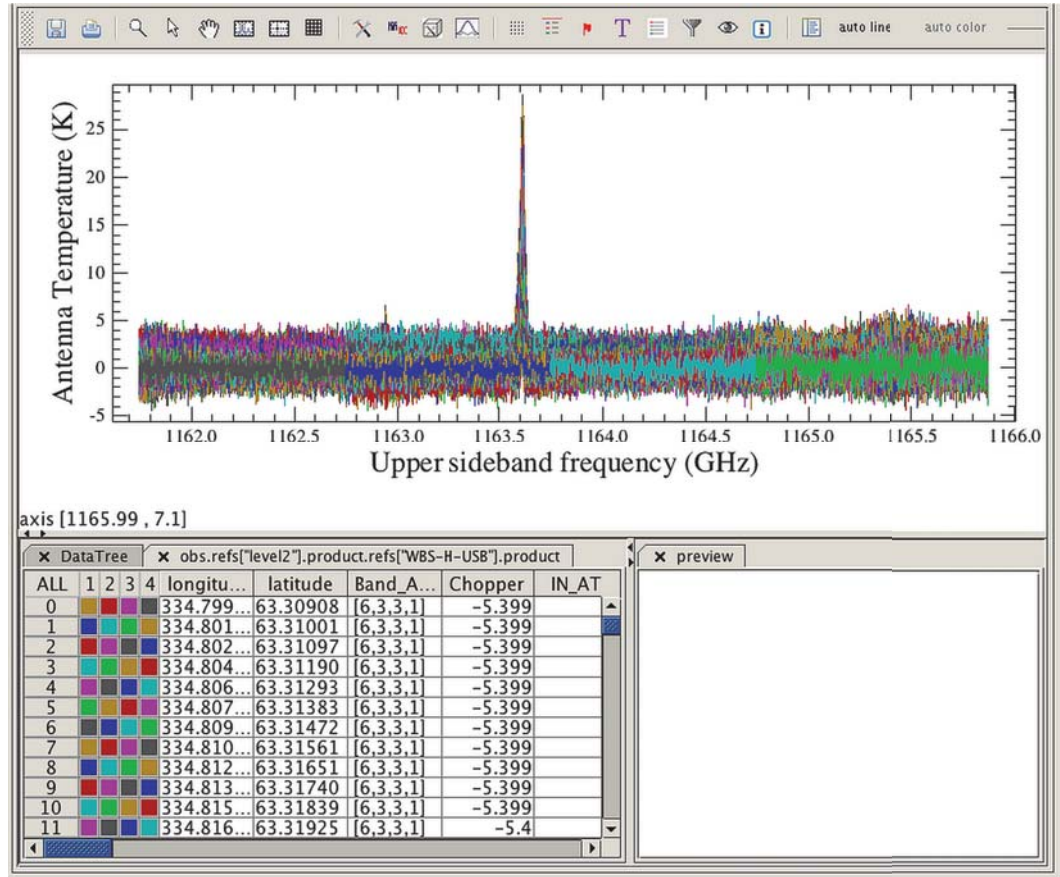
```
# recall spectra within datasets are numbered from 0
spectrum9 = extract(selection=[9], segments=[1, 2, 3, 4], ds=ds1)
```

You can look at each dataset individually by opening it in Spectrum Explorer with a double click, or using the Open With → Spectrum Explorer option after right-clicking on the dataset number, see [Section 6.2.1](#) later in this manual, and the section [How to display spectra](#) of the *Herschel Data Analysis Guide* for how to use the Spectrum Explorer to view spectra. Alternatively, you can look at all the spectra in the HTP by opening the HTP itself in Spectrum Explorer using the Open With → Spectrum Explorer option after right-clicking on the HTP name. This will open the HTP in the Data Tree, which



allows you to maintain the product structure within the HTP thereby allowing you to keep track of spectra within scan-legs. See [Section 6.2.3](#) later in this manual, and the section [How to display spectra](#) of the *Herschel Data Analysis Guide* for how to use the Spectrum Explorer and the Data Tree to view HTPs.

In [Figure 2.61](#), we plot all the spectra in the Level 2 WBS-H-USB HTP of the OTF observation in the Spectrum Explorer. The baseline level varies on a level approaching 10 K over the entire map and the spectral cube would benefit from baseline correction prior to gridding.



**Figure 2.61.** The spectra in the Level 2 HTP WBS-H-USB in the OTF observation plotted in Spectrum Explorer, the extent of baseline drift in the observation can be seen.

One particularly useful feature of the Spectrum Explorer for mapping data is the ability to inspect the locations of the readouts within the map. This is done by plotting all the data in the HTP, and then selecting the raster view of the Spectrum Explorer, followed by the location option, as described in [Section 6.2.3](#). We discuss this further in the next section.

### 2.6.3.5. Level 1

Science data at Level 1 is in units of Intermediate Frequency (IF) [MHz] and on the  $T_A$  temperature scale [K]. The data is organised into HTPs for each spectrometer and polarisation used in the observation and still contains calibration and reference (off) integrations. The best way to identify which are the science data is to use the *summary table*, which can be found in each HTP. You can double click on the summary table to open it with the *Dataset Viewer* and you can also extract it from the Observation Context in the same way you can a spectrum:

```
WBS_H_L1_SummaryTable=obs.refs["level1"].product.refs["WBS-H"].product["summary"]
```

Here we investigate what can be seen in the Level 1 data for the OTF observation. We show the summary table for the WBS-H at Level 1 in [Figure 2.62](#)

Meta Data												
None												
Table Data												
Index	dataset	type	Bbid	bbNumber	isLine	isHrs	isWbs	fullName	LoFrequency [GHz]	LO-Throw [GHz]	start	length
0	1	tune	6613	1	false	true	true	WBS attenuators block	1157.9085	0.0	0	3
1	2	comb	6004	1	false	true	true	WBS Zero Comb	1157.9085	0.0	3	2
2	3	hc	6005	1	false	true	true	HIFI Calibrate_hot_cold	1157.9085	0.0	5	2
3	4	science	6021	1	false	true	true	HIFIContOffIntegration	1157.9085	0.0	7	5
4	5	science	6022	1	true	true	true	HIFIContOnIntegration	1157.9085	0.0	12	18
5	6	science	6021	2	false	true	true	HIFIContOffIntegration	1157.9085	0.0	30	5
6	7	science	6022	2	true	true	true	HIFIContOnIntegration	1157.9085	0.0	35	18
7	8	science	6021	3	false	true	true	HIFIContOffIntegration	1157.9085	0.0	53	5
8	9	science	6022	3	true	true	true	HIFIContOnIntegration	1157.9085	0.0	58	18
9	10	science	6021	4	false	true	true	HIFIContOffIntegration	1157.9085	0.0	76	5
10	11	science	6022	4	true	true	true	HIFIContOnIntegration	1157.9085	0.0	81	18
11	12	science	6021	5	false	true	true	HIFIContOffIntegration	1157.9085	0.0	99	5
12	13	science	6022	5	true	true	true	HIFIContOnIntegration	1157.9085	0.0	104	18
13	14	science	6021	6	false	true	true	HIFIContOffIntegration	1157.9085	0.0	122	5
14	15	science	6022	6	true	true	true	HIFIContOnIntegration	1157.9085	0.0	127	18
15	16	science	6021	7	false	true	true	HIFIContOffIntegration	1157.9085	0.0	145	5
16	17	science	6022	7	true	true	true	HIFIContOnIntegration	1157.9085	0.0	150	18
17	18	science	6021	8	false	true	true	HIFIContOffIntegration	1157.9085	0.0	168	5

Figure 2.62. The summary table for the WBS-H at Level 1 in the OTF observation.

The first three datasets in the HTP are calibration integrations; a tuning for the LO, a comb measurement to use for frequency calibration, and a measurement of the hot and cold loads ("hc") that will be used to calculate the bandpass used in the intensity calibration. The science (integrations in the map) data are labelled "science" in the *type* column of the summary table, and are listed as "true" in the *isLine* column. The off measurements are also labelled "science" in the *type* column but are listed as "false" in the *isLine* column. Off spectra can be expected to contain standing waves (it is the subtraction of these from the on map integrations that removes the standing waves in the final science data).

You can compare the positions of each phase (on, off, calibration) in the observation from the Level 1 data using the location option of the raster mode in the Spectrum Explorer. In Figure 2.63, we show what is seen in this view initially when inspecting the locations of all spectra in the WBS-H. Zooming in (by drawing a box around the points) on the scattered points to the bottom right, and hovering the mouse cursor over those points informs us that these are spectra 0-2 in dataset 1. These are the three tuning measurements that were taken as the instrument was slewing to the map coordinates.

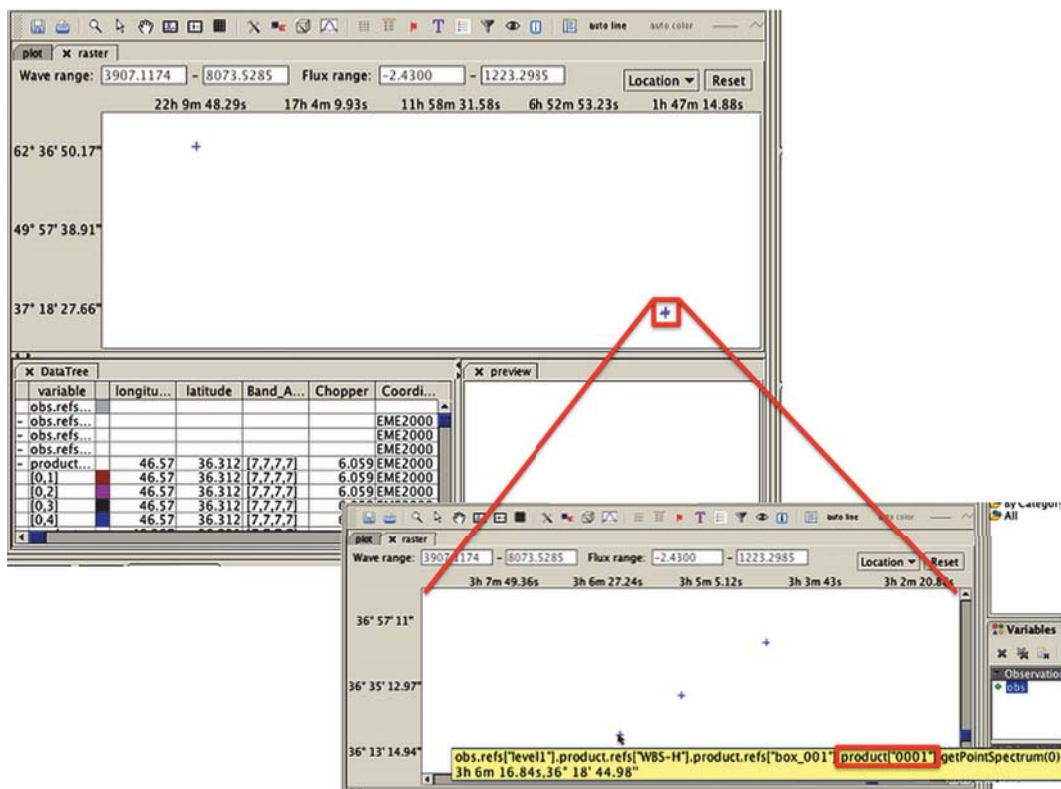
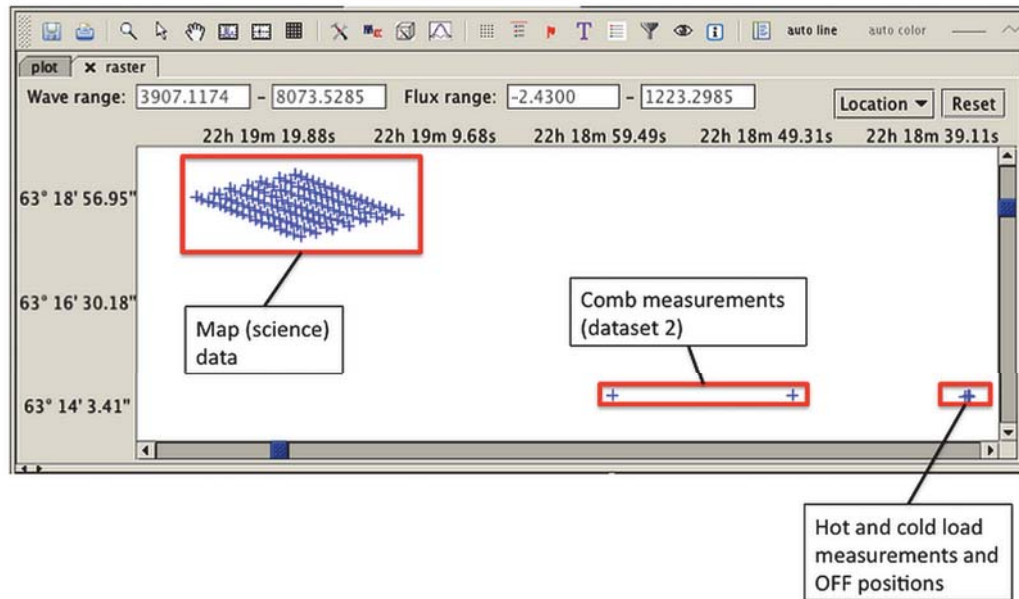


Figure 2.63. The location option in Spectrum Explorer's raster mode for the WBS-H at Level 1. Hovering the mouse cursor over the three points to the bottom right (see inset), we see that these are the spectra in dataset 1 (rectanged in red).

By zooming back out (right click on the plot) and then zooming in on the map region, we see the map (science) data and the off positions. By hovering the mouse over these and comparing with the dataset indices in the Summary Table, we can identify the tuning measurements and the hot-cold load measurements (see [Figure 2.64](#)). In this observation the off positions were all taken at the same location (within the pointing uncertainty).



**Figure 2.64.** The location option in Spectrum Explorer's raster mode for the WBS-H at Level 1, zooming in (twice) on the map region.

Level 1 data for DBS Raster observations take the same form as for point mode DBS observations (see [Section 2.2.4.1](#)). In [Figure 2.65](#), we show the Summary Table for the WBS-H of observation 1342205481. Science observations, labelled as *Science*, contain the average of the reference subtracted target position spectra. The science observations, labelled as *isLine* "true", were taken with the telescope on source, while the science observations, labelled as *isLine* "false", were taken with the telescope at the nod position. There is a pair of *on source* and *on nod* science observations for each point in the map for as many repeats as were made of the map. For example, in [Figure 2.65](#), we see that there are nine such pairs, as the map was observed only once, but for a 3-by-3 raster map observed with two repeats, we would see 18 pairs of science data at Level 1. Tuning and calibration observations of the comb and hot-cold load measurements are labelled as described above.

obs_dbs.refs["level1"].product.refs["WBS-H"].product["summary"]									
Index	dataset	type	Bbid	bbNumber	isLine	isHrs	isWbs	fullName	LoFrequency
0	1	tune	6613	1	false	true	true	WBS_attenuators_block	1897.695
1	2	comb	6004	1	false	true	true	WBS_Zero_Comb	1897.695
2	3	hc	6005	1	false	true	true	HIFI_Calibrate_hot_cold	1897.695
3	4	hc	6005	2	false	true	true	HIFI_Calibrate_hot_cold	1897.695
4	5	science	6042	1	true	true	true	HIFIFastChopOnIntegration	1897.695
5	6	science	6042	2	true	true	true	HIFIFastChopOnIntegration	1897.695
6	7	science	6042	3	true	true	true	HIFIFastChopOnIntegration	1897.695
7	8	comb	6004	2	false	true	true	WBS_Zero_Comb	1897.695
8	9	hc	6005	3	false	true	true	HIFI_Calibrate_hot_cold	1897.695
9	10	hc	6005	4	false	true	true	HIFI_Calibrate_hot_cold	1897.695
10	11	science	6043	1	false	true	true	HIFIFastChopOffIntegration	1897.695
11	12	science	6043	2	false	true	true	HIFIFastChopOffIntegration	1897.695
12	13	science	6043	3	false	true	true	HIFIFastChopOffIntegration	1897.695
13	14	science	6043	4	false	true	true	HIFIFastChopOffIntegration	1897.695
14	15	science	6043	5	false	true	true	HIFIFastChopOffIntegration	1897.695
15	16	science	6043	6	false	true	true	HIFIFastChopOffIntegration	1897.695
16	17	comb	6004	3	false	true	true	WBS_Zero_Comb	1897.695
17	18	hc	6005	5	false	true	true	HIFI_Calibrate_hot_cold	1897.695
18	19	hc	6005	6	false	true	true	HIFI_Calibrate_hot_cold	1897.695
19	20	science	6042	4	true	true	true	HIFIFastChopOnIntegration	1897.695
20	21	science	6042	5	true	true	true	HIFIFastChopOnIntegration	1897.695
21	22	science	6042	6	true	true	true	HIFIFastChopOnIntegration	1897.695
22	23	science	6042	7	true	true	true	HIFIFastChopOnIntegration	1897.695
23	24	science	6042	8	true	true	true	HIFIFastChopOnIntegration	1897.695
24	25	science	6042	9	true	true	true	HIFIFastChopOnIntegration	1897.695
25	26	comb	6004	4	false	true	true	WBS_Zero_Comb	1897.695
26	27	hc	6005	7	false	true	true	HIFI_Calibrate_hot_cold	1897.695
27	28	hc	6005	8	false	true	true	HIFI_Calibrate_hot_cold	1897.695
28	29	science	6043	7	false	true	true	HIFIFastChopOffIntegration	1897.695
29	30	science	6043	8	false	true	true	HIFIFastChopOffIntegration	1897.695
30	31	science	6043	9	false	true	true	HIFIFastChopOffIntegration	1897.695

Figure 2.65. The summary table for the WBS-H at Level 1 in the DBS Raster observation.

## 2.6.4. Spectral Map Mode Data Reduction

### 2.6.4.1. Data artefacts and data cleaning in spectral maps

Mapping observations suffer from the following types of data artefacts:

- Optical standing waves: These can be corrected by using `fitHifiFringe`, see [Chapter 12](#) for more information.
- Electrical standing waves in bands 6 and 7: These are quite prominent in OTF maps due to the short stability time compared with time between reference observations. They can be corrected by using the `HebCorrection` task. See [Section 12.4](#) for more information.
- Baseline residuals: In the case of OTF Position Switch data, these are very commonly parabolic in shape. Baseline residuals can be corrected using the task `fitBaseline`. See [Chapter 13](#) for information on using `fitBaseline`.
- Spurs: These are usually flagged by the pipeline but if there is any feature in your data that you would prefer be ignored by post-processing tools, you can use the `flagTool` task. See [Section 11.5](#) for more information.
- OTF maps display a zig-zag: As described above in [Section 2.6.2.3](#).
- Striping: During Performance Verification, it was shown that there should not be significant striping once drift artefacts (baseline residuals) are removed. See [HIFI AOT Observing Mode Release and Performance Notes](#). As a consequence, maps of peak or integrated intensity should not suffer from striping, but continuum maps and maps in the HEB bands (bands 6 and 7) which have longer settling times than the SIS bands (bands 1-5) frequently do. For this reason, it is not recommended to use OTF maps for continuum studies.

All tasks mentioned above can be included in the Interactive Level 2.5 pipeline prior to the creation of cubes by dragging the task name from the *Tasks View* to the Interactive Pipeline panel in the HIFI pipeline GUI. See [Section 5.4.1](#) for more information.

The Level 2 data in the OTF observation above shows that the baseline levels have drifted by approximately 10 K over the map area during the observation. It is for this reason that OTF maps are

not recommended for continuum studies. The final map will benefit from correcting the data for this baseline drift, and this can be done in HIPE using `fitBaseline`. See [Chapter 13](#) for information on using `fitBaseline`.

In the case of this observation, where each subband shows slightly different baseline behaviour with subbands 1 and 4 showing more curvature than subbands 2 and 3 (see [Figure 2.61](#)), it is recommended to fit each subband individually. Using a second order Polynomial gives best results.

## 2.6.4.2. Assessing OFF data for contamination

### OTF maps

The smoothed and averaged off position spectra, uncorrected for the bandpass, for each spectrometer and polarisation used in the observation are available for inspection in the Calibration Context under Calibration → pipeline-out → Baseline.

These off spectra are only partially calibrated and not straightforward to understand. However, the OFF data is also processed up to an equivalent Level 2 by the pipeline, and have USB and LSB products. You can find the OFF positions for all spectrometers in the Calibration product in the Observation Context; choose calibration → pipeline-out → ReferenceSpectra.

For a fix target, you will find one OTF calibrated OFF spectrum per map. For a moving target, there will be several OTF calibrated OFF spectra per map, taken at different positions on the sky due to the SSO tracking.

The OFF positions spectra can be run through the standard tasks such as `fitHifiFringe`, `fitBaseline`, and `flagTool`, and will also be corrected from Electrical Standing Waves by default.

### DBS Raster maps

The pipeline calculates the difference in the two sky reference (chop) positions. If there is any emission (or absorption) in one of the chop positions it can be seen in this difference. The position averaged difference spectra for each spectrometer can be found in the *Calibration* Context in the Observation Context. Look under Calibration → pipeline-out → ReferenceSpectra. On fixed and moving targets, there is one OFF spectrum per raster position.

In [Figure 2.66](#) we show a Level 2 WBS-V-USB spectrum (box\_0001/0003) and the difference in the chop positions, as found in the *ReferenceSpectra* in the Calibration Context. We see that the absorption in the line in the Level 2 data is due to contamination in a chop position. The method to correct for contamination is highlighted in the point mode DBS cookbook (see [Section 2.2](#))

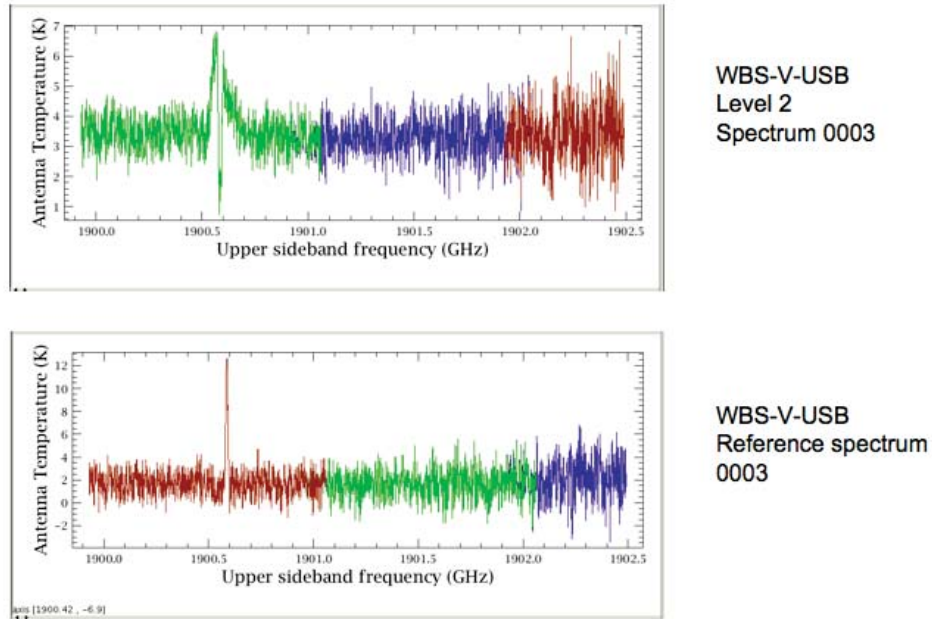


Figure 2.66. Contamination due to emission in chop position in the DBS Raster observation 1342205481.

### 2.6.4.3. Creating spectral cubes

The gridding of mapping data into spectral cubes is done with the `doGridding` task. The task can be run as part of the Interactive Level 2.5 pipeline, or as a stand-alone task. If you include `doGridding` in the Interactive Level 2.5 pipeline (possibly as the step after `fitBaseline`) then the `cubesContext` in the Level 2.5 product will be over-written with the new cubes generated. Note that when running the pipeline interactively like this, `doGridding` is only run once. So, if the map was performed at a non-zero position angle, only the rotated maps will be created. As a stand-alone task `doGridding` runs on Level 2 HTPs rather than Observation Contexts and will create a variable that is a context containing cubes for each subband in that HTP. However you run `doGridding`, it will automatically take information about how the mapping observation was performed (beam size, number of readouts, maps size etc) from the HTP metadata and use that to create the cubes. You only need to tweak parameters in `doGridding` if your science goals require it, for example, if you require a specific beam or pixel size in order to compare with other observations.

To add `doGridding` to the Interactive Level 2.5 pipeline, you select *Interactive level 2.5 Pipeline* in the Interactive Pipeline panel in the `hiPipeline` task GUI, and then activate `doGridding`. See [Section 5.4.1](#) for more information. To run `doGridding` as a stand alone task you can either open the task in the *Tasks View* on an HTP - the GUI will autofill with information from the HTP metadata - or run in the command line with:

```
cubesContext = doGridding(htp=htp) [0]
```

You can learn in detail how to use the `doGridding` task in [Chapter 15](#). Here, we merely show how to create cubes for a selection of subbands, which saves memory, and with a specific pixel size:

```
# Create cubes for subbands 1 and 4, with a pixel size of 16.36" square
#
cubesContext = doGridding(htp=htp, subbands=Int1d([1, 4]), pixelSize=[16.36]) [0]
#
```

```
# Extract the cube for subband 1
cube1 = cubesContext.refs["cube_WBS_V_USB_1"].product
# Extract the cube for subband 4
cube4 = cubesContext.refs["cube_WBS_V_USB_4"].product
#
# Pixels can also be specified to be rectangular:
cubesContext = doGridding(ftp=ftp, pixelSize=[10.0, 20.0])
```

## 2.6.4.4. Spectral cube analysis

Cubes can be viewed in HIPE using the [Spectrum Explorer](#), and analysed using the [Spectrum and Cube Toolboxes](#). In addition, you can fit models to the spectra in the cube using HIPE's Spectrum Fitting Toolbox, the [Spectrum Fitter GUI](#).

The use of these Toolboxes is described in the links to the *Herschel Data Analysis Guide* given above but here we give some tips for using the Cube Toolbox with HIFI data:

- Crop cubes spectrally, and spatially if needed, to help prevent long computation times. In addition, tasks in the Cube Toolbox that involve line fitting, such as the `computeVelocity` task, can produce poor results if there is more than one line present in the data.
- The `computeVelocity` task in the Cube Toolbox provides a means to produce integrated maps using either a Gaussian fit to the line in the data, or a moments computation. Lines in HIFI spectra are rarely Gaussian in nature and this causes the Gaussian fit approach to produce poor results. The moments calculation works for simple lines shapes. However, the complex line profiles typically seen with HIFI's high spectral resolution are best treated by fitting multi-component lines in the Spectrum Fitter, and taking the integrated flux output from that.
- Always remove baselines prior to making calculations, particularly when creating integrated maps. It is recommended to do this prior to creating the cube.
- When using the `computePVMap` task, check the `correctAspectRatio` radio button to set this parameter to "True". HIFI data typically has a large spectral range compared to spatial range and this causes display problems with the PV map unless the aspect ratio is corrected.
- Sometimes, cube images produced by the Cube Toolbox do not display well in the *Standard Cube Viewer* and may even appear to be blank. To remedy this, right click on the image (or where it should be) and select the `Edit cutLevels` option, then set the cut levels to one of the choices available.

## 2.7. Spectral Scan Mode

Last updated: 27 July, 2015

### 2.7.1. Introduction

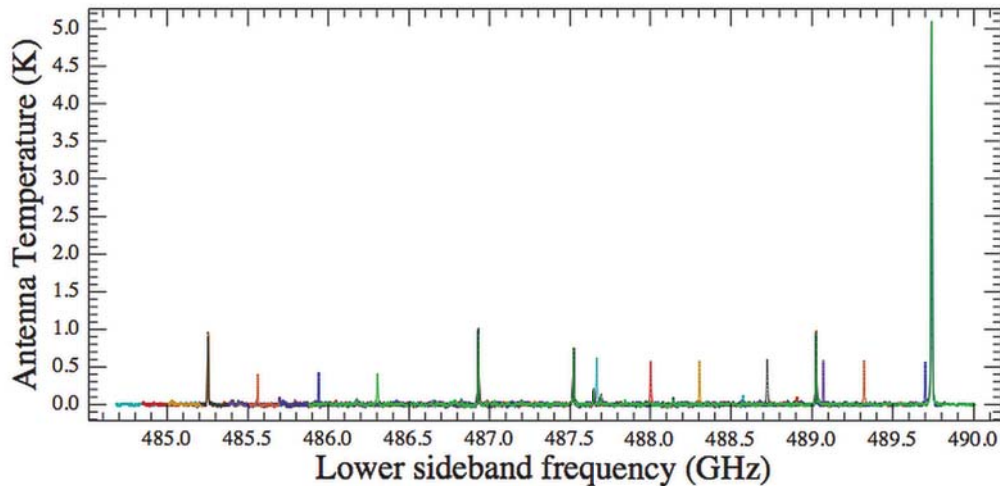
The Spectral Scan observing mode is one of the three main AOTs (Astronomical Observing Template) used with the HIFI instrument. It allows to build spectra over spectral ranges larger than the instantaneous bandwidth of the Wide-Band Spectrometer (4 GHz in bands 1 to 5, 2.4 GHz in bands 6 and 7) and cover large line surveys at very high spectral resolution. This cookbook describes the strategy used to collect the data and addresses the main issues to take into account when working with such data from the archive.

### 2.7.2. How Spectral Scan Mode observations are taken

#### 2.7.2.1. Observing Principles

The Spectral Scan observations consist in taking individual spectra at semi-regular frequency steps over the spectral range targeted by the observer. Because HIFI works with Double-Sideband (DSB)

receivers, there is an intrinsic degeneracy in the sky frequency to be assigned to a given channel of the spectrometers. Because lines in the respective Upper and Lower Sidebands (USB and LSB) will move in opposite directions in observations taken between two close frequencies, observing a given sky frequency several times with slightly different Local Oscillator (LO) tunings will allow to discriminate between the two sidebands. The reconstruction of the Single Sideband (SSB) spectrum from the DSB spectra is called *deconvolution*, and will be discussed in more details in [Section 2.7.4](#). [Figure 2.67](#) illustrates the individual Level 2 spectra collected over a short spectral scan.



**Figure 2.67.** Example of WBS spectra collected over a mini-scan in band 1a between LO=492.7 GHz and LO=493.9 GHz (8 settings, Obsid 1342191505). Each colour corresponds to a different LO tuning. The sky frequency scale used here is the LSB one. Those lines falling at the same sky frequency at each tuning belong to the LSB (e.g. at 489.75 GHz), while those falling at different frequency at various tuning belong to the USB.

The number of spectra covering the same sky frequency is driven by the so-called *Redundancy*, which is one of the input parameters of the Spectral Scan AOT and can vary between 2 and 12. Combined to the lower and upper ends of the spectral range to be covered, these parameters define the total number of spectra collected to build the spectral survey. They also define the exact LO tuning steps that will be used to build the redundancy. These tunings will use a slightly irregular stepping, necessary for the deconvolution algorithm to work optimally (strictly identical steps would create aliasing).

Spectral Scans can only be done per LO bands so that a survey over the whole HIFI tuning range involves at least 14 Obsids. Partial scans are also possible to target a particular spectra range – these mini-scans are particularly useful in sources with crowded spectra where redundancy is needed to disentangle the contribution from the respective sidebands. Like Point Mode observations, Spectral Scans observe along one single line-of-sight.

## Spectrometers in use

The default spectrometer in Spectral Scan observations is the WBS. However, the HRS may be used as well depending typically on the requested redundancy. This is not something chosen by the User but is automatically switched by the uplink engine. The HRS data do not have the necessary redundancy to achieve a decent deconvolved SSB spectrum, however they will provide complementary data at higher spectral resolution.

## Data noise in Spectral Scan

The sensitivity of the HIFI detector is usually not constant over the tuning range of a given band. If strictly the same time is spent at each LO tuning, some spectral ranges will suffer from degraded noise. In order to partially compensate for that, the observing time at LO frequencies known to be less



sensitive on average will be doubled. Those duplicated spectra will be kept separate at the Level 2, i.e. they will not be averaged.

### 2.7.2.2. Reference Schemes

There exist three flavours of the Spectral Scan mode, distinguished by the referencing scheme in use to collect the data, and usually optimised for the source geometry and/or nature (see also [Section 2.2](#)):

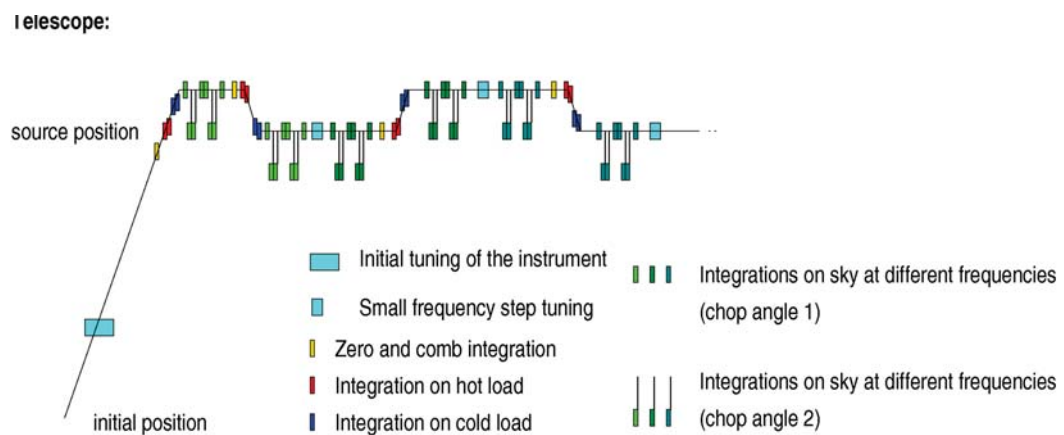
- Double Beam Switching (DBS), available both in Slow or Fast-Chop sub-modes
- Frequency Switching (FSW), with or without a Reference
- Load-Chop (LC), with or without a Reference

For each of the above scheme, we refer to the corresponding Single Point mode Cookbook for details about the observing mode characteristics: Dual Beam Switch ([Section 2.2](#)), Frequency Switch ([Section 2.4](#)), and Load Chop ([Section 2.5](#)).

Irrespective of the chosen scheme, the sequence of CAL-ON-OFF observations is repeated at each individual LO tuning, and put side-by-side in such a way that it optimises telescope slews as much as possible.

### 2.7.2.3. Observing Timeline

[Figure 2.68](#) illustrates the different observing blocks involved in a Spectral Scan observation for the particular case of the DBS referencing scheme. For each LO tuning (cyan box), a calibration block will be taken (yellow, red and blue boxes in [Figure 2.68](#)) in-between the two ON/OFF target blocks (green boxes). The difference shades of green correspond to different LO frequencies.



**Figure 2.68.** Sketch illustrating the observing sequence considered in Spectral Scans combined with DBS. Observing blocks are labeled as in the legend showed at the bottom right.

### 2.7.2.4. Frequency Grouping

Early in the Herschel mission, the idea of sharing calibration measurements over several consecutive LO tunings was considered as a way of optimising use of observing time. The observing time efficiency was then improved by acquiring several LO tuning points before visiting the calibration sources, or slewing to e.g. move to the second nod position in DBS observations. This is illustrated in [Figure 2.69](#), which follows the same colour code as in [Figure 2.68](#). This approach was tested during the Performance Validation (PV) phase but did not achieve the expected data quality when processed with the standard pipeline. As a consequence this version of the Spectral Scan was never rolled out to the Users. A certain number of Obsids taken in this fashion is however now publicly available in the Herschel Archive. The Users should be warned that part of those data is suffering from severely distorted baselines and very special care need to be taken when interpreting those data (see also the [disclaimer](#) note for public calibration data).

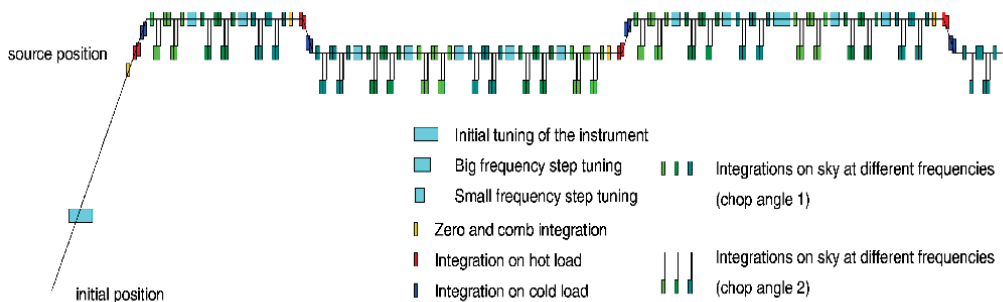


Figure 2.69. Same as in Figure 2.68 for Spectral Scans with a frequency grouping of 3 combined with DBS. The three shades of green are used here to represent the three different LO tunings combined within one single calibration block.

## 2.7.3. Inspecting Spectral Scan Mode data

### 2.7.3.1. Data Structure

The Spectral Scan data structure is similar to that of the equivalent Single Point mode data, but contains more data due to the typically large number of LO tunings in an Obsid. Figure 2.70 illustrates the summary tables of a Spectral Scan at the Level 1, where calibration and OFF phase observations are still present. The measured LO frequency for each spectrum is listed in this table.

In this particular case, one can see that HRS data were acquired during the observation. Note also that the very first dataset (“tune”) corresponds to an adjustment of the WBS attenuator levels, and should not be confused with the LO tuning blocks, which are not listed in this summary table as they don’t contain any spectra.

Apart from the first frequency, which is bracketed by a calibration block, one can recognise the sequence illustrated in Figure 2.68: the calibration block, composed of a “comb” (WBS frequency calibration) and an “hc” (code for “Hot-Cold”, for the bandpass and flux calibration), is followed by the ON- and OFF-target blocks (here corresponding to the two DBS nodding phases, see Section 2.2). The LO tuning takes place right before the calibration block and has a fixed duration that depends on the frequency. Tuning indeed implies some thermal stabilisation of the LO chain so that a dead time allowing for this is allocated to the tuning block. As we will see in 3.4, this dead time may be too short in some circumstances and lead to data imperfection.

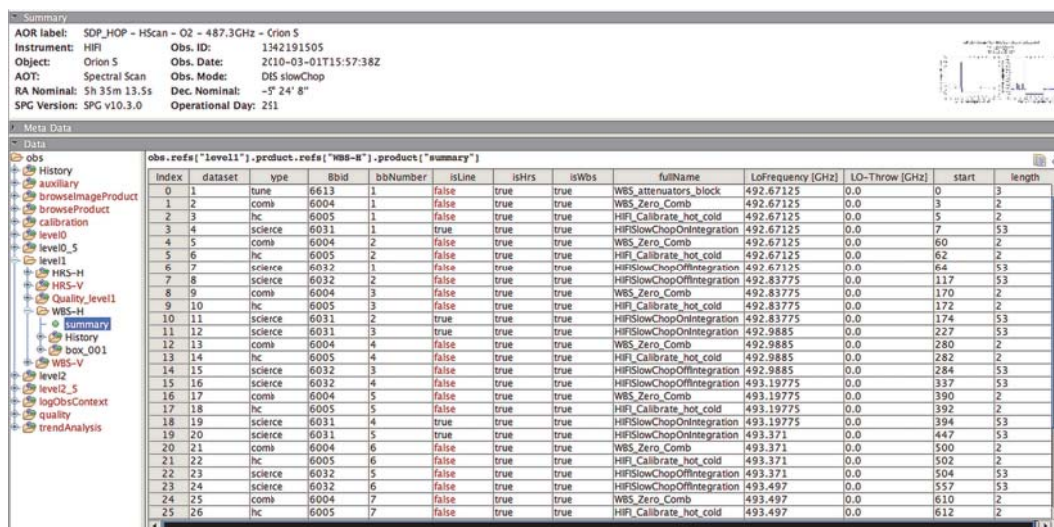


Figure 2.70. Observation context of a Spectral Scan, and summary table at Level 1 (here only partial)

At Level 2, only ON-target spectra are maintained and there will be one spectrum per individual LO tuning. Note that in case a given LO frequency is observed twice, the corresponding data will not be averaged but rather kept independent. It is possible to visualise all individual WBS subbands of a given polarisation in the same plot, as was illustrated already in [Figure 2.67](#). For this, you just have to open the Hifi Timeline Product (HTP), i.e. one of e.g. WBS-H-USB, WBS-H-LSB, etc, with the Spectrum Explorer (hyperlink to SE doc), and select all spectra contained in it.

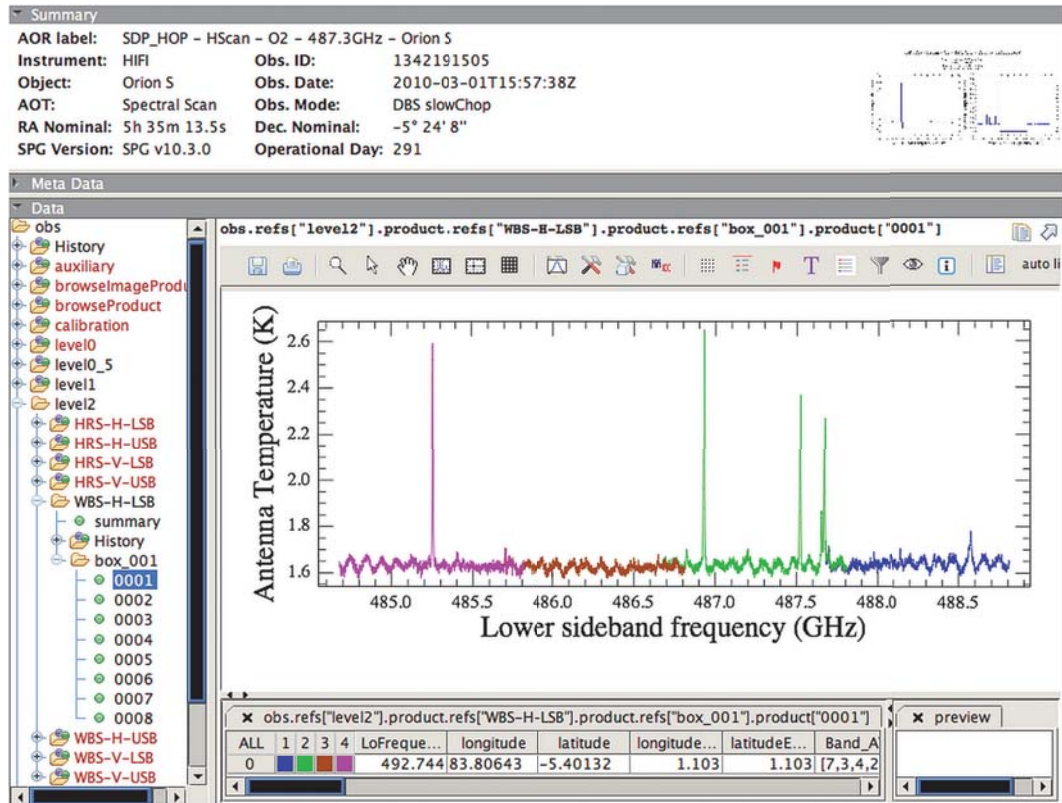
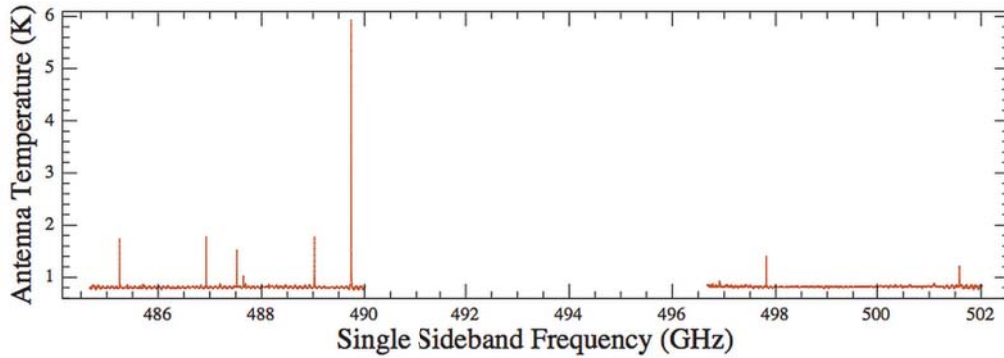


Figure 2.71. Observation context of a Spectral Data, and example spectra from the Level 2 products. Note the residual standing wave in those data.

### 2.7.3.2. Level 2.5: Deconvolved Spectra

In comparison to Single Point mode data, the Spectral Scan observation context contains an additional Level 2.5 product hosting the result of the deconvolution task run on the Level 2 data for each spectrometer. These data are also reflected in the browse product featured at the upper right corner of the observation context visualiser, as well as quick-look figure in the HSA. It is important to note that the data at Level 2.5 are the straight result of the Level 2 data deconvolution and that no particular data cleaning is performed prior to that. This means that any artefact still present at Level 2 will be propagated. In that sense we warn that Level 2.5 products may not be fit for science usage and we recommend to carefully cleaning the Level 2 data before re-running the deconvolution task. Some recipes are given in the following section.



**Figure 2.72.** Level 2.5 deconvolved spectra for the WBS-H data in Obsid 1342191505. Note the residual standing wave resulting from imperfect data quality at the Level 2. Note also the gap between the lower and upper side band sky frequency ranges due to the limited LO tuning coverage.

Because of its Double Sideband nature, HIFI detects the continuum in both the Lower and Upper Sidebands simultaneously and they add up in the DSB data at Level 2. Typically, for a Sideband Ratio-balanced receiver, the DSB continuum will be about twice its SSB value. The deconvolution algorithm is capable of recovering the SSB information both from the lines and from the continuum, so that the continuum at Level 2.5 has a single sideband scale too. Whether this continuum is accurate or not on an absolute scale is independent of the deconvolution process and very much depends on the quality of calibration in the data fed into the deconvolution. This means that any artefact affecting the continuum in the Level 2 products will propagate as a continuum error in the deconvolved spectrum.

## 2.7.4. Spectral Scan Mode Data Reduction

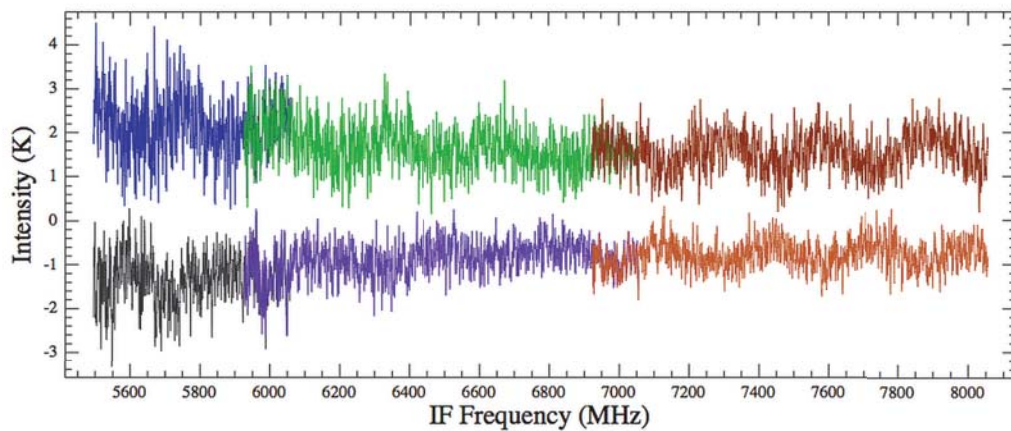
### 2.7.4.1. Data Artefacts and Cleaning

The typical data artefacts associated with Spectral Scan data do not differ too much from those usually encountered with their associated Single Point mode observations – one exception may be the baseline distortion involved by potential LO tuning settling time issue, which is discussed in more details in the following sub-section.

The main data problems one can find in Spectral Scan can be summarised as follows:

- Standing Waves:
  - Optical standing waves can be present in the data, with typical periods depending on the band used (see this [report](#) for more details). Those baseline modulations are usually enhanced in the presence of non-negligible continuum (see [Figure 2.71](#)), and can become really severe on planet observations. For these standing waves, the `fitHifiFringe` task (see [Chapter 12](#)) usually does a pretty good job – it is recommended to not use more than 3 components.
  - Additionally, bands 6 and 7 are affected by a peculiar Electrical Standing Wave (ESW) that forms behind the mixer in the IF (Intermediate Frequency) amplification chain (see [Figure 2.73](#) for an example). This modulation is not sinusoidal in nature and can only be dealt with by `fitHifiFringe` in case of very simple isolated lines (i.e. narrow, without wings). For more complex cases, a dedicated algorithm has been developed (so-called *matching technique*) and is offered in the `hebCorrection` task (see [hebCorrection](#) and [Section 12.4](#)). From HIPE 13.0 onwards, the ESW are automatically corrected in the pipeline by the task `doHebCorrection`, which applies prepared solutions stored in the HIFI calibration product. Not every observation is perfectly corrected, unfortunately, though we are pursuing those judged in need of improvement. Having said that, it has been observed that Spectral Scan are usually less sensitive to this kind of modulation than Single Point or Mapping modes – there can however be exceptions.
- Spurious spectral features:

- There are two main sorts of spurious features in the HIFI data: narrow spur lines and saturation (probably a very broad form or the narrow spur). While the HIFI pipeline does its best to automatically detect and flag those, it is possible to fine-tune this data flagging with the `flagTool` task (see [Section 11.5](#)).
- Residual baseline slopes/structure:
  - In addition to the spurious features, it is still possible to have residual baseline structure, usually in the form of a slope or a more complex shape. These residual can be corrected using polynomial fits to the baseline with the `fitBaseline` task (see [Chapter 13](#)). It should be noted that this kind of problems is particularly strong in Spectral Scans taken in FSW or LC mode **without** a reference, as well as Spectral Scans using FSW in bands 6 and 7. Although all these options were strongly discouraged, it is possible to find such data in the archive (part of them belonging to the Performance Verification Phase) so very special care needs to be taken if you want to exploit such data-sets.



**Figure 2.73. Illustration of an Electrical Standing Wave in a Spectral Scan (Obsid 1342244537) at two different LO tunings. The data are here shown at Level 1 on an IF scale. As can be seen a continuum offset is usually also associated to the data distortion.**

One of the fundamental points in Spectral Scan data processing is that any of the artefacts still present in the Level 2 data fed into the deconvolution algorithm will be treated as valuable information to build the SSB solution and can therefore severely distort the output result. [Figure 2.74](#) illustrates the progressive improvement (and residual artefacts) in the deconvolved data when taking or not into account some of the above-mentioned effects. In the final panel (c), one can see that the flagging of some of the spurious areas can result in small portions of the deconvolved data being empty if too many channels need to be thrown away.

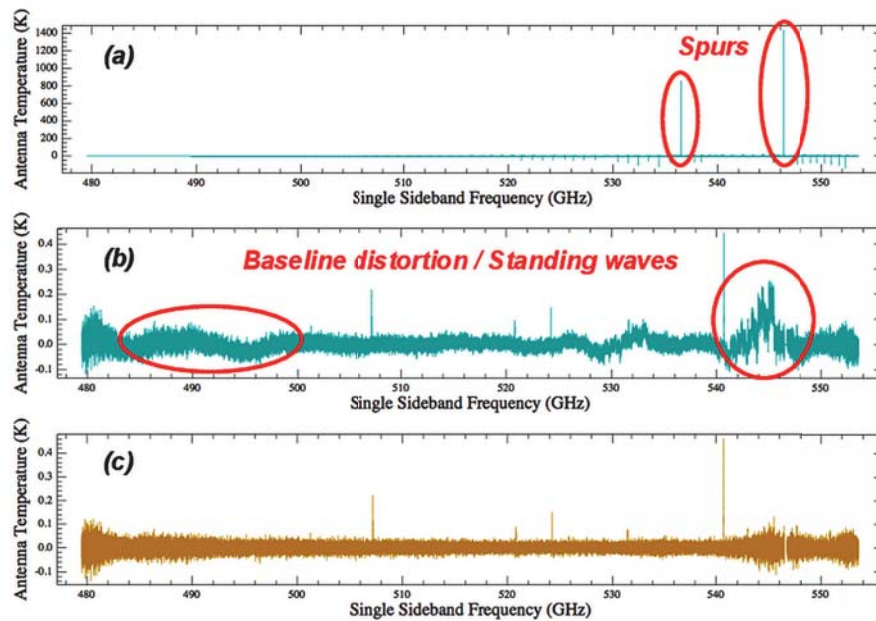


Figure 2.74. Output of the deconvolution at three levels of data cleaning in Obsid 1342190099. The top panel uses un-corrected data at Level 2, where spurs were still present and un-flagged (note the presence of negative ghosts as well). The middle panel uses data with spurs flagged but residual baseline structure still present. The lower panel uses data with the fringes and baselines corrected.

We indicate in the following two tables the flags that are relevant to the data cleaning of Spectral Scan data, in particular those recognised by the `deconvolution` task. We recall that *rowflags* refer to the whole spectrum, while *channel flags* apply to channel ranges – see the HIFI Data Reduction Guide, [Chapter 10](#) for more details.

Channel Flag Name	Description	Comment
SPUR_CANDIDATE	Potential spur at this channel	Set by pipeline
BRIGHT_LINE	Flag to denote a bright line	Set by User
IGNORE_DATA	Flag to denote channels to be ignored	Set by User

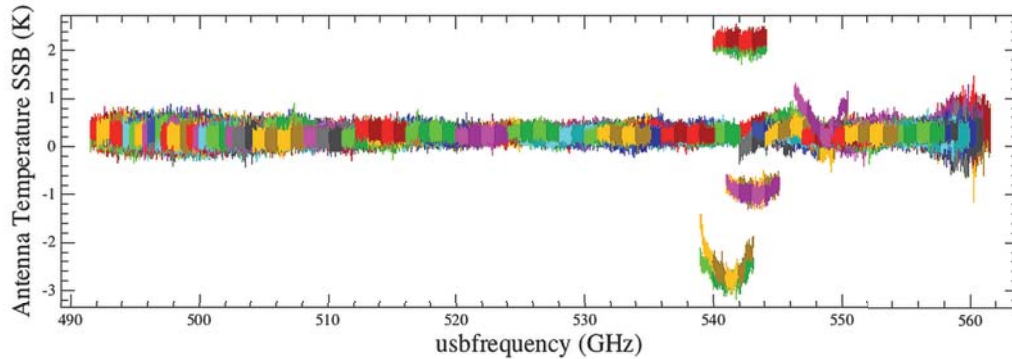
Rowflag Flag Name	Description	Comment
SUSPECT_LO	Potentially bad spectrum	Set by pipeline
IGNORE_DATA	Flag to denote spectrum to be ignored	Set by User

Note that the `SUSPECT_LO` *rowflag* is based on an a priori table of well known unruly LO frequencies, but does not necessarily mean that the data are corrupted. Any of the above flags can be added/undone by the `flagTool` task. We describe in more details in [Section 2.7.4.3](#) how some of the `deconvolution` task options can be adjusted in order to honour, or not, some of the above flags.

## 2.7.4.2. LO Tuning Settling Issues

There is an intrinsic thermal settling time involved in the HIFI electronics every time the instrument has to be configured to observe at a new frequency. In Spectral Scans, these re-tuning occur at a relatively quick pace since only limited time can be afforded at each frequency point (typically up to

10 seconds ON-target per LO tuning). Depending on the frequency, it may happen that the settling time exceeds the dead time allocated to the tuning block before starting the observation. In this case, part of the spectra may be acquired under sub-optimal stability conditions and result in imperfect baseline calibration at the Level 2. This problem usually occurs at isolated ranges of a given band, as illustrated in [Figure 2.75](#).



**Figure 2.75.** Example of LO settling time issues in part of a Spectral Scan in band 1a (Obsid 1342232978). This plot shows the collection of all WBS-H subband data collected at Level 2 on an USB scale. The settling time issues occur here at USB frequency around 542 GHz.

We currently have no particular solution for this instrumental effect in the pipeline. Obviously these baseline outliers can have very severe impact on the quality of the deconvolved data. There are different alternatives to circumvent this problem:

- The brut force approach would consist in flagging those spectra as `BAD_DATA` using the `flagTool` so they get totally ignore in further processing.
- If possible, it is however preferable to correct the baseline distortion using tasks such as `fitBaseline` although this would probably not allow to recover any continuum information (if applies).
- In case the Level 2 spectrum at a given affected frequency consists of the average of several individual Level 1 spectra (i.e. the total integration time was large enough that it implied more than one chopping cycle), a more sophisticated way can be envisaged at Level 1 whereby typically the first of those Level 1 chopping cycle spectra is flagged so it is ignored in the average used to form the Level 2 data. This usually leads to much cleaner baseline quality at the expense of increased noise.

### 2.7.4.3. Deconvolution Tricks

#### How to deconvolve more than one observation at a time

By default the Level 2.5 products only consider the Level 2 data from its own Obsid. This means that data at the lower and upper end of the scan will sample sky frequencies only in one sideband. When spectral surveys have been taken over contiguous frequency ranges, it is therefore advantageous to perform the deconvolution over a combination of several Obsids. Indeed, in this case, band edges will benefit from data available in both sidebands, allowing to better constraint the SSB solution, as well as improving the noise in those data. This is achieved with the following command lines (here for example with 2 Obsids):

```
obs1 = getObservation(obsid1)
obs2 = getObservation(obsid2)
obs_array = [obs2]
decon_result = doDeconvolution(obs = obs, obs2_array = obs_array)
```

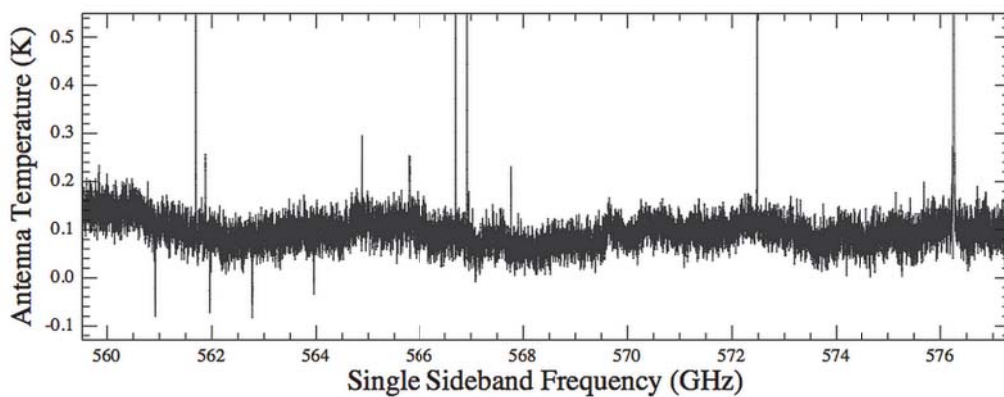
Obviously this will only make sense if the combined Obsids share the same line of sight. If not, the task will complain.

## Deconvolution and flagged spurs

As explained in [Section 2.7.4.1](#), there are several ways of masking parts of the spectrum containing spurious information. In terms of deconvolution, both row and channel flags `IGNORE_DATA` will always be discarded, regardless of any option. However, flags concerning spurs will or will not be taken into account depending on the `spur_rejection` option. It is sometimes the case that flags automatically set, like `SPUR_CANDIDATE` or `SUSPECT_LO` are too conservative and mask useful data. It is recommended that you review those with the `flagTool` task and remove them if applies. When this is carefully done, the best option for `spur_rejection` should be `REJECT_SCAN_WITH_SPURS`.

## Deconvolution of strong lines

The deconvolution algorithm can create spectral ghost features (usually showing up as negative features) in the neighborhood of relatively strong lines (typically several tens of Kelvins). This is illustrated in [Figure 2.76](#). In order to recover both weak and strong spectral line emission in the SSB spectrum (level2\_5 -> myDecon -> myDecon\_WBS-H -> dataset), it is necessary to mask those lines in the Level 2 (using the `BRIGHT_LINE` mask) and then run the deconvolution in two steps, with and without the `ignore_bright_line` option selected. This is explained below.



**Figure 2.76. Spectral Ghost artefacts from the deconvolution of a strong line (standard Level 2.5 products from WBS-H in Obsid 1342215923). The strong 12CO line lies at 576.5 GHz (70 K) and injects negative signal in the deconvolution in the range 561-564 GHz.**

By default deconvolution runs ignoring any flag on the bright lines, i.e. it corresponds to:

```
decon_bright_notflagged = doDeconvolution(obs = obs, ignore_bright_line = 0)
```

Once this is done, an inspection of the output (level2\_5 -> myDecon -> myDecon\_WBS-H -> dataset) should indicate to the user where bright lines might have resulted in negative ghost, like in [Figure 2.76](#). The User then needs to go back to the Level 2 data and use the `flagTool` task to flag all channels of the corresponding bright lines with the flag `BRIGHT_LINE`. Note that this has to be done for all tunings where the bright line is present in a spectrum (usually appearing both in the USB and in the LSB). The deconvolution can then be run again:

```
decon_bright_flagged = doDeconvolution(obs = obs, ignore_bright_line = 1)
```

In the above result, the bright line will be totally absent from the SSB spectrum, but so will be the negative ghosts. It is then a matter of merging the two above deconvolution outputs in order to basically inject the isolated signal of the bright line recovered from `decon_bright_notflagged` into the spectrum from `decon_bright_flagged`. As an example, for the 12CO line at 576.5 GHz:

```
# We now need to concatenate those two spectra only at the location of the bright lines
```



```
# Extract SSB spectra from decon output
decBright = decon_bright_notflagged["dataset"]
decNoBright = decon_bright_flagged["dataset"]

# Extract flux from decBright in channels where decNoBright is NaN
freqInd = decNoBright["flux"].data.where((IS_NAN(decNoBright["flux"].data)))
decNoBright["flux"].data[freqInd] = decBright["flux"].data[freqInd]
```

## Deconvolution of Spectral Scan in FSW mode

Taking Spectral Scan in combination with the FSW referencing scheme was essentially limited to the Performance Verification phase, and in effect almost no science program did finally use it over the mission. In total there are 19 Spectral Scans taken in this fashion in the Herschel Science Archive (only four are not from the Calibration programme). In general the data quality of these data meets the general level achievable with other referencing schemes (with the exception of bands 6 and 7 where it does not do a good job at all), however there can be issues when it comes to deconvolving those data. [Figure 2.77](#) and [Figure 2.78](#) illustrate the presence of ghosts resulting from imperfect treatment of the negative phases of the FSW sequence. We recommend using these data and the outcome of their deconvolution with very special care, and favour any equivalent dataset taken with another referencing scheme such as DBS or Load Chop.

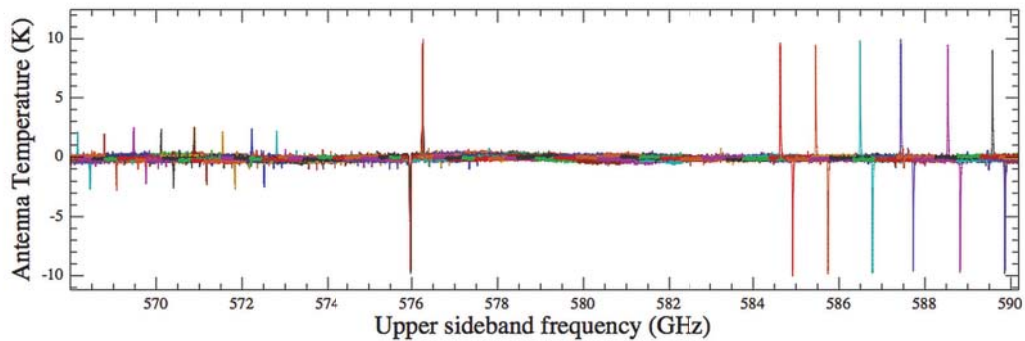


Figure 2.77. See [Figure 2.78](#) caption

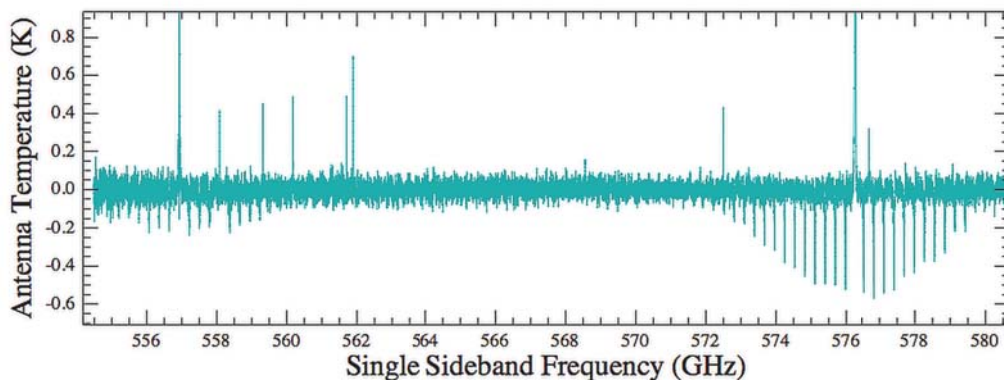


Figure 2.78. Spectral Scan FSW data in Obsid 1342190186. The upper panel ([Figure 2.77](#)) shows part of the Level 2 spectra from the WBS-H in the USB scale. The water line at 557 GHz (from the LSB) can be seen on the lower end of the spectrum, together with the 12CO (5-4) seen both in the USB and LSB (upper end of the spectrum). The lower panel ([Figure 2.78](#)) shows the outcome of the deconvolution algorithm (on baseline-corrected Level 2 data) around the water and 12CO lines. Note the ghost features associated with the negative phase of the FSW and separated by the frequency throw.

### 2.7.4.4. Assessing emission in OFF data

It is possible to check whether the OFF position data contain potential line contamination from a non-blank sky position. As of HIPE 13.0, those data are readily available for Spectral Scan taken in any

mode, and can be found in the calibration product (*calibration* > *pipeline-out* > *ReferenceSpectra*) – see [Figure 2.79](#).

The OFF data are now processed up to an equivalent Level 2, and have USB and LSB products. For Spectral Scan observations, on fixed and moving target, there is one OFF per LO Tuning, per backend/sideband. OFF positions spectra can be run through the standard tasks (*fitHifiFringe*, *fitBaseline*, *flagTool*). Additionally for Spectral Scan, the OFF positions can be run through the deconvolution to be directly compared to the ON-target Level 2.5 data (use option *use\_reference* from *doDeconvolution*). Note that OFF positions will also be corrected from Electrical Standing Waves by default.

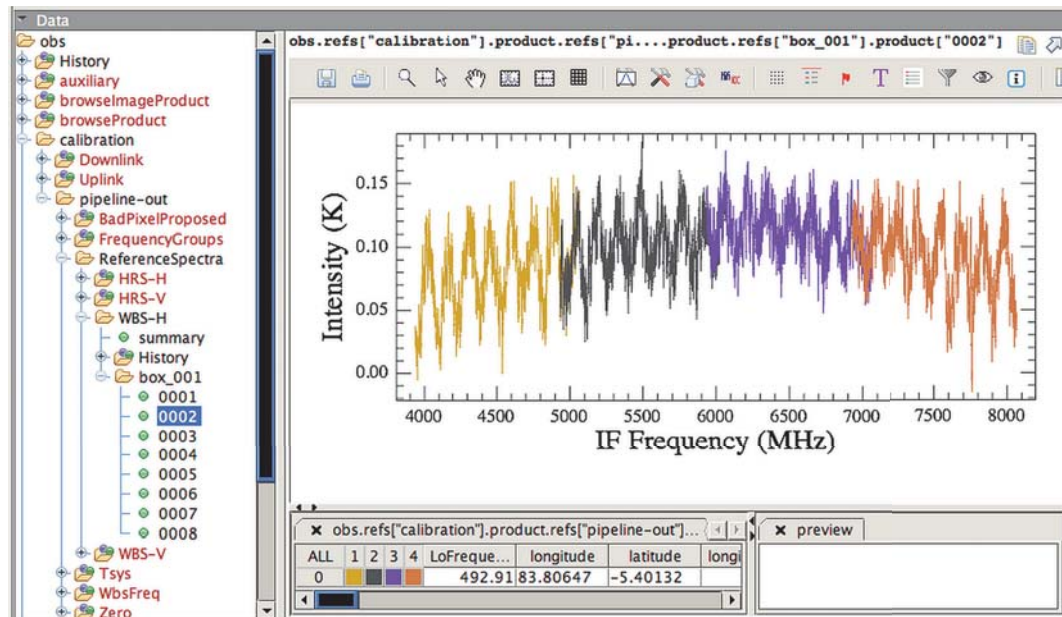


Figure 2.79. Example of Reference Spectrum at a given LO tuning for Obsid 1342191505. Since the spectrum is made of a single difference of two OFF spectra taken at different chopper positions, the optical standing waves are not as optimally corrected as in a double-difference calibration.

### 2.7.4.5. Combining data from the two polarisations

Although the H and V mixers do not strictly point at the same position in the sky (the offset between the two is small compared to the beam at the applicable frequencies), it is often useful to combine the signal from the two to improve the signal-to-noise. This combination can be done after the deconvolution using the *accumulate* task.

```
deconTotal = accumulate(ds = [deconH,deconV], pointingTolerance = Double.NaN)
```

It is also useful to compare the respective H and V deconvolution outputs to check for consistency in the detection of weak lines.

### 2.7.4.6. Exporting deconvolved data

The deconvolution output can be exported to a FITS file:

```
simpleFitsWriter(deconTotal, "/Your/Path/deconTotal.fits")
```

It can also be exported to a Class-compatible FITS file:

```
hifiDeconToClass(deconTotal, filename = "/Your/Path/deconTotal.fits")
```

# Chapter 3. Tour of a HIFI ObservationContext

Last updated: 4 February, 2016

## 3.1. Data Primer

All Herschel data comes from the HSA in the form of an *ObservationContext*, which is comprised of various layers of science data, auxiliary data, and other information about your observation in an onion skin type of structure. In this chapter we describe what you will find in an HIFI ObservationContext. First we give a short introduction to the structure of Herschel HIFI data storage, starting with the smallest data structure, the *dataframe*, and work our way up to the outer skin of the ObservationContext.

### 3.1.1. Data frames

The Herschel spacecraft stored data onboard (up to two days' worth) until it was transmitted to Earth. Science data, such as a WBS spectrometer readout, came naturally in sets, or Frames. Data frames were packetised for transmission from the Herschel Space Observatory to Earth. Along with House Keeping (HK) data, they were downlinked to the [tracking station](#) and thence to the Mission Operation Centre (MOC) at ESOC in Darmstadt, or to the latter directly. The data packets then flow from the MOC to the Herschel Science Centre (HSC) at ESA's European Space Astronomy Centre (ESAC) in Madrid. The HIFI ICC copied the data from HSC, as well.

At ESAC, the data packets were 'ingested' into a database and the science data frames were reconstituted.

The combination of HK and science data created a *SpectrumDataset*, or spectrum in more common language.

### 3.1.2. Data Products

A Herschel Data Product consists of metadata keywords, tables of *SpectrumDatasets*, and the history of the processing that generated the product. There are various product types (Observation, Calibration, Auxiliary, Quality Control, User Generated).

The types of Observation Data Product for HIFI is called a *HifiTimelineProduct*, or HTP. This is a time-ordered series of *SpectrumDatasets* comprising all the integrations in the observation.

For more information about Herschel products, see the [Product Definitions Document](#).

### 3.1.3. Contexts

A Context is a subclass of Product, a structure containing references to Products and necessary meta-data. A Context can contain Contexts, giving rise to Context 'trees.' Types:

1. ListContexts (for grouping products into sequences or lists, hardly used)
2. MapContexts (for grouping products into key value dictionaries)

#### 3.1.3.1. Herschel Observation Context

A MapContext instance serves as the organisational product unit for the Herschel Data Processing system. It contains the following contexts:

1. Level-0, Level-0.5, Level-1, Level-2, Level-2.5, & Level-3 (optional) Contexts

2. Calibration Context
3. Auxiliary Context
4. Quality Context
5. Browse product
6. Trend Analysis Context
7. optional Telemetry Context: not by default, only when the HSC deems it necessary because of a serious problem in the processing to Level-0 data.

The uses of these Contexts will be described in [Chapter 5](#).

Note that the descriptive modifiers "Product" and "Context" are often dropped conversationally.

## 3.2. HIFI Science Data

HIFI Science data is found in the Level 0, 1, 2, and 2.5 Contexts and is the result of each stage of the pipeline (data at Level 0.5 is removed in order to conserve memory as it is less useful). The availability of data from each level of the pipeline allows you to easily check for problems in the data that are better seen at earlier stages of processing, such as individual integrations showing signs of poor thermal stability ("bad scans") or erroneously repeated calibration observation datasets that cause the pipeline problems (because the pipeline expects data in a set pattern of science and calibration integrations for each observing mode). Additionally, the presence data from each stage of processing means that you do not need to re-pipeline the entire observation if you wish to re-run or modify only, say, the Level 2 pipeline.

The following is found in the Level 0 to 2.5 Contexts:

- **Level 0**

The Level 0 Context contains a *HifiTimelineProduct* (HTP) for each spectrometer used in the observation, which contains a dataset for each integration in the observation, and as a consequence contains more datasets than higher levels of processing. The Level 0 Context also contains a *Quality* Product based on checks on the Level 0 data for telemetry issues common to both the WBS and HRS (in the *CommonTm* product), and for checks on the data frame count and quality (in the *Quality* product for each spectrometer).

Level 0 is the most raw that you will see your data. It has been minimally manipulated into a HTP, had pointing information from the satellite associated with it, and undergone several "sanity checks" to flag any incidences of housekeeping parameters, such as the mixer currents, being out of allowed limits. Additionally, information from the *Uplink* product, which contains information calculated by HSpot concerning how the observation should be carried out, e.g., the spacing between scan legs in a map, is copied to the HTP and metadata. Level 0 data has units of channel number and counts.

- **Level 0.5**

Level 0.5 science data is removed upon the successful creation of a Level 1 product but you can recreate it by re-running the pipeline up to Level 0.5, or to higher levels, and using the `removeLevel0_5=False` option in the command line pipeline, see [Section 5.2](#).

Quality information is retained at Level 0.5, and it contains the WBS-H and WBS-V comb, and zero quality checks.

- **Level 1**

The Level 1 context contains HTPs for each spectrometer used in the observation and a *Quality* product.

Level 1 data is frequency and intensity calibrated, and also corrected for the velocity of the spacecraft. Level 1 denotes the stage of processing that the ICC believes can safely be done automatically. Data processed to Level 1 is in the IF scale (in MHz) and on the  $T_A^*$  temperature scale (in K).

The quality product at Level 1 contains, for each spectrometer used in the observation, the results of the phase checks done by the Level 1 pipeline (in *PhaseChecks* and, in the *FlagsSummary* product. The *FlagsSummary* contains a table summarising all the row flags that have been assigned to the individual Level 1 spectra of a given spectrometer. The table is complementary to the list of spurs detected by the pipeline, and compiled in the *TrendAnalysis* context (see [Section 3.7](#)). Row flags can also be graphically visualised on individual data-frames in the Spectrum Explorer (see [Section 6.2](#)). The list of HIFI row flags can be found in section [Section 10.3](#).

- **Level 2**

The Level 2 Context contains an HTP for each spectrometer used in the observation. Level 2 data is converted to antenna temperature scale ( $T_A^*$  in K) and to sky frequency (GHz). Spectra are averaged together, per each spectrometer, for each LO setting, and each spatial position in the observation. This results in a single spectrum (for each spectrometer) for point observation mode, averages per LO setting for spectral scans, and spectra averaged per position and LO setting for maps (OTF maps are made by scanning continuously and are not averaged). These data products may be at a publishable quality level, although corrections for baseline issues are likely required, and should be suitable for Virtual Observatory access.

- **Level 2.5**

The contents of the Level 2.5 Context depend on the observing mode. The data is derived from the Level 2 data and is combined into cubes in the case of mapping data, deconvolved to produce a single sideband product in the case of Spectral Scans, or the HTP are stitched, folded (if Frequency Switch) and converted to *simpleSpectrum* format in the case of point mode observations.

Note that the Quality Products provided with each level of data processing are the results of checks done and are always populated with information even if nothing is found to be wrong. In addition, some of the quality checks are done for trending purposes. This is in contrast to the *quality* Context (below the *logObsContext* in the *ObservationContext* tree) that is only filled with reports of potential problems with the quality of your data (see [Section 3.6](#)).

You can find more information about the pipeline processing steps in [Chapter 4](#).

## 3.3. HIFI Calibration Data

The *CalibrationContext* contains all of the data passed to the pipeline for calibration (*Downlink*), the calibration files created by the pipeline (*Pipeline-out*), and information about how the observation was carried out (*Uplink*). The contents of these are described below. Products tables can be plotted using right-click on "Open With / TablePlotter".

The *Downlink* calibration node contains the following:

Node	Contents	Description
Eng	DiplexerCoefficients	Table containing the diplexer coefficients for the diplexer bands (3, 4, 6 and 7).
Generic	BeamProfiles	Tables containing the HIFI beam profiles. There is a node containing the tables for the 2D beam model and a node containing the tables for the azimuthally-averaged (1D) beam models. Beam models are provide for two spot frequencies, per mixer band for each polarisation H and V. Spot frequencies are chosen to

Node	Contents	Description
		be close to the central frequency of the LO subbands a and b, respectively (see <a href="#">Chapter 8</a> ).
	Spurs	Obsolete. Historically, tables listing known LO impurities. Now replaced by spurFlags.
	spurFlags	List of spurs to be applied as channel and rowflags to a given Obsid. For point and map observations, the vast majority is based on a spur predictor model and labelled as "SPUR_WARNING".
	apertureEfficiency-H, apertureEfficiency-V	Tables listing the aperture efficiencies for each band for the H and V spectrometers, respectively. For each spectrometer there is one node containing the efficiencies for a Gaussian beam approximation (provided for backwards compatibility), e.g. apertureEfficiency-H, and a second node containing the efficiencies for the 2D beam model, e.g., apertureEfficiency2dBeams-H (see <a href="#">Chapter 8</a> ).
	beamEfficiency-H, beamEfficiency-V	Tables listing beam efficiencies for each band for the H and V spectrometers, respectively. For each spectrometer there is one node containing the efficiencies for a Gaussian beam approximation (provided for backwards compatibility), e.g. beamEfficiency-H, and a second node containing the efficiencies for the 2D beam model, e.g., beamEfficiency2dBeams-H (see <a href="#">Chapter 8</a> ).
	beamWidth-H, beamWidth-V	Tables listing beam widths for each band for the H and V spectrometers, respectively. For each spectrometer there is one node containing the beam widths for a Gaussian beam approximation (provided for backwards compatibility), e.g. beamWidth-H, and a second node containing the beam widths for the 2D beam model, e.g., beamWidth2dBeams-H (see <a href="#">Chapter 8</a> ).
	chopperPositions	Table listing the voltages required to point the chopper at each load. There is one table for the prime (not used after OD 81) and redundant sides.
	chopperThrows	Table listing the chopper throws (in degrees) for each band.
	couplingEfficiency-H, couplingEfficiency-V	Tables listing the efficiency in coupling to the hot and cold loads for each band for the H and V spectrometers, respectively.

Node	Contents	Description
	forwardEfficiency-H, forwardEfficiency-V	Tables listing the forward efficiency for each band for the H and V spectrometers, respectively.
	hebCorrection	Spline models and fit results of standing waves found in the HEB mixers (bands 6 and 7). These are also known as ESW.
	mixerCurrentTolerances	Table listing the mixer current tolerances for each band.
	sideBandGainIF-H, sideBandGainIF-V	Tables listing the sideband gain coefficients in the IF for each band for the H and V spectrometers, respectively.
	sideBandGainLO-H, sideBandGainLO-V	Tables listing the sideband gain coefficients in the LO for each band for the H and V spectrometers, respectively.
	smoothOffWidth-H, smoothOffWidth-V	Tables listing the smoothing widths in MHz, per band, to be used in the pipeline for Load Chop and Frequency Switch observations for the H and V spectrometers, respectively.
	uncertaintyModel-H, uncertaintyModel-V	Tables containing the uncertainties (percentages) related to the various error budget components of the HIFI flux calibration, for the H and V spectrometers, respectively (see <a href="#">Chapter 9</a> ).
HRS-H/HRS-V	CalHrsPowCorr	Gain non-linearity correction table used in HRS pipeline.
	CalHrsQDCFast	Table needed to perform an approximate correction of the quantisation distortion of the correlation functions of HRS. This table is not used as a standard part of the HRS pipeline.
	CalHrsQDCFull	Table needed to correct the quantisation distortion of the correlation functions of HRS. This calibration is used in standard processing of HRS data.
Level 0	APE	Measured absolute pointing error for different time periods of Herschel operations.
	BBids	Table listing BBids used in the pipeline to identify observing mode and phases of the observation.
	CleanDF	Tables of observations (obsids) which required slightly corrupted by the on-board software.
	HK	Table listing house keeping parameters, their units, and a description.
	UpConvertLo	Contains the upconversion factors (in the metadata).
	hkThreshold-H, hkThreshold-V	Contains tables listing the threshold parameters for the H and V spectrometers,

Node	Contents	Description
		respectively, for: the mixer currents of the SIS bands (1-5), the magnetic resistance for all bands; the diplexer current for the diplexer bands (3, 4, 6 and 7); and the Local Oscillator Unit (LOU) currents for bands 1, 3, and 7 (to avoid impurities).
WBS-H/WBS-V	badPixels	Table listing the pixels known to be bad for the WBS-H and WBS-V spectrometers, respectively.
	combFitParameters	The parameters used for the fitting of COMB spectra.
	linearityCoefficient	Table listing the WBS linearity coefficients for each band for the WBS-H and WBS-V spectrometers, respectively.
	zeroThresholds	A table of minimum and maximum acceptable Zero values.

The Uplink calibration node contains the following:

Node	Contents	Description
HifiAORData	Values of HSPOT parameters	If present, indicate the values and parameters used in HSPOT to plan the observation.
HifiUplinkData	UplinkModes	The list of all possible HIFI observing modes and the expected uplink parameters.
	UplinkParameters	A description of each uplink parameter.

The pipeline-out calibration node contains the following:

Node	Contents	Description
BadPixelProposed	WBS-H, WBS-V	Table listing pixels identified by the WBS pipeline to be saturated ('bad') for the WBS-H and WBS-V spectrometers, respectively.
Baseline	HRS-H, HRS-V, WBS-H, WBS-V	Only for observations containing a sky reference (or 'OFF') position. Spectra of the smoothed and averaged OFF positions for each of the spectrometers.
ESWCorrection	all spectrometers	The list of fit parameters per spectrometer of the found ESW for each spectrometer.
ReferenceSpectra	all spectrometers	Calibrated off spectra. For DBS observations, these are difference spectra of the averaged CHOP positions for each spectrometer.
FrequencyGroups	HRS-V, HRS-V, WBS-H, WBS-V	Tables listing the dataset index mapping for the frequency



Node	Contents	Description
		groups identified by the pipeline for each of the spectrometers.
Tsys	HRS-V, HRS-V, WBS-H, WBS-V	Spectra of the Tsys and Band-Pass calculated by the pipeline for each of the spectrometers.
Uncertainty	WBS-H-USB, WBS-H-LSB, WBS-V-USB, WBS-V-LSB, HRS-H-USB, HRS-H-LSB, HRS-V-USB, HRS-V-LSB	Table providing, for each Level 2 spectrum, the uncertainty table for the flux calibration error budget (percentages), based on the error propagation of the uncertaintyModel entries from the Downlink node (see <a href="#">Chapter 9</a> ).
WbsFreq	WBS-H, WBS-V	Frequency coefficients calculated by the MkWbsFreq step of the WBS pipeline for the WBS-H and WBS-V spectrometers, respectively.
Zero	WBS-H, WBS-V	Spectra of the zero level observations for the WBS-H and WBS-V spectrometers, respectively.

## 3.4. HIFI Browse Products

A browse product is an automated extraction from the results of the standard pipeline. It allows you to have a quick look at your data and assess what you have observed, as well as any problems you may have to deal with. The browse product is **not** intended to be used for science work.

The browse product is displayed in the Observation Context summary as a thumbnail image and in the Observation Context tree in the *browseImageProduct*. Clicking once on the thumbnail image and double clicking on the browse Image Product will open a large version in the *Browse Image Viewer*, you can zoom in and out on the image by scrolling with the mouse wheel (or using the equivalent trackpad motion). Click on the image again to exit the Browse Image Viewer. The spectra used to create the browse product are stored in the *browseProduct* Context in the Observation Context tree.

The HIFI browse products are as follows:

- *Point Mode*: this shows two plots of unstitched Level 2 WBS spectra with the H-polarisation to the left and the V-polarisation to the right. The upper and lower axes of the plots show the LSB and USB frequency scale, respectively. The AOR label and observation number are used to title the plot. The observing mode, source name, requested RA and dec are below the plot title.

The spectra stored in the browseProduct Context are all the Level 2 spectra, including the HRS which are not used for the browse product otherwise.

- *Mapping*: this shows a sets of map-averaged Level 2 spectra for each subband with the integrated map, also for that subband, to the right. The sets of images are arranged in order of increasing frequency, i.e., subband 1-4 from top left to bottom right for bands 1-5, and subbands 4 to 2 from top left to bottom left for bands 6 and 7. The AOR label and observation number are used to title the plot. The observing mode, source name, requested RA and dec are below the plot title.

For each spectrum, the upper and lower axes of the plots show the LSB and USB frequency scale, respectively. The y-axis is scaled to a factor 1.2 times the peak value in the spectrum excluding data flagged as a spur, this allows you to see that the data is impacted by a spur but still see other features in the spectrum.

To the right of each spectrum are integrated maps that are created with no correction done for any baseline issues. In the cases that the baseline suffers from drifts and/or standing waves the continuum will dominate the map. The right and bottom x- and y- axes show the RA and declination, respectively, while pixel coordinates are shown on the auxiliary axes. The colour scale used for the image is 'heat' and the intensity scale used is 'ramp' so the strongest emission in the map should appear white.

The spectra stored in the browseProduct context are the stitched and averaged WBS-H and WBS-V Level 2 spectra.

- *Spectral Scan*: this shows the single sideband solution after deconvolution of the Level 2 WBS spectra. No baseline correction has been done prior to deconvolution. The H-polarisation is shown to the left and the V-polarisation to the right. The gap between the sidebands is shown as a line at 0 K. The AOR label and observation number are used to title the plot. The observing mode, source name, requested RA and dec are below the plot title.

The data stored in the browseProduct Context are the output from `doDeconvolution`, as performed on the Level 2 HTP, these are identical to the Level 2.5 products.

## 3.5. HIFI Auxiliary Data

The information tables in the Auxiliary Product are passed to the pipeline from sources which have stored how the observation was planned and carried out, including data from related sub-systems of the telescope such as the pointing system (ACMS). They are required in order to assign pointing information to the data, and provide some constraints in certain steps of the pipeline, as well as for technical diagnostics. For most data reduction, you are unlikely to need to work directly with the Auxiliary Product, but it is helpful to know what and where the “usable” information is located in this product in case of tailored reductions needing the same information as in the pipeline. This will pertain mainly to the `HifiUplinkProduct`.

### Attitude Control and Measurement System (ACMS) Telemetry Product

The spacecraft Attitude Control and Measurement System (ACMS) consist of several components. Telemetry are provided from the main constituents of the ACMS which consist of the attitude control computer (ACC), gyroscopes (GYR), star trackers (STR), reaction control system (RCS), reaction wheel assembly (RWA), Sun acquisition sensors (SAS), coarse rate sensors (CRS), and attitude anomaly detectors (AAS).

### Events Log Product

The events log product is intended to provide a uniform product containing event reports from either the instruments, or the spacecraft.

### HifiUplinkProduct

This product was introduced into the Auxiliary Product tree to capture the essential parameters related to the planning, execution, and predicted performance of the observation. Some of these parameters are used in subsequent pipeline steps, described below.

The `HifiUplinkProduct` solves two main issues of connecting the observation planning and execution with the data processing by means of key observing configuration parameters:

1. In HSpot the parameters which completely describe the planning, execution, and expected performance of observations were not always stored in the associated AORs, thus they were not available in the data products.

This is because of a limitation in the HIFI design of the observation planning tool that treated such parameters as informational for the User during time estimation. The HIFI planning software in HSpot was modified to solve this roughly half-way into the mission, and this message content

(which has seen several format changes over the mission) is henceforth stored with the AORs for further processing to merge parsed parameters with the data products in the pipeline. The merging is done at Level 0, starting in OD 835 (the *odNumber* of an observation is in the obsContext header). For earlier observations, the HIFI ICC has regenerated and stored in the HIFI calibration tree the parameters for each obsid using the same version of AOT software and calibration files, the co-called Mission Configuration (MC) for configuring and operating the instrument, that was operational at the time the observation was scheduled regardless of when it was initially delivered by the program PI to the HSC. This synchronisation is necessary to be consistent with the performance of the instrument during the observation, for instance system temperatures and thus sensitivities are dependent on how the Local Oscillators were operated at the time in different bands and specific frequencies (more information is contained in the [HIFI Observing Modes Release and Performance](#) notes). However, note that not all performance parameters provided in versions earlier than HSpot 5.0 are consistently produced, and this time dependency means that some parameters variably appear with UNKNOWN values in the HifiUplinkProduct.

To give an example, the baseline noise goals for an observation have been initially entered in HSpot by the User while setting up the AOR, in terms of either an observing time goal or a desired noise value. But the predicted single sideband noise performances (which are crucial when comparing noise measured in the data to that expected) are returned in the message output after time estimation. These appear, accompanied by units and descriptions, in the HifiUplinkParameters table of the

```
HifiUplinkProduct.obs.refs["auxiliary"].product.refs["HifiUplinkProduct"].product["HifiUplinkParameters"]
```

which can be navigated to through the Context Viewer in HIPE. If the values of *noiseMinUsb*, *noiseMaxUsb*, and so on are UNKNOWN, then they were not predicted (or else not in a retainable format) in HSpot under the version of the AOT software when the observation was carried out.

2. The AOR configuration and performance parameters with each observation must be unique to the HIFI observing mode which was used.

The Auxiliary Product tree also contains an UplinkProduct for all three Herschel instruments. For HIFI it contains tables of parameters describing the astronomical setup or configuration of the observation (such as the desired dimensions of a spectral map). However, this product has serious deficiencies, first that the parameters are restricted to the setups reflecting the User's goals for the observation, and excludes any of the actual configuration or optimised parameters following time estimation. Using spectral map dimensions again as an example, the User has entered desired X and Y dimensions in arcminutes in the AOR setup panel of HSpot, and these are stored as *flyX* and *flyY* in the UplinkProduct. However the actual dimensions are computed as part of time estimation, and depend on the input values of *flyX*, *flyY*, the *sampling* (Nyquist or other), and adopted *beam size*. There may be other factors which adjust the map dimensions (OTF maps typically have one extra map point along each scan line to mitigate a telescope scanning anomaly). These do not appear in the UplinkProduct, yet they record the actual mapping pattern that was commanded, and are needed to check the dimensions of the spectral cubes constructed in the pipeline. Furthermore, the UplinkProduct suffers a design deficiency that it contains the parameter sets for all HIFI observing modes, not just the one associated with the obsid. The extraneous parameters are left at default values, and this has led to confusion for both User and Expert as to which parameters are applicable and valid for the observation. Thus, the UplinkProduct has no value to the general User; the HifiUplinkProduct is considered valid and complete.

The properties related to the usage of the HifiUplinkProduct for data reduction may be summarised as follows:

- The full list of so-called uplink parameters associated with the employed observing mode and available for data processing, trending, and informational purposes is given in a table in the HIFI Calibration Tree (note the different location when navigating, it is not in the Auxiliary Tree):

```
obs.refs["calibration"].product.refs["Uplink"].product.refs["HifiUplinkData"].product["UplinkModes"]
```

- The units and descriptions of the combined lists of uplink parameters for all modes are tabulated also under the HIFI Calibration Tree:

```
obs.refs["calibration"].product.refs["Uplink"].product.refs["HifiUplinkData"].product["UplinkParameters"]
```

This table is informational here, its usage as a reference is in the Level 0 pipeline during generation of the HifiUplinkProduct, into which only the valid parameters relevant to the employed observing mode have been populated.

- The HifiUplinkParameters table in the HifiUplinkProduct consists of two sets of parameters of different origin from HSpot:
  - a. Configuration and setup parameters which have been entered by the User, or set by the AOT logic with no dependence on time estimation.
  - b. Configuration and observing performance parameters (time efficiency, noise estimation, mapping patterns) which are deduced through time estimation.

At the moment, there is no distinction in the product tables identifying the origin of the parameters.

- Most parameters in the HifiUplinkParameters table for each obsid are also copied into the HifiUplinkProduct header as metadata, to facilitate trending studies and database queries.
- The HifiUplinkProduct also contains a history of the AOT commanding during the observation, ObsBlockExecutionData, for expert diagnostics.

In the pipeline, the HifiUplinkProduct has several uses which can be turned into interactive tasks for tailored data reductions.

The [doUplink](#) task in the Level 0 pipeline copies several of these parameters into the HTP and HIFI Spectrum Dataset headers as MetaData. The copied parameters are often renamed (to be compatible with earlier versions of software). For example, *flyX* and *flyY* are present in the HifiUplinkProduct, while the equivalent *mapWidth* and *mapHeight* are present in the HTP and SDS headers. Other copied parameters and their equivalents can be found in the URM description for the doUplink pipeline task. This task operates on Level 0 HTPs and is not configurable, the User should normally not need to re-run this on data produced in HIPE or SPG versions 7 or later.

DoGridding (see [Chapter 15](#)) convolves the timeline of spectral map datasets into a spectral cube, using the attitude information which has been assigned to each dataset from the Pointing Product (in the Level 0 doPointing task) and counting the scan lines and readouts to set the map dimensions and pixel scales. The default dimensions should closely match those which can be reconstructed from HifiUplinkProduct parameters *mapLines* and *mapLineStep*, which specify the number of scan lines and their angular separation, and *mapReadouts* and *mapReadoutSep*, which specify the number of map points and their distance within each scan line. This is how the map pixel sizes are constrained, and it is important to construct the cube in this way in the default pipeline to be consistent with the requested pattern and for evaluating noise performances. The reconstructed attitude of the spectra at each map point (deviating from the ideal grid) is taken into account in the convolution. The User may adjust the pixel sizes or map dimensions and other task parameters in HIPE for over- or under-sampling, adjusting the WCS, etc, depending on scientific needs.

[MkRms](#) is a task which measures baseline rms noise in an observation, and compares the result to the expected noise as estimated in HSpot. The measurement depends somewhat differently on observing mode and requires several reference and noise performance parameters from the HifiUplinkProduct (spectral scan noise is predicted at a reference frequency selected by the AOT logic, for example). The key input parameters taken from the HifiUplinkProduct are *noiseMinUsb*, *noiseMaxUsb*, *noiseMinLsb*, *noiseMaxLsb*, *noiseMinWidth*, *noiseMaxWidth*, *tmbReference*, *noiseRefFrequency*, and *oneGHzReference*. The output of this task is populated in

---

tables and as metadata, and can be trended. Further information on these parameters is found in the description column of the HifiUplinkProduct, and in the URM mkRms entry on how they are used.

### **Horizons Product**

### **Housekeeping**

Data providing information about the state of the spacecraft during an observation. Further details are found in [doHkCheck](#).

### **MissingTM**

This product contains information of missing TM (Telemetry) packets after ingestion in the HSC. It has been designed to contain the minimum information required to unambiguously identify the missing TM packets. It is generated per Operational Day (OD).

### **MissionTimeLine**

This product packs the information provided within the EPOS summary file: pointing requests data, reaction wheel profile data, ground station coverage and DTCP data, and delta-V manoeuvre data. It is generated per OD. Not yet available.

### **Out-Of-Limit (OOL)**

The HPMCS SCOS-2000 BEHV performs behaviour checking for all parameters specified in the MIB OCF table. This information furnished to the HSC by means of DDS auxiliary TM data products. The Out-of-limits product shall pack all the information provided therein. It is generated per OD.

### **OrbitEphemeris**

The predicted and reconstructed products have identical format and contain time-dependent S/C state vector information as provided by FDS as Orbit Ephemeris Message (OEM) data. It is generated per OD.

### **OrbitEventsProduct**

These products have identical format and contain the predicted/reconstructed orbit events data furnished by Flight Dynamics (FDS) in the (short term) orbit events file. Events include Acquisition/loss of TM/TC signal at the ground station and eclipse events information. It is generated per OD.

### **Pointing**

The pointing product contains time-dependent spacecraft attitude information and will be built using information provided in the Attitude History File (AHF) furnished by the Flight Dynamics System (FDS). It is generated per OD.

### **Siam**

This product contains the Spacecraft/Instrument Alignment Matrices transforming vectors in the Herschel spacecraft reference frame to/from vectors in the different instruments' frames. The SIAM product is valid for a given period of time in the mission until a new measurement is done, and the product is updated.

### **SREMCALProduct**

The Standard Radiation Environment Monitor (SREM) detects and counts electrons, protons and cosmic rays with a coarse spectral resolution and some 20 degrees angular resolution. This product contains the calibrated accumulation and acquisition data, including the proton/electron count rates in the three detectors, fitted particle spectra and total dose in the internal RadFET. It is generated per OD.

### **SREMRawProduct**

Contains raw SREM accumulation and acquisition data, including readings from the different channels of detectors and internal RadFET, temperature and voltage data, etc. It is generated per OD.

#### TeleCommandHistory

This product contains information of telecommand history as furnished by the Herschel MCS by means of DDS service. It is generated per OD.

#### TimeCorr

The Time Correlator component within the HPMCS maintains the correlation between the spacecraft on-board time and ground time, providing interfaces to correlate OBT to UTC and vice versa. The Time Correlation product should contain all the relevant information produced by the Time Correlator component and stored in the SCOS-2000 Time Correlator Coefficient packets. It is generated per OD.

#### UpLinkProduct

The UplinkProduct is a default table produced in the SPG, and is obsolete for HIFI data reduction purposes. For all parameters relating to the planning and expected performances of HIFI observations on an obsid and observing mode basis, the User should refer to the HifiUplinkProduct.

## 3.6. HIFI Quality Context

This Context contains information about the Quality Flags of different severity, described in [Section 10.4](#). A viewer for QualityControl products can be obtained via the QualityContextExplorer, or printing the quality report found from the ObservationContext:

```
report1=obs.refs["quality"].product
print report1
```

or

```
report2= obs.getQuality()
print report2
```

The Quality Control Report also contains the state, if failed, the observation failed the quality assessment. An empty quality report indicates no problems in processing.

## 3.7. HIFI TrendAnalysis Context

This Context contains HifiCalibrationDatasets, i.e., products useful for tracking systematic changes in instrument response over time. These are:

### 1. CombTrend Context (only WBS)

The CombTrend Context represents the statistics of particular quantities (line, resol, zero, and position); for line, resol, and position, the statistics over 11 comb lines that are seen in every CCD for each backend (WBS-H and WBS-V), for the zeros, the statistics over all channels. The four entries (mean, stddev, min, max) denote average, standard deviation, minimum, and maximum of the particular quantity, respectively. Line is the peak of the Gaussian fit to the comb lines in counts, i.e., on a scale from 0 to 1013. Resol is the full-width-half-maximum of the Gaussian fit to the first comb line in a CCD. Zero is the zero level of the CCD, i.e., without any input in units of counts. Position is the channel number of the position of the Gaussian fit to the first comb line in a CCD zero.

### 2. FpuTrendProduct Context

The FpuTrendProduct Context contains a table with values of some HK parameters from the Focal Plane Unit (FPU) in function of time for a specific observation. The Columns of this table have

as name the HK mnemonics. If the input FpuTrendTable is missing a SEVERE message is raised: "Trend table is null. No check is possible".

### 3. LoTrendProduct Context

The LoTrendProduct Context contains a similar table as the FpuTrendProduct focussed on HK parameters from the Local Oscillator Unit (LOU). Both the FpuTrendProduct and the LoTrendProduct content are used in the Level 0 pipeline in order to flag possible out-of-limits on specific HK parameters. When this occurs, a dedicated quality flag will be raised and added to the quality product (see [Section 10.4](#)).

### 4. SpurTable Context (only WBS)

The SpurTable Context contains a table of the spurs detected in the calibration cold load spectra. This search is done at the end of the Level 0.5 pipeline in the WBS. It assumes a saturation level of "800 counts" in the WBS raw intensity scale at Level 0.5 (pre-bandpass calibration), and will return flagged areas as 1.5x broader than the detected faulty regions. This saturation level can be adjusted by the User. Next, the task looks for Gaussian-like features in the flagged region. Resulting spurs are categorised with the following names: "Positive Spur" (a spur that is brighter than the continuum in the cold load), "Negative Spur" (a spur that is fainter than the continuum in the cold load - note that they are very rare), "Saturated" (intensities are above the saturation threshold - see above), "Err: out of range". In this latter case, the flag indicates that the data is corrupted in a given range, but that the algorithm could not determine a good fit to a Gaussian. In any of the spur cases, an appropriate flag is set to the data-set (see [Chapter 10](#)). For more information about the algorithm, see the description of the [mkSpur](#) pipeline step.

### 5. Statistics Context

The Statistics Context contains a series of tables with computation of the first momentum of the observed spectra (mean and standard deviation) as well as the median. It is provided for each spectrometer, each subband of a given spectrometer, and for the datasets at Level 1 and Level 2. At the present time, only saturated pixels are excluded from the computation. Additional flags such as line masks or spur flags will be taken into account in the future.

### 6. TMpage Context

The TMpage Context contains diagnostic tables for every LO tuning that was performed during the observation. This information is mostly of interest for instrument scientists, and offers a dump of a selection of LOU HK parameters during pre-defined steps of the tuning process, at time stamps finer than the sampling rate offered for the periodic HK compiled in e.g. the FpuTrendProduct or the LoTrendProduct contexts.

### 7. Tsys Context

The Tsys Context contains the TsysTrendTable per backend and per subband, and provides the LO Frequency, the central IF, the observation time, and (nominal) resolution for the backend, and the Double-Side-Band System noise temperature and associated standard deviation computed from the calibration load datasets. The corresponding spectra can be visualised in the calibration product, under the pipeline-out > Tsys context (see section [Section 3.3](#)).

# Chapter 4. What was done to my data?

Last updated: 26 October, 2015

## 4.1. Introduction

Data retrieved from the HSA has been processed via the HIFI pipeline. Each step is described in detail in the [HIFI Pipeline Specification](#) document. In this chapter, we briefly describe what has been done to the data at each stage of the pipeline.

In the following sections, we show pipeline flow diagrams from the [HIFI Pipeline Specification](#) document. The data input and output from each pipeline step up to the end of Level 2 is always a HifiTimeline Product (HTP). The Level 2.5 pipeline expects input to be HTP but the output at the end of the pipeline can be a different type of product, for example cubes (SpectralSimpleCubes) are produced for mapping observations.



## 4.2. Level 0

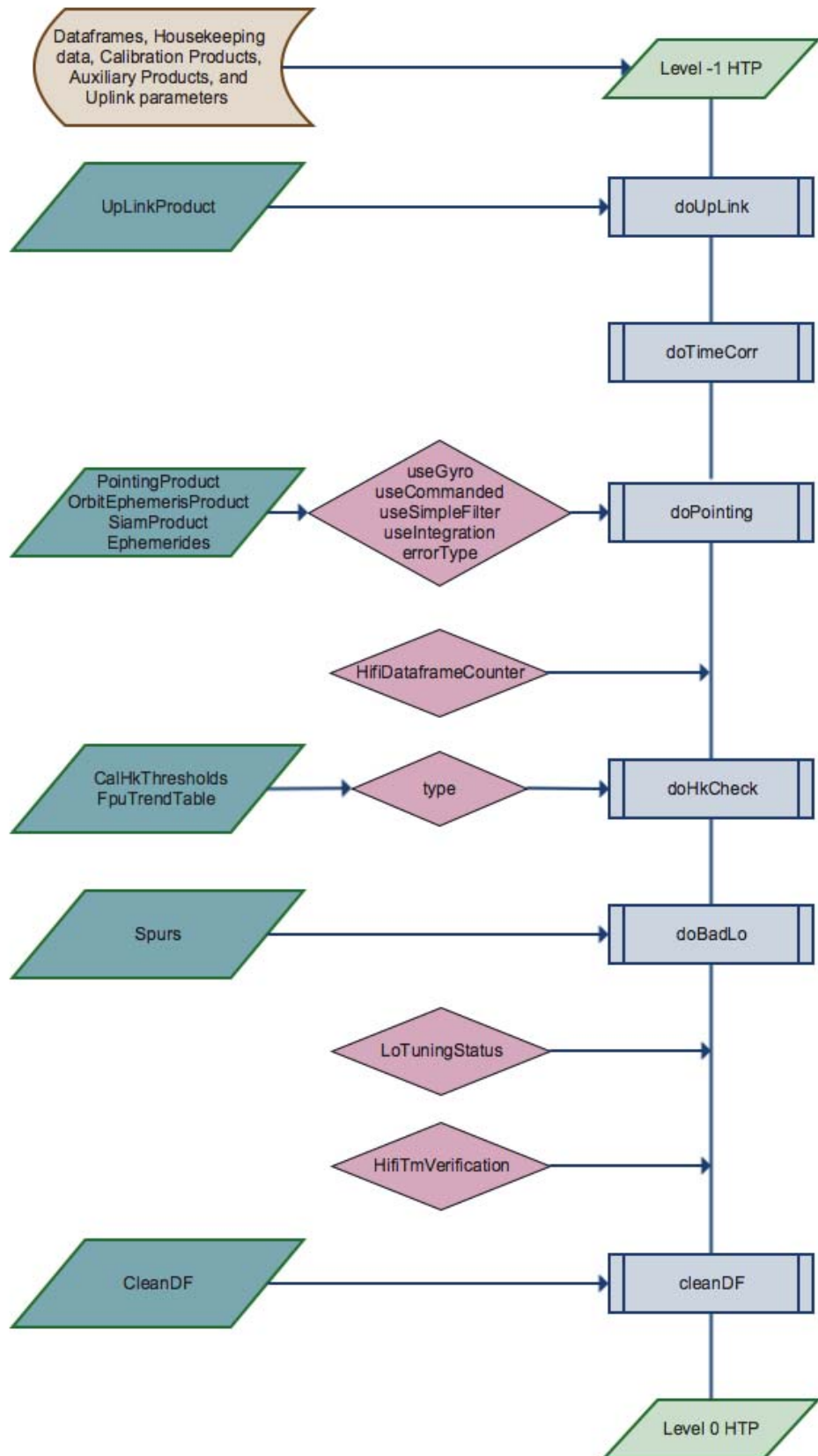


Figure 4.1. Level 0 pipeline flow diagram

---

Following the formation of an HTP, the Level 0 pipeline does the following to create a Level 0 HTP:

- **doBadLo**

Checks for LO settings that are known to be bad and flags that data.

- **doHkCheck**

Checks that the housekeeping parameters stored in the data are within the threshold limits determined by the ICC. If parameters are found to be out of limits then quality flags are raised, see [Section 10.4](#) for information about which quality flags can be raised, and the threshold conditions.

- **doPointing**

Associates the pointing information from the satellite with HIFI data. Note that HIFI is pointed such that the coordinates requested in HSpot fall between the H- and V- beams, this is known as the synthesised aperture. The pointing information for each polarisation can be found in the longitude and latitude columns of the datasets, while the pointing information for the synthesised aperture is found in the columns "longitude\_cmd" and "latitude\_cmd".

- **doTimeCorr**

Corrects the observation time of a scan to be consistent between the WBS and HRS. Scans at Level 0 from the WBS generally are formed from one integration, whereas scans from the HRS can be comprised of several integrations. If a correction was necessary then the description field of the obsTime is appended with "corrected".

- **doUplink**

Copies items from the Uplink Product that describe how the observation was carried out into the data and metadata of HTPs and `HifiSpectrumDatasets` in the HTP, these items are taken from the parameters calculated by HSpot when you set up your observation. This is particularly useful for mapping observations because the `doGridding` task uses the information copied into the metadata in order to correctly run the gridding routine.

- **cleanDF**

Task which sets the `IGNORE_DATA` rowflag when bad data are known to exist. The known bad data are in a calibration entry.

Level 0 data, which has units of channel numbers and counts, is the most raw that you will see your data. The Level 0 pipeline produces Level 0 data from the completely raw (or Level -1) data frames, housekeeping information, and telemetry information. This raw data is not available to you within the Observation Context so the Level 0 pipeline is not a part of the HIFI pipeline available to you in HIPE.

As you cannot re-run the entire Level 0 pipeline yourself, you are not able to take advantage of changes to the Level 0 pipeline until a bulk re-processing occurs at the HSA. However, it can be possible for you to avoid waiting for a new bulk re-processing by re-running some of the Level 0 pipeline tasks that occur after data has been combined into a `HifiTimelineProduct` (HTP). For example, a change in how the pointing information is applied to that data could require the `doPointing` step to be re-run. Note that you must always run `doPointing` with defaults (aside from the path and name of the new pointing product), and more specifically, you should never run `doPointing` with `useGyro = true` option.

The most common situation for re-running `doPointing` would be when new/improved pointing products are available for observations downloaded from the archive (HSA) which have not yet been bulk re-processed with the latest pointing information.

If it is required to re-run steps from the Level 0 pipeline, you can expect to either be informed in the appropriate place in this manual, by the Helpdesk, and/or on the [HIFI Instrument and Calibration page](#).

Re-running Level 0 pipeline steps requires you to work on the HTP rather than the Observation Context. Below we give two examples for re-running the `doPointing` step.

### Re-run the `doPointing` task for all spectrometers, and re-insert the result back into the Observation Context.

```
# Import doPointing task
from herschel.hifi.pipeline.level0 import DoPointingTask
#
# Initialise ChopperPosition
from herschel.hifi.pipeline.generic.utils import ChopperPosition
ChopperPosition.initialize( obs.calibration, obs.getStartDate() )
#
# Fetch the chopper throw calibration table
chopperThrow=obs.calibration.getCalNode( "Downlink" ).getCalNode( "Generic" ).\
getProduct( "chopperThrows" ).getByStartDate( obs.getStartDate() ),
"ChopperThrows" ).getProduct().getTable()
#
obsid="13421xxxxx"
#
# to get data from HSA use this
obs = getObservation(obsid, useHsa=True)
# to recover it from a local store use this
obs = getObservation(obsid, poolName="MyPoolName")
#
doPointing=DoPointingTask()
lev0=obs.getProduct("level0")
#
# This script fails if you do not have data from all spectrometers so, in the
# line below, list the spectrometers you DO have data from in your observation
for htpname in ["WBS-V", "WBS-H", "HRS-H", "HRS-V"]:
    htp=lev0.getProduct(htpname)
    htp=doPointing(htp=htp,useIntegration=1, aux=obs.auxiliary,
    chopperThrow=chopperThrow)
    lev0.setProduct(htpname,htp)
obs.setProduct("level0",lev0)
#
# The next command runs the pipeline, and overwrites your variable "obs"
obs=hifiPipeline(obs=obs, fromLevel=0, upToLevel=2.5, cal=True, save=False)
#
# Save observation, saveCalTree saves the calibration too, which is needed if you
# want to reprocess the data again
saveObservation(obs, poolName="NewPointing", saveCalTree=True)
```

### Reprocess an observation with a new pointing product.

This example assumes that the pointing product to be used is available as a FITS file on disk.

```
# Initialise ChopperPosition
from herschel.hifi.pipeline.generic.utils import ChopperPosition
ChopperPosition.initialize( obs.calibration, obs.getStartDate() )
#
# Fetch the chopper throw calibration table
chopperThrow=obs.calibration.getCalNode( "Downlink" ).getCalNode( "Generic" ).\
getProduct( "chopperThrows" ).getByStartDate( obs.getStartDate() ),
"ChopperThrows" ).getProduct().getTable()
#
# Fetch the new pointing product
mypointing = simpleFitsReader("pointing_od_XXXX.fits")
#
# Reprocess pointing at level0 and replace in obs context
# As, above, only list the spectrometers that are available in your observation
context
for htpname in ["WBS-H", "WBS-V", "HRS-H", "HRS-V"]:
    htp=lev0.getProduct(htpname)
    htp=doPointing(htp=htp, useIntegration=1, aux= obs.auxiliary,
    chopperThrow=chopperThrow, pointing=mypointing)
    lev0.setProduct(htpname,htp)
```

obs.setProduct("level0", lev0)

### 4.3. Level 0.5

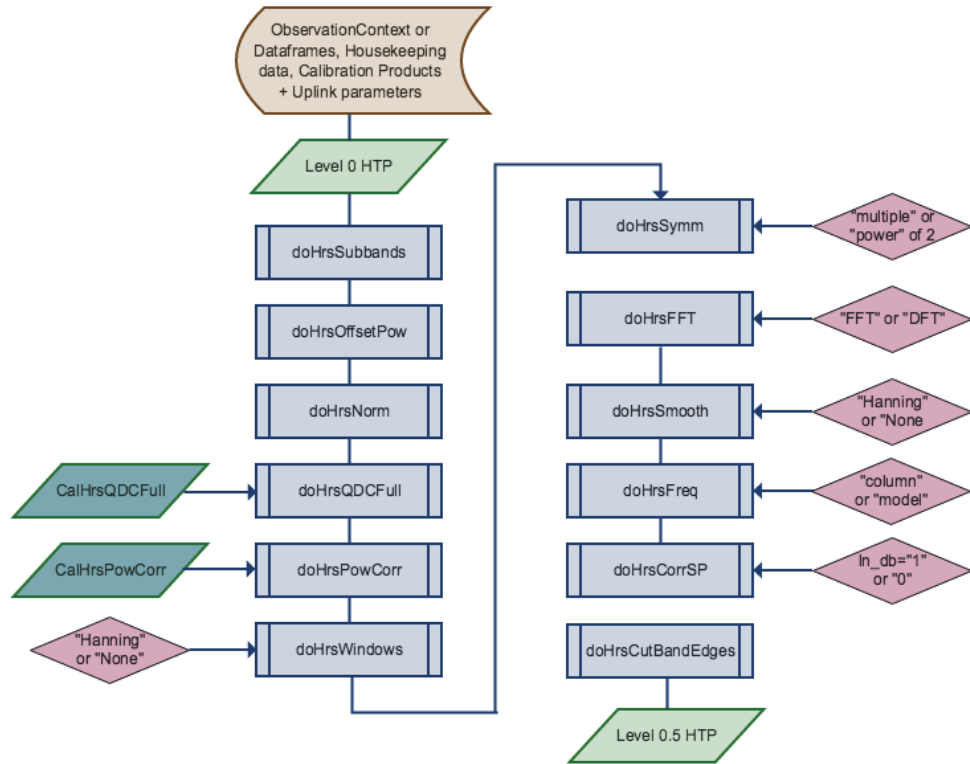


Figure 4.2. HRS pipeline flow diagram

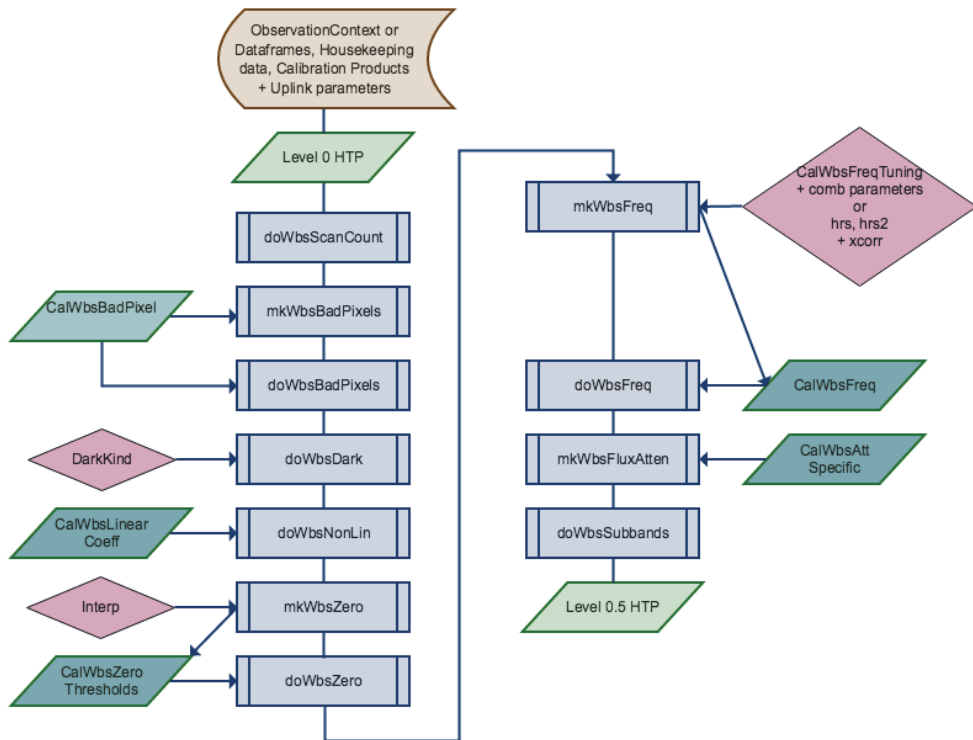


Figure 4.3. WBS pipeline flow diagram

Level 0.5 data is not found in the ObservationContext obtained from the HSA but you can recreate it yourself using the `hifiPipeline` task if something seems to have gone wrong with the frequency calibration of your data.

Level 0.5 data have been frequency calibrated and had instrumental effects, e.g., non-linearities in the WBS, removed. The spectra are in the IF scale, and split into subbands.

## 4.4. Level 1

Level 1 spectra are both frequency and intensity calibrated, although some effects of both the observatory and the observing mode are still to be removed. Level 1 is considered by the HIFI ICC to be as far as automatic processing of data is "safe", it is expected that human interaction above Level 1 will improve the results of the pipeline.

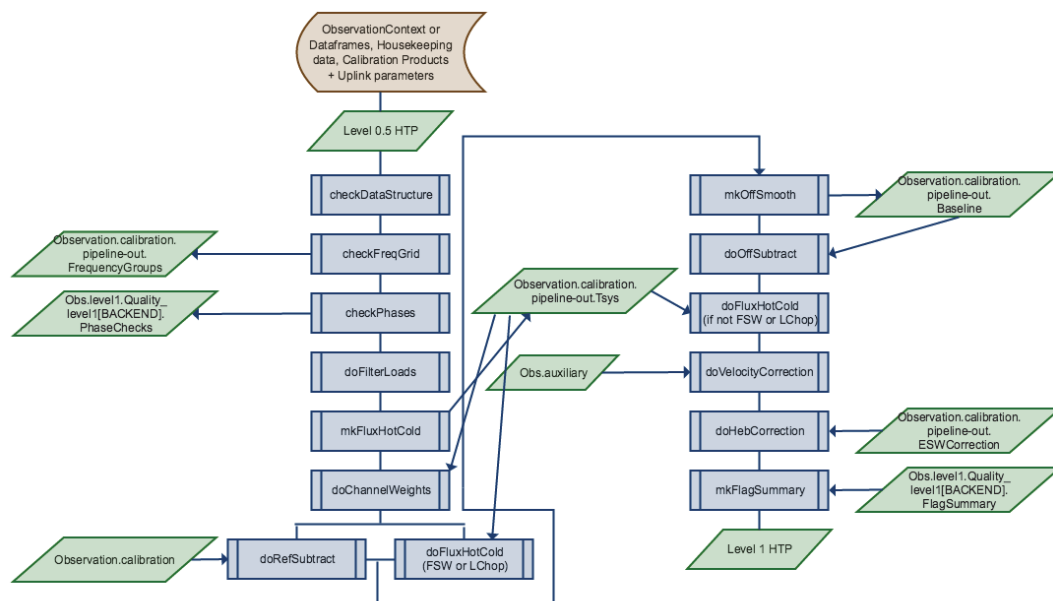


Figure 4.4. Level 1 pipeline flow diagram

### checkDataStructure, checkFreqGrid, checkPhases

The initial three steps of the Level 1 pipeline are checks that the data have the structure, content and form the patterns that are expected from the observing mode. The `checkFreqGrid` step also groups datasets according to LO (local oscillator) tuning, and checks for any possible frequency drifts in the data.

### doFilterLoads

An optional pipeline step that smooths the measurement of the loads to prevent the standing waves that are present from being propagated to the science data. For usage, see [Section 12.2](#).

### mkFluxHotCold

Hot/Cold load measurements are used to obtain both, the receiver (or system) temperature and the bandpass for each "hc" dataset in the original timeline product, and included in a calibration product (of type `CalFluxHotCold`). The bandpass is used for the intensity calibration, the system temperature and, possibly, for the determination of the channel-dependent weights to be included in the spectra. If channel weights are not to be determined by the system temperature, then this step of the pipeline can be moved to anywhere before the `doFluxHotCold` step.

### doChannelWeights

Fill values into the weights column, per subband. The default behaviour is to make use of the receiver temperature from `mkFluxHotCold`. Other possibilities include weighting by integration time, or by the variance of the spectra within a given (moving) window.

### **doRefSubtract**

Reference measurements taken from blank sky (in DBS modes), from an internal load (in Load Chop modes) or taken at a different LO frequency (in Frequency Switch modes) are subtracted from the source measurements (science datasets) in order to eliminate instrumental drifts from the source measurements.

This step constitutes one half of the double subtraction scheme typical for HIFI and by the end of it, the science datasets are replaced by the differenced spectra.

For the Frequency Switch modes, the shifted and the un-shifted spectra are overlaid (with opposite signs) by `doRefSubtract` and, reflecting the fact that each of these phases represent half of the observation, the integration time reported in the HTP is half of that reported in HSpot. The pair of shifted and unshifted spectra need to be 'folded' using the `doFold` task, the integration time will then be doubled as reported in Hspot.

### **mkOffSmooth**

Average and smooth (on the frequency scale) the flux data from the OFF measurements. The calculation is processed on a per dataset basis so that a baseline is constructed for each OFF dataset. These baselines will be subtracted in the `DoOffSubtract` step.

The default operation of this step is to first take the average over all the spectra included in the OFF dataset and then to smooth the data using a Gaussian filter (a box filter is an alternative). Other options available are to apply a polynomial fit after averaging the OFF datasets, or to only take the average of the OFF datasets. There are many options available when taking the average and these are described in the `mkOffSmooth` section in the HIFI Pipeline Specification document.

### **doOffSubtract**

The calibrated baseline(s) calculated in the `mkOffSmooth` step are subtracted from the ON measurements of load chop, frequency switch, and position switch modes (on a row-by-row basis for the position switch modes). In the case of DBS modes, the ON and OFF positions, which both contain science data, are averaged on a scan-by-scan basis. Note that if you want to reprocess your observation without `doOffSubtract`, you must run `doCleanUp` with the option `retain="science"` in order to only propagate ON and OFF spectra up to Level 2.

### **doFluxHotCold**

The calibrated intensity scale obtained in the `MkFluxHotCold` task - the bandpass - is applied to the flux data. This transforms the intensity scale to Kelvin units ( $T_A$ ). All science data (dataset with type 'science') found in the given product are adjusted in this way.

In the case of Frequency Switch and Load Chop observations, the reference subtracted data are intensity calibrated prior to OFF subtraction, which results in a better calibrated continuum, and so this step is performed between `doRefSubtract` and `mkOffSmooth`.

### **doVelocityCorrection**

Corrects the frequency scale for the velocity of the spacecraft and possibly of the source. A relativistic approach is adopted when correcting for the motion of the spacecraft relative to SSB or LSR or, in the case of SSOs, relative to the SSO. For non-SSOs, the motion of the sources relative to the LSR or SSB is treated classically.

Possible target rest frames to transform to are "HSO" (Herschel Space Observatory), "GEOCENTRIC", "SSB", or "BARYCENTRIC", "LSR" or "SOURCE". By default, the task transforms to the "LSR" frame for non SSO's and "SOURCE" for SSO's. A description of the definitions of these frames and their relation to each other is [Chapter 25](#).

**doHebCorrection**

Applies spline models and fit results of standing waves (stored in the HIFI calibration) that are found in the HEB mixers (bands 6 and 7). The task also creates a TableDataset which contains all the model parameters generated. The correction is performed on Level 1 spectra (type 'science': on-target spectra by default, and also off-target spectra if input parameter offsource=True).

**mkFlagSummary**

Creates a summary table with all the row and channel flags that have been set in the given timeline product. It loops through all SCIENCE ON data, collects this flag information, and puts this into a suitable summary table.

## 4.5. Level 2

The Level 2 pipeline attempts to remove the remaining effects due to the Observatory and the observing mode. It is expected that these steps will need to be repeated to improve upon the default settings and/or that undesirable features such as standing waves or spurs will need to be removed.

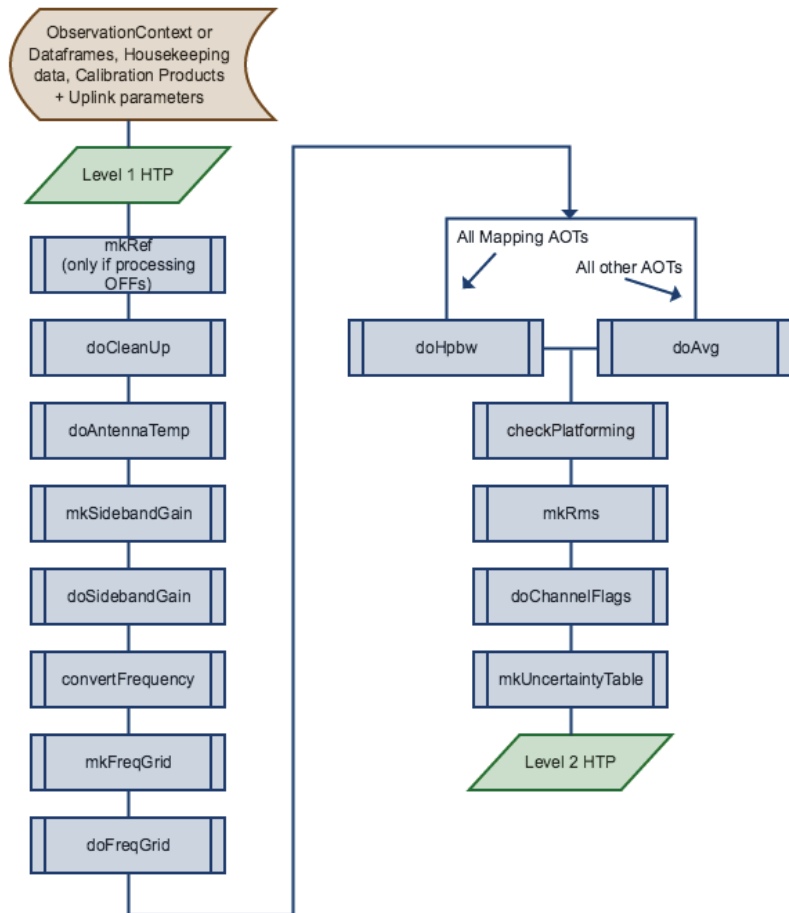


Figure 4.5. Level 2 pipeline flow diagram

**mkRef**

For DBS Modes this task calculates the difference in emission between the two chop positions. The resulting spectra give(s) a rough idea of whether or not the proposer chopped onto a region with emission.

### doCleanUp

This step removes all data from the timeline product that are not of type 'science' (`sds_type`) and that correspond to 'ON' measurements. Furthermore, science datasets belonging to the same LO tuning group, and/or the same raster point, and/or the same scan line number, are merged to form new datasets.

### doAntennaTemp

This step corrects for all telescope dependent parameters except the coupling of the antenna to the source brightness distribution, i.e. it translates to a  $T_A^*$  scale where  $T_A^* = T_A/\eta_1$  with forward efficiency  $\eta_1$ .

### doMainBeamTemp

This is an optional step that could be used in place of `doAntennaTemp` and converts to main beam temperature. It changes the label for the intensity axis to "Main Beam Temperature DBS", or to "Main Beam Temperature" if the sideband gains correction has been applied.

### mkSidebandGain

This step provides the sideband gains coefficients dependent on detector band, sideband, LOF-scale, and IF-scale. These coefficients will subsequently be applied in the `doSidebandGain` task. In you want to work with only the default coefficients 0.5, you can skip this task and call `doSidebandGain` without passing a coefficients via the `cal` task parameter.

### doSidebandGain

Divides the flux (at this stage typically an intensity) by the sideband-specific, detector-band specific, LOF- and IF-dependent gain coefficients. The gains applied to the data are written to the metadata of the Level 2 spectra and are denoted by *usbGain* and *lsbGain* for the upper and lower sidebands, respectively.



#### Intensity scales

HIFI is a double sideband (DSB) instrument. In consequence, application of sideband gains mean that the detected lines are calibrated to single sideband (SSB) scale, while the continuum contains contribution from both sidebands, and remains at the DSB scale. If the gains are unity, then the continuum is at twice the SSB scale. Beware, however, that the introduction of sideband gain coefficients at the low frequency end of band 2a, and in band 5 mean that a different treatment is required in these cases, see [Chapter 16](#).

### convertFrequency

In this step of the pipeline, the spectra are transformed from the IF frequency scale to sideband frequencies. For detector bands 1-5 this is defined by:

$$f_{\text{usb}} = f_{\text{LO}} + f_{\text{IF}} \text{ and } f_{\text{lsb}} = f_{\text{LO}} - f_{\text{IF}}$$

For bands 6 and 7, it is:

$$f_{\text{usb}} = f_{\text{LO}} + \text{CF} - f_{\text{IF}} \text{ and } f_{\text{lsb}} = f_{\text{LO}} - \text{CF} + f_{\text{IF}}$$

where the conversion factor, CF, is given by 10.4047 GHz for horizontal and 10.4032 GHz for vertical polarisation.

At the same time, the units are changed from MHz to GHz.

This step is not a dedicated pipeline step and is also provided for use in interactive analysis so that data can be transformed to different scales, including the velocity scale.

### mkFreqGrid



This step creates a linear frequency grid that can be used by the `doFreqGrid` task to resample the spectra. By default, the width between successive gridpoints is set to 0.5 MHz for WBS data, while for HRS data, it is determined by inspecting the input spectra.

### **doFreqGrid**

Resamples all the HIFI spectra to the frequency grid specified as input task parameter. By default, the grid determined in the `mkFreqGrid` step is used. The resampling scheme is set as a trapezoidal integration scheme in combination with a linear interpolation scheme. By default, the flux values in the output grid are resampled using an Euler scheme.

### **doHpbw**

Task to add the assumed half-power beam width (`hpbwAssumed`) parameter in the metadata of the HTP and the `HifiSpectrumDatset`. This task is applied to all mapping AOTs.

### **doAvg**

Computes the average over different scans that belong to the same LO tuning group (frequency surveys), the same raster column and row (in raster maps), or the same line number in OTF maps. Furthermore, science data from ON or OFF are not mixed. Various different options for how to do the average, and for pre-selecting the scans to be averaged are available but the default in the standard pipeline is to return a single dataset for each of the conditions described above.

### **checkPlatforming**

Platforming is seen as offsets between neighbouring WBS subbands and/or curvatures within subbands. Platforming can be addressed using the standard `fitBaseline` task (see [Chapter 13](#)) and by applying a low order polynomial to each WBS subband. Note that platforming is an artifact of the WBS. If you correct your spectra with `fitBaseline` specifically for platforming, you should not apply the same correction to the HRS data. Also be aware that if there was a significant continuum in the WBS data, it will be removed by `fitBaseline`. The standard pipeline checks for platforming between subbands within a `HifiTimelineProduct`. If platforming is detected, a Quality Flag, *Platforming present in overlapping subbands* is raised.

### **mkRms**

Computes the mean, rms, and medium of the level 2 data.

### **doChannelFlags**

Flags data where spurs are known to exist. This task uses data from the calibration tree to flag the data.

### **mkUncertaintyTable**

Generates a product containing a table of uncertainty values for Sideband ratio, Optical Standing Waves, Hot/Cold Load Coupling, Hot/Cold Load Temperature, Pointing, Planetary Model, and Beam Efficiency.

## 4.6. Level 2.5

The default Level 2.5 pipeline (that is used to create products in the HSA) is observing mode dependent.

- **Point modes**

HTP are stitched, folded (if Frequency Switch) and converted to `simpleSpectrum` format. HRS spectra are stitched together using the `fillGaps` option set to `True` so any gaps between subbands are filled with NaNs. The `mkRms` task is run on the data to calculate the rms noise, the output is stored in the *Statistics* node of the *Trend Analysis* product.

- **Mapping modes**

The Level 2 HTP are stitched and Frequency Switch data are folded. The HRS subbands are stitched only if they overlap in frequency (`doStitch` is called using `fillGaps=False`) in order to avoid NaNs in the cubes. In this stitching process, the subband numbers will be re-ordered by incrementing frequencies. Also, note that folding in HRS subbands can lead to invalid channels in the case that the HRS subband is small compared to the frequency throw. Pay attention to the resulting line profiles and compare them to the WBS.

The `doGridding` task is run on the resulting HTP, producing a `cubesContext` containing `cubesContexts` for each polarisation and sideband combination. If the map was taken with a non-zero rotation angle, then another set of cubes, `cubesContextRotated`, are created with the rotation angle applied.

The cubes are created with no baseline correction done prior to gridding, and it is expected that in many cases you will need to go back to the Level 2 HTP and clean the data prior to re-running `doGridding`, this can be done using the interactive Level 2.5 pipeline, see [Section 5.4](#).

The `mkRms` task is run on the cubes in order to calculate the rms noise in each pixel, the result is stored in the *Statistics* node of the *Trend Analysis* product.

- **Spectral Scans**

The `doDeconvolution` task is run, producing a `Context` called *myDecon* that contains an output of `doDeconvolution` for each spectrometer used in the observation (normally, this is the WBS-H and WBS-V). No baseline correction is done prior to deconvolution, and it is expected that in many cases you will need to go back to the Level 2 HTP and clean the data prior to re-running `doDeconvolution`, this can be done using the interactive Level 2.5 pipeline, see [Section 5.4](#).

The `mkRms` task is run on the deconvolved spectra in order to calculate the rms noise and the result is stored in the *Statistics* node of the *Trend Analysis* product.

# Chapter 5. Running the HIFI pipeline

Last updated: 4 December, 2015

## 5.1. Introduction to the Pipeline

HIFI data is automatically processed through the HIFI pipeline before it can be accessed from the the Herschel Science Archive (HSA). The HIFI pipeline is used for processing data received from one or more of the four HIFI spectrometers into calibrated spectra or spectral cubes, and comprises five stages of processing:

1. Manipulate data taken from the satellite into time ordered Data Frames (a `HifiTimelineProduct`, or HTP) for each spectrometer. This is done in the Level 0 pipeline and results in the Level 0 data Product, which is the least processed data available to the scientific community.
2. Remove backend instrumental effects - essentially a frequency calibration. There are separate pipelines for the WBS and HRS spectrometers, and the result is a Level 0.5 Product. You will not see this Product in the Observation Context unless the generation of a Level 1 product fails. However, you can generate it for yourself when reprocessing.
3. Application of observing mode specific calibrations, i.e., subtraction of reference and OFF positions, and intensity calibration using Hot/Cold loads. This is done by the Level 1 pipeline, and resulting Level 1 Products are sets of frequency (IF) and intensity ( $T_A'$ ) calibrated spectra.
4. The Level 2 pipeline removes further instrumental effects by converting to antenna temperature ( $T_A^*$ ), applying sideband gain corrections, and converting to sky frequency. Spectra at Level 2 are averaged according to LO setting and position.
5. The Level 2.5 pipeline combines the Level 2 products into final products for each observing mode:
  - Point mode spectra are stitched, folded (in the case of Frequency Switch), and converted to `SimpleSpectrum` format. HRS data are stitched only in the case that subbands overlap in frequency.
  - Mapping mode Level 2.5 HTP contain spectra that are stitched and folded (in the case of Frequency Switch), a `cubesContext` is also produced at Level 2.5 containing cubes constructed from the stitched and folded (if applicable) spectra. Stitching of HRS data is done as for point mode observations. In the case of mapping observations carried out with a non-zero rotation angle, a `cubesContextRotated` is also produced, which contains cubes generated from the Level 2.5 HTP and with the rotation angle applied.
  - The deconvolved single sideband spectra is produced at Level 2.5 for Spectral Scan observations.

There are several reasons why you may wish to run data through the HIFI pipeline yourself. For this reason, the HIFI pipeline is available in HIPE.

In theory, Level 2 and Level 2.5 products can immediately be used for scientific analysis. However, they are produced without correction for baseline issues (see [Chapter 13](#)) or standing waves (see [Chapter 12](#)). Furthermore, you may wish to change the temperature scale or reference frame of the data (see [Section 18.1](#)). You can introduce these steps into the pipeline using the *Interactive Level 2.5 pipeline*, see [Section 5.4.1](#).

In some cases, data may need to be looked at more carefully before scientific analysis can be done. You may wish to take advantage of the *Customisable Pipeline*, see [Section 5.4.2](#), to re-run all, or part of the pipeline in order to omit tasks or to change task defaults. Alternatively, you may wish to provide your own pipeline algorithm, or examine the results of each step of processing in order to improve data quality. Finally, you may wish to reprocess data in order to take advantage of new developments to the pipeline or calibration. To these ends, the Observation Context that is obtained from the HSA contains, along with the Level 0-2.5 data Products, everything you need to reprocess your observations

- calibration products, satellite data - as well as quality, logging, and history products, which you can use to identify any problems with your data or its processing.

The following sections explain how to re-run the pipeline using the `hifiPipeline` task. For a full review of all tasks used by the pipeline, please see [Chapter 4](#).

## 5.2. How to run the HIFI Pipeline



### Warning

Although it is possible to modify parameters values and add certain tasks to be processed by the `hifiPipeline`, via the *Customize Pipeline* and the *Interactive Level 2.5 Pipeline*, please note that the order in which the tasks are to be performed is very important. The expert panel, accessed by pressing the *go to expert mode...* button at the top of the GUI also contains other options for running the pipeline. These are usually only of interest to HIFI ICC calibration scientists and are not typically relevant in the context of reprocessing data for science purposes. The order of the tasks shown in the GUI (and the flow diagrams shown in [Chapter 4](#)) constitute the default flow of the tasks. You may change the order only if you have a deep understanding of the pipeline. Wrongly changing the order may lead to obscure error messages in the *Console* window, and the pipeline not completing to the end. The tasks of the pipeline are described in details in the [HIFI Pipeline Specification Document](#) if you wish to learn in-depth details of each tasks.

The HIFI ICC recommends that the data should be processed using the HIPE version that you are using for data analysis to ensure that you do not run into data-software compatibility problems. However, it is not anticipated that you will have problems if you keep within one HIPE version, e.g. perform data analysis in HIPE 13 on data that was pipelined with HIPE 12.

The `hifiPipeline` task links together the five stages of the pipeline described above and it can be used to reprocess Observation Contexts up to any level, from any level, for any choice of spectrometer(s) and polarisation(s). The pipeline can be run via a GUI or the command line.

### Opening the `hifiPipeline` GUI:

- Click once on an observation context in the *Variables* pane and the `hifiPipeline` task will appear in the *Applicable Tasks* folder, double click on it to open the Task dialogue in the *Editor* View.
- Alternatively, open the `hifiPipeline` task by double-clicking on it under the Hifi Category in the *Tasks* View.

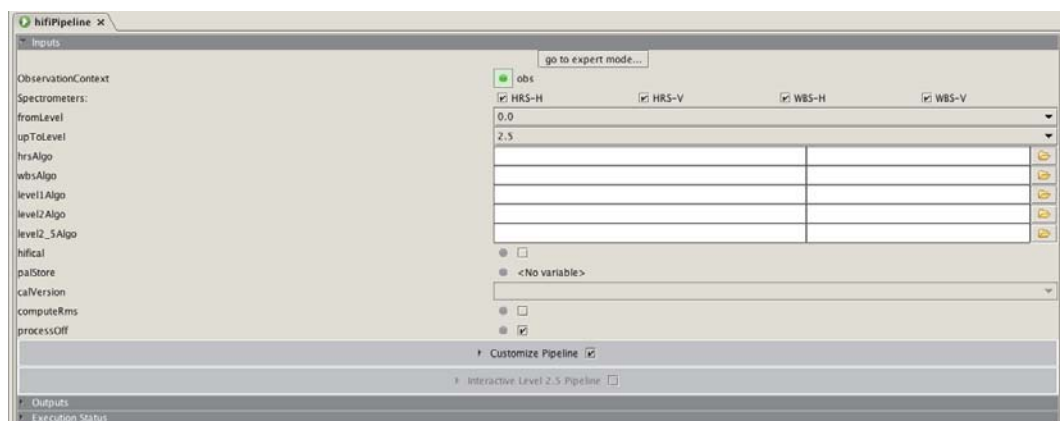


Figure 5.1. HIFI pipeline task: default view

### Running the `hifiPipeline` task:

The default (or basic) dialogue in the GUI allows you to re-process data in an observation context. It also allows you to:

- Pass any version calibration to the pipeline, see [Section 5.3](#) for details on passing a new calibration version to the pipeline.
- Configure the pipeline to your own specifications, see [Section 5.4.2](#) for information about customising the pipeline and [Section 5.4.3](#) for information about configuring the pipeline algorithms.
- Change the order of tasks and introduce new ones in the Level 2.5 pipeline with the *Interactive Pipeline*, see [Section 5.4.1](#).
- Enable or disable the algorithms to process the OFF spectra in the observation or calculate the rms noise in the Level 2 and/or Level 2.5vdata.

The default set-up of the pipeline is to reprocess data from Level 0 to 2.5 for all four spectrometers (or as many as were used in the observation), using the calibration that can be found in the Observation Context and the default pipeline algorithms.

- The way the data is to be reprocessed is defined in the *Inputs* section:
  - If the `hifiPipeline` task was opened from the *Applicable Tasks* folder then the Observation Context selected in the *Variables View* will automatically be loaded into the Task dialogue, and you will see its name by the observation context bullet, which will be green. Alternatively, drag the name of the observation context to be reprocessed from the Variables view to the ObservationContext bullet.
  - Select the spectrometers you wish to process data for by checking the desired combination of instrument(s) and polarisation(s). By default, the box for a given spectrometer is checked only if data is present at Level 0.
  - Select which levels to (re-)process 'from and to' via the drop-down menus. By default the pipeline will process Level 0 data up to Level 2.5 but you can choose any start and end point you wish. The lowest level that you can re-process from is Level 0. An option to process from Level -1 is available but it is meant for use by the HSA and the ICC. Processing from Level -1 to Level 0 (the *Level 0 pipeline*) involves the combination of the unsorted raw data and housekeeping frames, which you do not have access to.
  - Enable or disable the algorithms to process the OFF spectra in the observation with the *processOff* parameter. If enabled, the OFF spectra will be processed up to Level 2 (sky frequency and antenna temperature) and stored in the Calibration Tree in the *Pipeline-out* node. By default the algorithm is enabled, to disable it, uncheck the *processOff* box.
  - Enable or disable the algorithm to calculate the rms noise in the Level 2 and/or Level 2.5 data with the **computeRms** option. If the **computeRms** option is enabled, the `mkRms` task is enabled in both the Level 2 and the Level 2.5. You may choose to leave both actions enabled, or disable one of them. The `mkRms` task can be run in the Level 2 pipeline **and/or** the `mkRms` algorithm can be run in the Level 2.5 pipeline. The resulting tables are stored in the *Trend Analysis* node in the Observation Context. By default, this task is run during bulk reprocessing at the HSA but is turned off in HIPE because it can be time intensive. To enable the algorithm, check the *computeRms* box.
  - The remaining options are for passing new calibration to the pipeline or for customising the pipeline, and are discussed in [Section 5.3](#) and [Section 5.4](#).
- In the *Outputs* section, choose the name of the observation context that will be produced or use the HIPE default, `obs`.
- Click on *Accept* to run the pipeline. The status ("running" if all is well, error messages if not) and the progress of the pipeline are given in the *Info* section at the bottom of the Task dialogue. The command to run the pipeline is echoed in the console. You will also see more informative messages about the status of the pipeline written in the console and terminal.

The expert panel, accessed by pressing the *go to expert mode...* button at the top of the GUI contains other options for running the pipeline. These are usually only of interest to HIFI ICC calibration scientists and are not typically relevant in the context of reprocessing data for science purposes.

Below are some examples of running the `hifiPipeline` task from the command line. It is assumed that an Observation Context called `MyObs` has been loaded into the session.

```
# Reprocess an ObservationContext up to Level 2 for all spectrometers,
# without writing to memory to decrease processing time:
MyNewObs = hifiPipeline(obs=MyObs, save=False)
#
# Reprocess MyObs from Level 0.5 to Level 1, for all spectrometers:
MyNewObs = hifiPipeline(obs=MyObs, fromLevel=0.5, upToLevel=1, save=False)
#
# Now reprocess MyNewObs (which now contains data only up to Level 1) but only for
# the WBS.
# WBS-H and WBS-V are the horizontal and vertical polarisations, respectively:
MyEvenNewerObs = hifiPipeline(obs=MyNewObs, apids=['WBS-H', 'WBS-V'])
#
# What is an apid? "Application Program Identifier": it is what the pipeline calls
# spectrometers.
#
# You can retain Level 0.5
obs = hifiPipeline(obs=obs, upToLevel=2, removeLevel0_5=False)
```

#### Pipeline behaviour to note:

- The original ObservationContext (in HIPE, not on your hard disk) is overwritten with the results of running the pipeline task. As a consequence, levels higher than those you reprocessed to will be removed from the original Observation Context. In contrast, HTPs for all spectrometers are retained in the observation context regardless of what was selected but only the data for spectrometer(s) you selected will be modified.

When using the GUI, a new Observation Context variable is always created, even if you choose the output to be identically named to the input as HIPE will automatically append `_I` to the repeated variable name. However, in the command line you can ensure that no new variable is created in one of two ways:

```
# Assuming an observationContext called 'obs'
#
# 1. Choose the output name to be the same as the input
obs = hifiPipeline(obs=obs)
#
# 2. Do not specify an output variable name at all
hifiPipeline(obs=myObs)
#
```

Both approaches have the same result, just pick the one that makes most sense to you.

If you want to compare the effects of running the pipeline you will either need to reload the original Observation context or make a copy of products (spectra or HTP) before running the pipeline:

```
# Extract an HTP (here WBS-H-USB) and make a copy for comparison
htp = htp = obs.refs["level2"].product.refs["WBS-H-USB"].product
htp_orig = htp.copy()
```

- If you try to re-process from a higher level data than exists in the Observation Context, then the `hifiPipelineTask` will automatically select the highest existing level. For example, if you try to re-process from Level 0.5 to 1 but the ObservationContext only contains a Level 0 product then the pipeline will automatically run from Level 0 to Level 1.

- By default, the pipeline runs without saving the new observation context on disk. This is done to reduce the amount of time it takes to run the pipeline but is at the cost of forcing the pipeline to use more memory in HIPE in order to avoid writing to disk.

If you prefer to save the output automatically then you can switch the *save* option on by checking the box in the expert mode, or in the command line:

```
obs = hifiPipeline(obs=obs, save=True)
```

Note that unless you specify a pool to write to, using the *palStore* option (see below), then the observation will be written to `.hcss/lstore/pipeline-out`.

- When running the pipeline from the command line, the exact ordering of the parameters does not matter.
- If the pipeline is not behaving as you expect (keeping old values, for example) try resetting it:

```
hifiPipeline = hifiPipelineTask()
```

- Please note that the addition of Level 2.5 to the pipeline has made it more memory intensive. We have found that a 2 Gb machine is capable only of processing small observations (and data reduction may have to be carried out in a fresh HIPE session). Even a 4 Gb machines may have trouble processing larger maps and Spectral Scans. A more realistic memory allocation would be 6 Gb. We remind you of the option to request processing on demand from the HSA. See also [Chapter 27](#) for hints on dealing with memory problems.

### Saving the output:

There are several methods you can use to save your reprocessed observation in a pool (also known as a *local store*). See the Data Analysis Guide, [chapter 1](#) for more information about pools.)

- Right click on the output ObservationContext *obs* and select "Send to Local store".
- Alternatively, you can use `saveProduct` in the command line:

```
#
#To save MyNewObs to pool "reprocessed":
bg("saveProduct(product=myObs, pool='reprocessed', tag='My reprocessed data')")
```

Tagging the observation can help to quickly search for the correct version of the observation for a future HIPE session.

- In addition, when you run the pipeline, you can specify which pool the output should be written to. In the console type,

```
name="My-pipeline-out"
pool=ProductStorage(LocalStoreFactory.getStore(name))
```

and drag `pool` to the `palStore` bullet in the GUI.

Or completely in the command line:

```
# Specify the pool to which the pipeline should write output:
name="My-pipeline-out"
pool=ProductStorage(LocalStoreFactory.getStore(name))
MyNewobs = hifiPipeline(obs=Myobs, palStore=pool, save=True)
#
```

Using this option will result in the pipeline taking longer to run.

### Running the pipeline for multiple observations:

The script below will retrieve a list of Observation Contexts from pools, reprocess them, then save the reprocessed data in another pool.

```
list = [1342xxxxxx, 1342yyyyyy, 1342zzzzzz]
for obsid in list:
    obs=getObservation(obsid)
    obs = hifiPipeline(obs=obs, save=False)
    pool = '%i_new' % (obsid)
    saveObservation(obs, poolName=pool)
```

The script assumes that the original Observation Contexts are stored in pools with the name of the observation number, i.e., observation 1342190798 is stored in a pool with name 1342190798, and in this case, the output pool is 1342190798\_new.

## 5.3. How to process with new (or different) calibration data

Data that you retrieve from the HSA is processed with calibration data appropriate for the version of the pipeline used to populate the HSA during Standard Product Generation (SPG). You can identify the pipeline version by looking at the *SPG Version* in the Observation Context Summary; the SPG version corresponds to the version of HIPE used. Note that when you reprocess yourself, the SPG Version gets updated. You can also look at the build version of the last pipeline task reported in the *HistoryTasks* of the History Product to check the last version of the pipeline applied to the data.

ObservationContext for HIFI data of observation 1342231447			
Summary			
AOR label:	CII_1901-B_DBS - W51	Obs. ID:	1342231447
Instrument:	HIFI	Obs. Date:	2011-10-25T19:44:20Z
Object:	W51	Obs. Mode:	DBS fastChop
AOT:	Single Point	Dec. Nominal:	14° 30' 30.5"
RA Nominal:	19h 23m 43.9s	Operational Day:	895
SPG Version:	SPG v7.3.0		

Figure 5.2. Checking the version of the pipeline used at the HSA

- **Where can I find information about calibration updates?**

Updates to the HIFI calibration data are generally concurrent with the release of each major version of the HCSS-HIFI software. It is possible to have updates to the calibration data in between major releases of the software as the software and data are independent of each other. Information about calibration versions and their contents can be found on the [HIFI Instrument and Calibration page](#), which is updated whenever new calibration is available.

- **How do I find the calibration version used to process my data?**

The metadata item `calVersion` can be found in all calibrated products (Level 1 and 2), and the version number has the format `HIFI_CAL_version_number`.



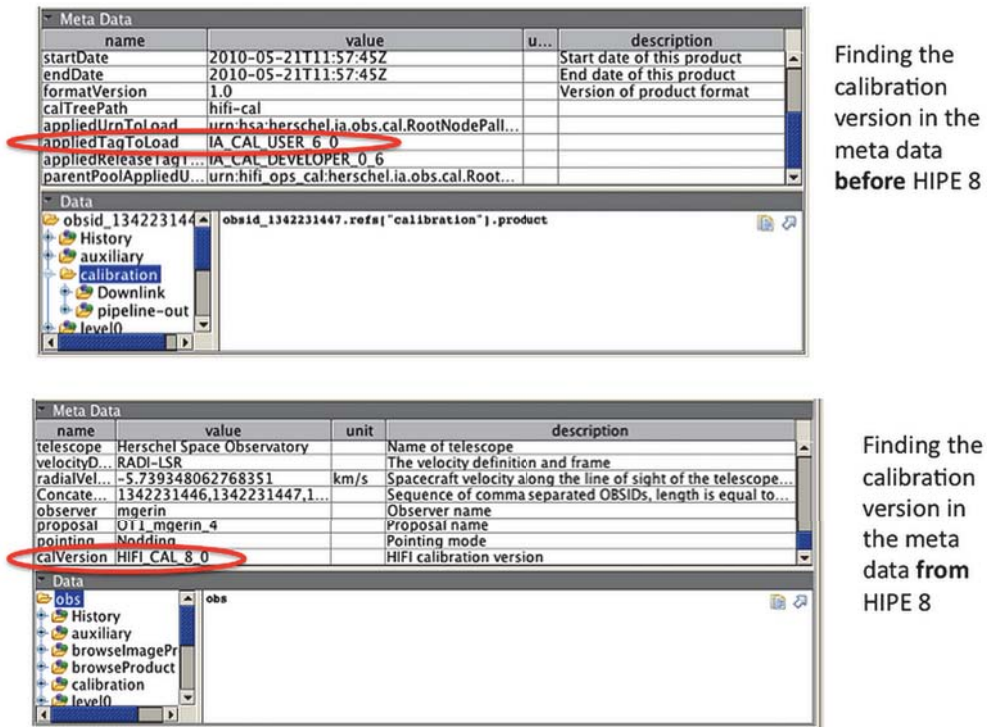


Figure 5.3. Checking the calibration version used prior to HIPE 8 (top) and from HIPE 8 (bottom).

- **Which calibration version should I use?**

There is no need to use different calibrations for different science goals with HIFI data. Instead, it is recommended to keep data version, calibration version, and software version compatible with each other. For example, this means using HIPE 13.0 to work with data that has been processed in HIPE 13.0, with calibration that was released with HIPE 13.0. This can be done by downloading HIPE and data from the HSA following bulk reprocessing, or by reprocessing the data yourself with the latest calibration version available after each HIPE release.

However, you may wish to use an older calibration version, for example, in order to more directly compare recently acquired data with older data, and it is possible to pass older versions of the calibration to the pipeline. Consult the [HIFI Instrument and Calibration page](#) to understand the differences between the different calibration versions. However, whilst every effort is made to ensure backwards compatibility, using old calibration with newer software can cause inconsistent results.

- **How do I re-process with a new (or different) calibration version?**

The default action of the pipeline is to reprocess the observation using the latest calibration that is already stored in the observation context. However, you can configure the pipeline to allow you to pass an older calibration version to it, or to access the latest calibration version held in the HSA, which requires an internet connection. This can be done with the following command (there is no way to do this in the GUI):

```
cal = configureHifiPipeline(useHsa=True)
```

In the GUI, you should then:

- drag *cal* from the Variables pane into the *palStore* bullet in the GUI,
- check the *hifical* box,

- select the calibration version you want from the drop-down menu, which automatically selects the latest calibration version available from the HSA from the list of all available calibration versions

You can then set up and run the pipeline as desired.

A completely command line example is below:

```
obs = getObservation("1342205520")
cal = configureHifiPipeline(useHsa=True)
obs_1 = hifiPipeline(obs=obs, cal=True, palStore=cal, calVersion="HIFI_CAL_18_0")
```

If you do not configure the pipeline but set `cal=True` then the pipeline will send a message to the console warning you that new calibration could not be updated, and reprocessing commences using the calibration available in the Observation Context. If you omit the `calVersion`, then the latest calibration version at the HSA will be used.

Configuring the pipeline as above also configures the pipeline to save output after each level is completed to a default location, which is `.hcss/pipeline-out`, when the `save=True` option is used (or the `save` box is checked in the expert pipeline panel):

```
obs = getObservation("1342205520")
cal = configureHifiPipeline(useHsa=True)
obs_1 = hifiPipeline(obs=obs, fromLevel=0.0, upToLevel=2.0, palStore=cal,
    save=True)
```

If you forget to configure the pipeline before running with `save=True`, and also do not provide a pool yourself for the pipeline to write to, then the pipeline runs to completion but without saving anything. You can then save the output yourself.

- **Downloading the calibration pool from the HSA**

Accessing the HSA every time you wish to reprocess data with a new calibration version may not be very efficient and requires that you have an internet connection. It is also possible to download the latest calibration available at the HSA into a local pool, which can be stored on your disk. This avoids the need for an internet connection every time you run the pipeline and makes reprocessing of multiple observations much more efficient.

The following command will get the latest calibration from the HSA and store it in a pool called `hifi-cal` (location `.hcss/lstore/hifi-cal`), which is the default location for the pipeline to obtain calibration data when not using the HSA. You only need to do this again when you want to update the calibration from the HSA. The length of time it takes to download the calibration pool depends on your internet connection; be aware that it is not a quick process.

```
hifical = getHifiCal(useHsa=True)
```

To pass this pool for use in the pipeline:

```
store=ProductStorage(["pipeline-out", "hifi-cal"])
obs_new = hifiPipeline(obs=obs, palStore=store, cal=True)
```

The "pipeline-out" is required as a precaution because of a subtlety in how HIPE deals with pools; the first pool registered is writeable. By default, the pipeline will write its output to pool and you do not want to risk polluting your calibration pool with a processed observation. Therefore, we specify a pool for the pipeline to write out to. In fact, it is the same output pool that is used in the pipeline configuration above.

Note that you do not need to download different versions of the calibration tree in order to apply different calibration versions to your data, the calibration tree contains all the past calibration versions already.

- **Getting calibration products**

The calibration contained within an `ObservationContext` can be obtained with the following:

```
cal = obs.getCalibration()
```

These calibration products will only apply to this observation and only represent a part of the HIFI calibration tree. The contents of the calibration tree are described in [Section 3.1](#).

To get the full HIFI calibration tree from the hifi-cal pool:

```
from herschel.ia.obs.cal import CalTreeFactoryManager
from herschel.hifi.cal import HifiCalTreeFactoryPalImpl
factory = CalTreeFactoryManager.getInstance()
store = ProductStorage("hifi-cal")
factory.setProductStorage(store)
registry = factory.getRegistry()
# user release version of HIFI calibration tree
calroot = registry.checkout()
# all versions of the calibration tree contained in the storage
state = registry.state
```

- **Re-pipelining with a new Auxiliary product**

An additional option, `aux=True`, allows to reprocess data with updated auxiliary information, e.g. data from the satellite regarding timing and position of the observation. This option is intended for the use of HIFI calibration scientists and is described here for the sake of completeness. This option requires connection to a dedicated data-base that is distinct from the HSA, which hosts the same version of the auxiliary products used in the latest bulk reprocessing.

```
obs = getObservation("1342205520")
cal = configureHifiPipeline(useHsa=True)
obs_1 = hifiPipeline(obs=obs, cal=True, aux=True, palStore=cal)
```

## 5.4. Modifying the pipeline

You may wish to modify the pipeline for the following reasons:

- **To allow existing data processing tasks to be included as part of the pipeline**

For example, you may wish to include baseline or standing wave removal prior to gridding mapping data or deconvolving spectral scans, or you might want to convert the data to Jy at the end of the pipeline. This can be done using the *interactive Level 2.5 Pipeline*, see [Section 5.4.1](#).

- **To omit steps from the pipeline or to use optional steps in the pipeline**

For example, you may wish to omit the averaging step at the end of the Level 2 pipeline or to turn on the optional `doFilterLoads` Level 1 pipeline step to help mitigate standing waves coming from the calibration loads. This can be done for Levels 1 and 2 using the *Customize Pipeline* tab in the `hifiPipeline` GUI, see [Section 5.4.2](#).

In the Level 2 pipeline, you could choose to use `doMainBeamTemp` to convert to main beam temperature instead of the default antenna temperature. This can be done by *editing the pipeline algorithm scripts* found in the *Pipelines* menu in HIPE, see [Section 5.4.3](#).

- **To calculate the RMS noise in your data**

In the SPG environment (the pipeline run to populate the HSA), the pipeline computes the rms noise in Level 2 and Level 2.5 data with the `mkRms` task. However, the task can take significant CPU time to run. For this reason, the rms calculation is turned off by default in the Interactive Pipeline (the pipeline you work with in HIPE). The `mkRms` task is enabled via the `hifiPipeline` option **`computeRms`**. Once the option **`computeRms`** is enabled, you may choose to run `mkRms` in Level 2 and/or Level 2.5.

The `mkRms` task can be used in both the Level 2 and Level 2.5 pipelines. At Level 2.5, the `mkRms` task is used as part of an observing mode specific algorithm, in addition, calculations are made for data converted to Main Beam temperature and also for combined H and V polarisation.

If you want to calculate the rms noise in your data yourself, you must run the pipeline with the `computeRms` option set to *True*. To do so, check the `computeRms` box in the pipeline GUI. In the command line, use:

```
obs = hifiPipeline(obs=obs, computeRms=True)
```

Checking the `computeRms` box in the pipeline GUI will cause the `mkRms` boxes in the *Customize Pipeline* and *Interactive Level 2.5 Pipeline* sections in the GUI to be checked. The `mkRmsAlgo` box in the *Interactive Level 2.5 Pipeline* will also be checked. This means that the calculations are automatically done at Level 2 and Level 2.5 but it is possible to limit the calculations to be done only to one Level or the other by unchecking the appropriate `mkRms` box.

The various options and parameters that can be modified in the `mkRms` are described in the [mkRms](#) in *HIFI User's Reference Manual* entry in the HIFI User's Reference Manual. It is also possible to modify the `mkRms` algorithm in the Interactive Level 2.5 pipeline. This is described in [Section 5.4.3](#) below.

The resulting tables of statistics are stored in the *Statistics* node of the *trendAnalysis* Product in the Observation Context, and can be used for comparing with expected noise performance. **Please note** that if you choose to reprocess part of your data, the *Statistics* node will be updated to reflect the data reprocessed and a copy of the statistics from the unprocessed data will still be present in the *Statistics* node. This is because the original statistic data i.e the one created with SPG is already present in the *trendAnalysis* Product and is not erased.

If you wish to remove all statistics information from the *trendAnalysis* Product before reprocessing with `hifiPipeline`, we show an example in this following script:

```
## obs is your observationContext
# To remove the existing statistics from Level 2.5

map=java.util.HashMap(obs.refs["trendAnalysis"].product.refs["Statistics"].product.refs)
for ref in map:
    if( ref[0:8]== "Level2_5"):

        obs.refs["trendAnalysis"].product.refs["Statistics"].product.refs.remove(ref)

# here, ref[0:8] corresponds to the number of characters for 'Level_2_5'
# note that you could equally have used: if ref.startswith("Level2_5"):
# at this point there are no statistics for Level 2.5 in obs

obs=hifiPipeline(obs=obs, fromLevel=1, upToLevel=2, computeRms=1)

# and at this point there are only statistics up to Level 2.0 in obs

# If, instead, you run the following commands

map=java.util.HashMap(obs.refs["trendAnalysis"].product.refs["Statistics"].product.refs)
for ref in map:
    if( ref[0:6]== "Level2"):

```

```

obs.refs["trendAnalysis"].product.refs["Statistics"].product.refs.remove(ref)

# at this point all statistics of Level 2.0 and 2.5 in obs are removed

obs=hifiPipeline(obs=obs, fromLevel=2, apids=["WBS-H"], computeRms=1)

# at this point there are only statistics for WBS-H Upper/Lower bands in Level
2.5.

# Finally, if you would like to remove the lot, regardless of the level ...

for ref in map:
    obs.refs["trendAnalysis"].product.refs["Statistics"].product.refs.remove(ref)

```

- **To use alternative options in the default steps of the pipeline**

One example includes choosing a different order Polynomial to fit, or a different interpolation method in the `mkOffSmooth` step of the Level 1 pipeline in order to improve baselines in the data. This can be done using the *Customize Pipeline* tab in the `hifiPipeline` GUI, see [Section 5.4.2](#), or, for those more comfortable with scripting, by *editing the pipeline algorithm scripts*, see [Section 5.4.3](#).

- **To introduce a task you have written into the pipeline**

This is expected to be of interest only to more expert users and can be done by creating a task to pass to the *interactive Level 2.5 pipeline*, see [Section 5.4.1](#), or by *editing the pipeline algorithm scripts*, see [Section 5.4.3](#).

## 5.4.1. Using the interactive Level 2.5 pipeline

The default Level 2.5 pipeline (that is used to create products in the HSA) is observing mode dependent, and this is reflected in the Interactive Level 2.5 Pipeline GUI.

- **Point modes**

HTP are stitched, folded (if Frequency Switch), and converted to `simpleSpectrum` format. HRS spectra are stitched together using the `fillGaps` option of `doStitch` set to `True` so that any gaps in the subbands are set to NaNs. The tasks listed in the Interactive Level 2.5 Pipeline GUI are `doStitch`, `doFold` (only for Frequency Switch observations), and `convertSingleHifiSpectrum`

- **Mapping modes**

HTP are stitched, folded (if Frequency Switch), and gridded. HRS spectra are stitched together only if the subbands overlap in frequency (by calling `doStitch` with `fillGaps=False`) in order to avoid NaNs in the cube. The tasks listed in the Interactive Level 2.5 Pipeline GUI are `doStitch`, `doFold` (only for Frequency Switch observations), and `doGridding`. Note that in the Level 2.5 Interactive Pipeline, `doGridding` automatically applies the rotation angle used in the observation to the gridding. This is a contrast to the pipeline used to populate the HSA, which creates non-rotated cubes and, in the case of maps carried out with a non-zero rotation angle, also a set of rotated cubes.

- **Spectral Scans**

The data are deconvolved using the `doDeconvolution` task, which is listed in the Level 2.5 Interactive Pipeline GUI.

- **All observing modes**

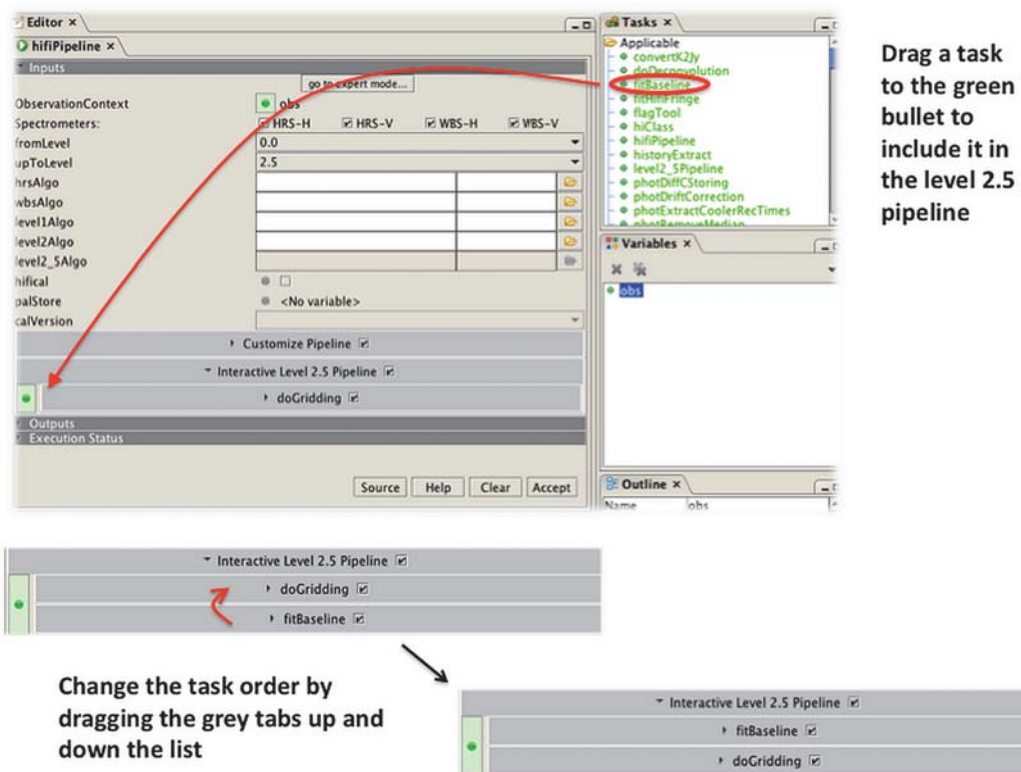
The `mkRmsAlgo` algorithm, which uses the `mkRms` task is run on the Level 2.5 data. For each type of observing mode, the `mkRmsAlgo` and `mkRms` tasks are listed in the Interactive Level 2.5 Pipeline GUI.

The interactive Level 2.5 Pipeline allows you to set up the tasks, automatically included in it, as you wish prior to running the pipeline. It also allows you to introduce other tasks into the pipeline, with the requirement that the output of one task be able to be passed to the next task in your customised Level 2.5 pipeline.

By default, the interactive Level 2.5 pipeline tab is inoperative when the pipeline GUI is first opened. To use it, check the box to the right of "*Interactive Level 2.5 Pipeline*" near the bottom of the `hifiPipeline` GUI, and open the tab by clicking on the arrow to the left. Tasks already included in the Level 2.5 pipeline can also be opened by clicking on the arrow to the left of the task name, and the task options can be modified. See [Chapter 14](#) and [Chapter 15](#) for information about setting up `doDeconvolution` and `doGridding`, respectively. Note that upon checking the "*Interactive Level 2.5 Pipeline*" box, these tasks will be initialised with their default settings: these may not be the same as in the standard pipeline. For example, `doGridding` will produce only cubes with the `flyAngle` applied when run this way, but in the standard pipeline, it would produce cubes with the `flyAngle` applied, and cubes to which no rotation is applied.

Other tasks that accept and pass HTPs can be added to the Level 2.5 pipeline. To find tasks that are applicable to HTPs, click on an HTP in the *Variables* view or in the Observation Context tree, and look under *Applicable* in the *Tasks* View. You can also pass tasks that loop through HTPs in an Observation Context, such as `doDeconvolution`, `fitBaseline`, `fitHifiFringe` and `convertK2Jy`. These can be found by clicking on an Observation Context in the *Variables* view or in the Observation Context tree before looking under *Applicable* in the *Tasks* View.

To pass a task to the Level 2.5 pipeline, drag it from the *Tasks* View to the green bullet in the tab, see [Figure 5.4](#). The tasks can be re-ordered by dragging them up and down the list, and can be set up as desired after clicking on the arrow to the left of the task name. Note that adding tasks to the list will remove any changes you made to a panel already in the list so it is best to add all the tasks you want to use first, and then set them up.



**Figure 5.4. Using the Level 2.5 Interactive Pipeline GUI**

As usual, the command to run the pipeline with your customised Level 2.5 pipeline will be echoed to the console and can be copied to insert into scripts. The output to the console will also include the

commands to customise the Levels 1 and 2 pipelines, unless you unchecked that option in the GUI. If you did not modify the Levels 1 and 2 pipelines then you can safely remove these by deleting `params = { ... }`.

### Writing your own task to provide to the Level 2.5 pipeline

You can provide your own task to run the Level 2.5 pipeline. The example below shows the framework of how to do that:

```

from herschel.ia.task import Task
from herschel.ia.task import *
from herschel.ia.gui.kernel import ParameterValidatorAdapter,
    ParameterValidationException
from herschel.ia.all import *

class HtpPrimeInputValidator(ParameterValidatorAdapter):
    def validate(self, val):
        if not isinstance(val, HifiTimelineProduct):
            msg = "The prime input is a %s. It must be a %s." % (val.__class__,
                HifiTimelineProduct)
            raise ParameterValidationException(msg)

class HifiPipelineTemplateTask(Task):
    """
    @jhelp A task to demonstrate how to write a task that is compatible with the
    the hifi interactive pipeline.

    @jcategory HIFI/Analysis

    @jalias hifiPipelineTemplate

    @jparameter htp, IO, HifiTimelineProduct, MANDATORY, None
    The number of the observation context to be compared

    @jparameter otherTaskParameter, INPUT, String, OPTIONAL, 'empty'
    Example of a non-prime task parameter

    @jexample How to register this task in HIPE and use it
    # Import your task if it is contained in the software build
    from herschel.hifi.scripts.users.share.HifiPipelineTemplateTask import *
    # Register your task in HIPE, and move task to the interactive pipeline (GUI)
    hifiPipelineTemplate = HifiPipelineTemplateTask()
    # To execute your task on the command-line
    output = hifiPipelineTemplate(obs=xxxx, otherTaskParameter='myInput')

    @jhistory 2012-02-13 KE Initial version
    """
    def __init__(self):
        self.setName("hifiPipelineTemplate")
        self.setDescription("A task to demonstrate how to write a task that is " +
            "compatible with the the hifi interactive pipeline")
        # HifiTimelineProduct - htp
        p = TaskParameter()
        p.name = 'htp'
        p.type = TaskParameter.IO
        p.valueType = HifiTimelineProduct
        p.nullAllowed = 0
        p.defaultValue = None
        p.description = 'The obsid of an ObservationContext'
        p.mandatory = 1
        p.parameterValidator = HtpPrimeInputValidator()
        self.addTaskParameter(p)
        # Non-prime task parameter
        p = TaskParameter()
        p.name = 'otherTaskParameter'
        p.type = TaskParameter.IN
        p.valueType = String
        p.nullAllowed = 0
        p.defaultValue = 'empty'
        p.description = 'example of a non-prime task parameter'

```

```

    p.mandatory = 0
    self.addTaskParameter(p)
#####
def execute(self):
    print self.getName() + " is being executed."
    httpCopy = self.http
    print httpCopy
    #self.http = HifiTimelineProduct()
    #self.http.setDescription("This is a test output")
#####
hifiPipelineTemplate = HifiPipelineTemplateTask()
#output = hifiPipelineTemplate(http=http, otherTaskParameter='myInput')
#print output

```

## 5.4.2. Customising the Level 1 and 2 pipelines

The *Customize Pipeline* section of the `hifiPipeline` GUI can be used to change the defaults used in the pipeline algorithms, and to omit steps from the Level 1 and 2 pipelines.

- Unhide the *Customize Pipeline* section by clicking on the arrow, all of the steps in the Levels 1 and 2 pipelines are displayed.
- If an Observation Context is loaded into the `hifiPipeline` GUI then `doPipelineConfiguration` will read the observing mode from the `instMode` (**not** the `obsMode`) metadata item, and the steps that are not applicable for that observing mode will be greyed out.

To find the value of the `instMode` metadata, you can either directly look in the Observation Context metadata, or use the following in the command line:

```
obsmode = obs.meta.get("instMode").value
```

- By unchecking the tick box by each pipeline step name, you can omit that step when you run the pipeline.
- By un hiding the panels in the GUI, you can see the parameters used in each step with the default settings for that observing mode, which you can modify as you prefer. Tooltips found by hovering over parameter names give indications of the options available to you, while more information is available in [Chapter 4](#). The pipeline steps are described in detail in the [Pipeline Specification Document](#).
- The command to run the pipeline as you have configured it is echoed to the console, allowing you to transfer your customised pipeline command to a script. This is a very simple way to put a customised pipeline into a script but you should be aware that this will *only* work for other observations of the *same* observing mode as the configuration is dependent on the observing mode and a different configuration must be created for a different mode.
- You can also customise the pipeline directly from the command line (without copying the echo to console) using `doPipelineConfiguration`, although the format is somewhat different than that echoed to the console. The echo to the console from the pipeline includes all the default settings for the observing mode but when using `doPipelineConfiguration`, it is only necessary to include the settings for the tasks that you wish to modify. In the example below, the `doAvg` step is omitted:

```

# This passes the instMode metadata value to the pipeline configuration
config = doPipelineConfiguration(obs)
# Now change the pipeline configuration to skip doAvg
config.setParameter('doAvg','ignore', True)
obs_1 = hifiPipeline(obs=obs, params=config)

```

And in this example, you may want to add `doFilterLoads` to the configuration:



```
# This passes the instMode metadata value to the pipeline configuration
config = doPipelineConfiguration(obs)
# Now change the pipeline configuration to add doFilterLoads
config.setParameter('doFilterLoads', 'filterMethod', 'cubic_splines')
obs_2 = hifiPipeline(obs=obs, params=config)
```

### 5.4.3. Editing the pipeline algorithms

The scripts for the algorithms of each stage of the pipeline and the algorithm for calculating the rms noise, `mkRmsAlgo`, can be found in the Pipeline → HIFI menu in the HIPE toolbar. These scripts are kept automatically up to date and they have been made as clear and well-commented as possible in order that you be able to modify them. However, it is strongly recommended that you look at the [Pipeline Specification Document](#) to understand what the pipeline does, and what alternatives are available for each pipeline step.

- Edit the pipeline algorithm script (optional), and save it on disk. You can then pass the modified algorithm to the pipeline by specifying the location of the saved script in the appropriate `algo` field of the GUI. You can browse for your saved script by clicking on the folder icon.
- You can also pass the path to the saved algorithm script in the command line, e.g.,

```
obs = hifiPipeline(obs=obs, level2Algo="/home/me/MyScripts/
MyLevel2PipelineAlgo.py")
```

- Alternatively, you can make use of the fact that the pipeline algorithm scripts define a function (`def TaskName (Parameters) :`), and pass the function name to the pipeline. To do this you must first compile your new pipeline algorithm by running the script with the double arrows (`>>`) in the HIPE toolbar. In the example below, the function has been called `MyLevel1Algo`:

```
# Include your own algorithm for the Level 1 pipeline, for all spectrometers,
  from Level 0 to 1:
obs = hifiPipeline(obs=obs, fromLevel=0, upToLevel=1, level1Algo=MyLevel1Algo)
#
```

### 5.4.4. Running the Pipeline step by step

- If you choose to modify or customise the pipeline, it can be helpful to see directly and quickly what changes will be made to the data. Running the pipeline, or one part of the pipeline, step by step allows you to inspect the results of each step and change the default parameters of the pipeline. You will find extended information of every steps the pipeline perform in [Chapter 4](#). If you wish to create your own algorithm, which must be written in jython, for a part of the pipeline, then this will likely be your first step.
- It is not expected that there will be much need to customise the spectrometer pipelines (up to Level 0.5), and indeed there are only a few steps of the spectrometer pipelines that have some options. It is more likely that you may wish to play with how *OFF* and *reference* spectra are subtracted in the Level 1 pipeline, although it is expected that the default settings should work well.
- To step through the pipeline, you must work directly on the appropriate level `HifiTimeLine` (HTP - the dataset containing all the spectra, including calibration spectra, made during an observation for a given spectrometer). So the first thing you must do is to extract the HTP you want to work on from your Observation Context:
  - Drag an HTP from the Observation Context tree in either the *Context Viewer* or *Observation Viewer* into the *Variables* view, and rename it if you desire by right clicking on the new variable and selecting "rename".

- In the command line, the formalism to extract an HTP is

```
http = obs.refs["level2"].product.refs["HRS-V-USB"].product
```

"level2" and "HRS-V-USB" should be replaced by the level and backend combination desired.

- When you select an HTP in the Variables view in HIPE, you will notice many tasks with names like `DoWbsDark`, `mkFreqGrid`. These are the names of all the steps in the HIFI pipeline; `mk...` signifies a step where a calibration product is being made, `Do...` is a step where a calibration is applied. You can step through the pipeline using these tasks, you will need to refer to the [HIFI Pipeline Specification Document](#) for the order that the steps should be applied in. Alternatively, you can use and modify the scripts that are supplied with the software from the Pipeline menu in HIPE, as described above. To learn about moving about and running scripts, see the [HIPE Owners Guide](#).
- For information on the steps of each level of the pipeline (their names, the order to run them in, and what options you can change) see the [HIFI Pipeline Specification Document](#).

# Chapter 6. Viewing Spectra

Last updated: 28 April, 2015

## 6.1. Introduction

HIFI spectra can be easily visualised using the *Spectrum Explorer* graphical interface, which is described in detail in the *Herschel Data Analysis Guide*. For spectra, see the [How to display spectra](#) section of the [Spectral Analysis](#) chapter, and for spectral cubes, see the [How to display the spectra in cubes](#) section of the [Spectral analysis for cubes](#) chapter. Here, we briefly describe the use of the Spectrum Explorer with an emphasis on its features for HIFI data.

You can also create plots using scripting methods, which is helpful to check output from data reduction scripts and to make figures for papers. There are two packages available to use. *SpectrumPlot* allows you to plot `SpectrumDatasets` with just a few commands but has relatively limited display options. *PlotXY*, on the other hand, requires you to get your spectrum into `Numeric1d` format before plotting but once that hurdle is passed it is possible to make sophisticated figures. The [Plotting](#) chapter in the "Herschel Data Analysis Guide" covers this topic in detail. Here we provide examples, details and tips tailored to HIFI data.

## 6.2. How to look at HIFI spectral data

HIFI spectra, spectral cubes, and HTP can be conveniently viewed in the Spectrum Explorer.

### 6.2.1. Spectra

To view a spectrum in the Spectrum Explorer, click on a `SpectrumDataset` in the Variables pane or in the Observation Context tree in the Observation Viewer with the right mouse button, and select `Open With` → `SpectrumExplorer`. This will open the Spectrum Explorer in the space used by the *Editor View* in HIPE. If the Spectrum Explorer is already selected as the default viewer, then a double-click will suffice to open the spectrum in the Spectrum Explorer. The default viewer is denoted by a circle to the left of the viewer name in the *Open With* menu, whichever was last chosen to view a given data type will become the default viewer for that datatype.

To get to the spectrum in a HIFI Observation Context, you need to click on the data level of interest (Level 0, 1, 2, or 2.5) and then the spectrometer of interest, spectra are stored within `DatasetWrappers` called *boxes*, see [Figure 6.1](#). Left clicking on a spectrum in the Observation Viewer opens the Spectrum Explorer, and open within the *Editor View* but keeps the Observation Context tree visible to the left, see [Figure 6.1](#). This allows you to visualise the contents of the Observation Context more quickly. In order to have more space to work, you can enlarge the Spectrum Explorer to fill the entire Editor View by clicking on the arrow in the top-right corner of the Spectrum Explorer.

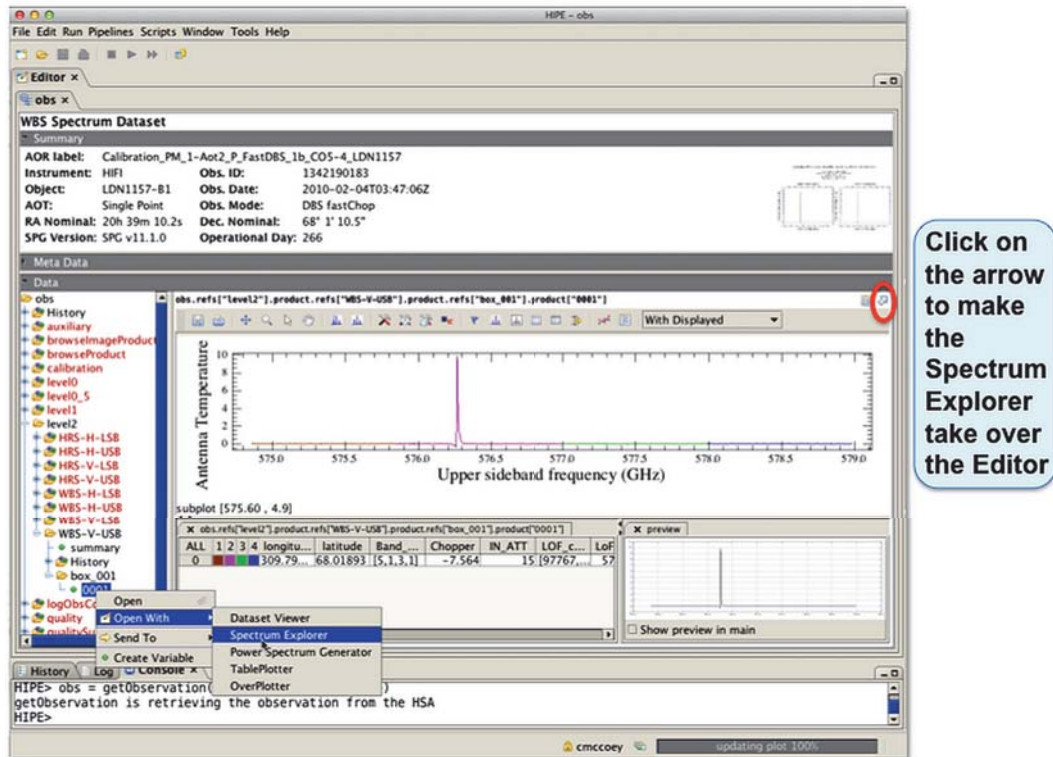


Figure 6.1. Opening the Spectrum Explorer on a HIFI Level 2 spectrum

Beneath the plot in the Spectrum Explorer is the *Data Selection Panel*. Each spectrum within the plotted *SpectrumDataset* is displayed on a different row in the Data Selection Panel, and each subband is represented by a different column. Note that each subband is considered to be an individual spectrum. You can plot all the spectra in a dataset by pressing the *All* button at the top left of the Data Selection Panel, and remove them all by pressing it again. All the spectra in a row or column can be plotted by clicking on the number of the row/column, and removed by clicking on it again. Individual spectra can be plotted by clicking in the box corresponding to it in the Data Selection Panel, and removed from the plot by clicking again. If your dataset contains only one row of spectra then they will be plotted immediately upon opening the Spectrum Explorer, while nothing will be plotted until it is selected in the case of datasets containing multiple rows of spectra. The remaining columns in the Data Selection Panel show the values of attributes in HIFI data, these can be sorted on and used to filter what data is displayed in the plot.

You are directed to the [How to display spectra](#) section of the 'Herschel Data Analysis Guide' for the complete documentation of the usage of the Spectrum Explorer.

### 6.2.1.1. HIFI Tool in Spectrum Explorer

The *HIFI Tool* extends the Spectrum Explorer to provide a convenient way to display different axes on HIFI spectra (only, spectral cubes are not supported). Note that this is for display only, the data are not converted. The tool is opened by clicking on the HIFI ICC icon in the Spectrum Explorer button bar. A panel opens to the right of the plotted spectrum.

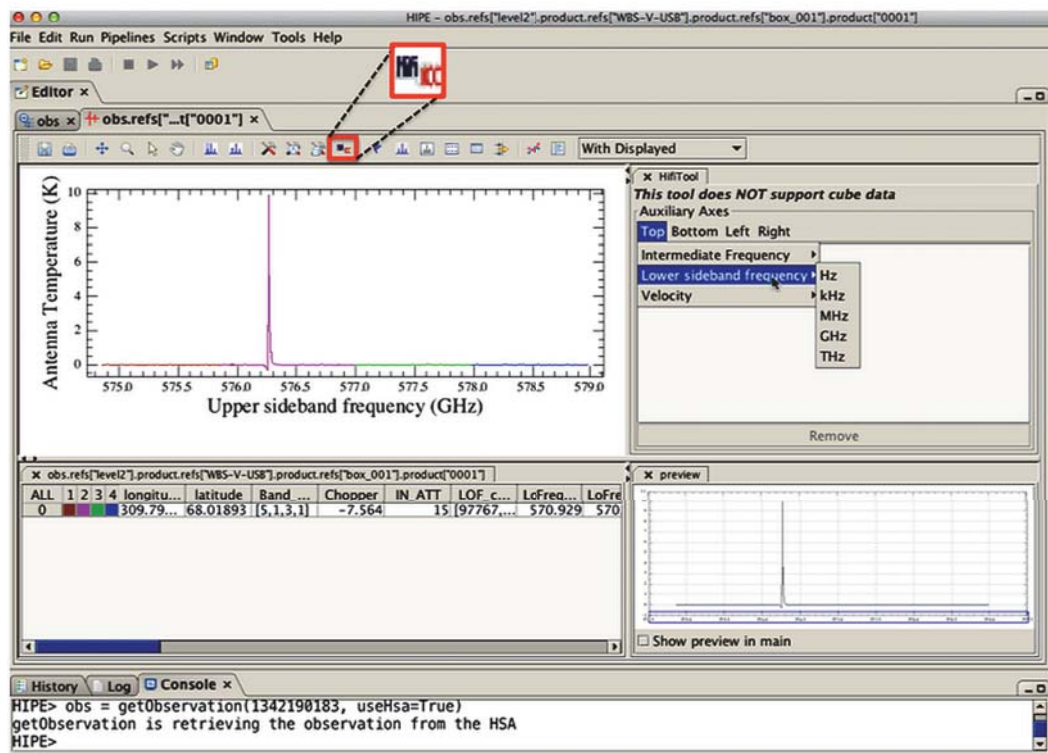


Figure 6.2. Opening the HifiTool in Spectrum Explorer on a HIFI Level 2 spectrum

To add a new axis or change an existing one, click on the axis location (top, bottom, left, right) in the panel and click on the type of axis you want, then select the units from the menu that appears to the right. For the top and bottom axes, you have the choice of the following axis types:

- IF (MHz)
- Upper or Lower Sideband (Hz, kHz, MHz, GHz, THz)

Only the 'other' sideband is offered, so if your data is USB then only LSB is offered.

- Velocity (m/s, km/s)

The calculation is done using `convertWavescale` and a pop-up box appears for you to enter the reference frequency.

For the right and left axes you can choose to display flux density values. The calculation is done using the `convertK2Jy` task and a pop-up box appears for you to enter the source size.

You can add as many axes as you wish. Each axis is identified in the panel with the following label convention *axis location\_axis type\_unit\_axis number*. To delete an axis from the plot, click on the label in the panel and then click on the delete button at the bottom of the panel.

## 6.2.2. Spectral Cubes

HIFI cubes are found in Level 2.5. The Level 2.5 context contains a `cubesContext`, which itself contains cube contexts for each spectrometer used in the observation (for both USB and LSB), and these contain a cube (technically a `SimpleSpectralCube`) made of the stitched subbands, see [Figure 6.3](#).

Spectral cubes are opened in the Spectrum Explorer via a right mouse click in the same way as spectra, see above. There are several useful options available for cubes in the *Open With* menu:

1. Spectrum Explorer is used for visualising spectra in cubes and gives access to the cube toolbox.

2. WCS Explorer allows you to see the cube header, which follows usual fits conventions.
3. Standard Cube Viewer allows you to inspect images of each layer of the cube (the same can be done in Spectrum Explorer).

These three options are shown in [Figure 6.3](#) and are found by right clicking on the `SpectralSimpleCube` variable (`cube_WBS_H_USB_1` in the Figure). Below that are three datasets containing the data for the image, the weights in the cube and the flags in the cube, these can be viewed with the *Dataset Viewer*. Note that the default layer displayed in the Spectrum Explorer is the middle layer of the cube, while the Standard Cube Viewer, by default, shows the last layer in the cube and shows it on a very zoomed out scale - you will probably have to zoom to fit to see the image.

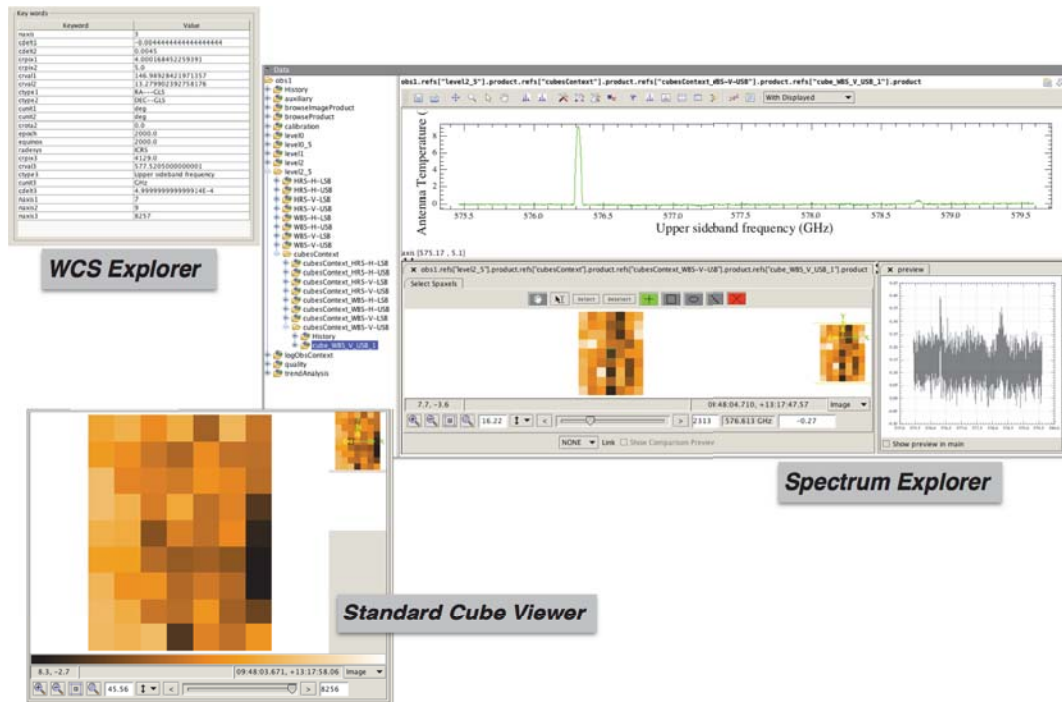


Figure 6.3. Opening the Spectrum Explorer on a HIFI spectral cube

Rather than the Data Selection Panel used for spectra in the Spectrum Explorer, as described above, the selection panel for cubes shows an image of the cube. There is nothing particular to HIFI data about the usage of the Spectrum Explorer with HIFI cubes and you are directed to [How to display the spectra in cubes](#) for the full documentation of using the Spectrum Explorer with spectral cubes. However, you should note that the pixel coordinates reported beneath the cube are reported as (Y, X) rather than the more usual (X, Y).

### 6.2.3. HifiTimelineProducts (HTP)

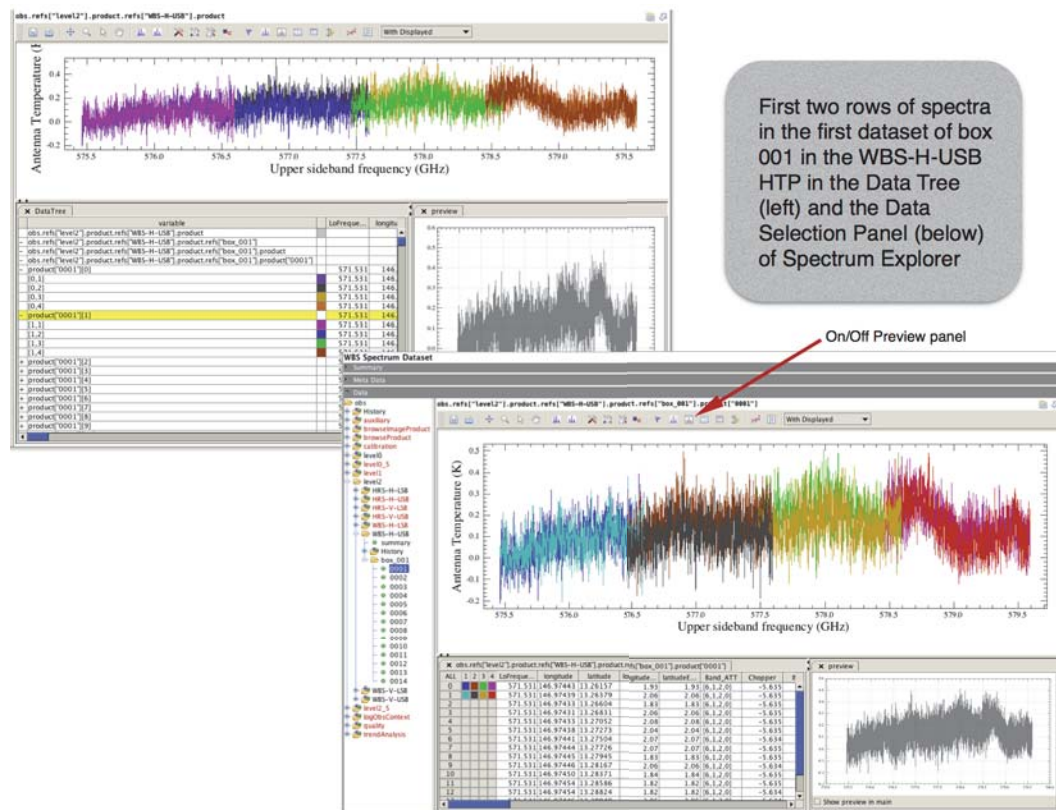
Using the Spectrum Explorer to inspect the HifiTimelineProduct (HTP) can be a powerful way to get an overview of your data, particularly for mapping observations. The HTP is opened in the Spectrum Explorer using a right click on the variable name (in the Variables pane or in the Observation Context tree), and selecting *SpectrumExplorer* from the *Open With* menu.

Nothing is plotted in the upper panel when opening an HTP in the Spectrum Explorer. The data are displayed in the *Data Tree* in the selection panel, which is initially collapsed. Click on the plus sign to the left of the variable name (which is the same as the reference to the data product) to expand the tree without plotting anything, or click on the box to the right of the variable name which will plot all the data but not expand the data tree. Be aware that plotting large amounts of data will take some time.

Once the contents of the Data Tree have been expanded, you will find a new row for each product in the HTP. The columns (from left to right) show:

- *First column:* a '+' for collapsed data containing multiple spectra, or a '-' for expanded datasets. Clicking on the symbol will expand or collapse that data tree.
- *Second column:* displays the variable name of the data, the default name given is that of the reference to the product. You can rename the variable by double clicking on the variable name, the renamed variable will be added to HIPE Variables pane. Double clicking on a variable name also causes a red cross to appear next to the variable name, clicking on the red cross allows you to remove the data (and all 'sub-data' belonging to it) from the Spectrum Explorer.
- *Third column:* displays the colour of the line (layer), if displayed. If collapsed data are displayed in the plot, the box will be grey coloured. Clicking on this box will display or hide all the spectra in this data set. On mouse-over, a displayed spectrum in the plot will be temporarily highlighted.
- *Remaining columns:* metadata in the data. As for the Data Selection panel, the columns can be reordered horizontally or sorted according to the metadata value (see the [Filtering and Sorting what is viewed](#) section in the 'Herschel Data Analysis Guide').

It can be helpful to widen the *variable* column in order to see the full reference to the product. Each product within the HTP can be expanded in the same way, producing a row for each spectrum contained in the product. This way of interacting with the data is quite different than in the Data Selection Panel used for spectra. To illustrate, the first two rows of spectra in the first dataset in box\_001 in the WBS-H-USB is shown in both the Data Selection Panel and the Data Tree in [Figure 6.4](#). The advantage of using the Data Tree is that all of your data is available in one tab, allowing you to easily pick and choose what to plot from all of the data in the HTP. Using the Spectrum Explorer and the Data Selection Panel requires you to plot each dataset in a box separately. The disadvantage is that the Data Tree is less intuitive to use and it is not possible to plot, say, all the spectra in one subband with one click.



**Figure 6.4. Opening the Spectrum Explorer on a HTP: comparing the Data Tree and Data Selection Panel**

Clicking anywhere in a row in the Data Tree will cause that row to be highlighted in yellow, and the spectra in the product to be displayed in the *Preview* pane to the right of the Data Tree. If you do not see a Preview pane then click on the 'Display the preview panel' icon (marked by a red arrow in

[Figure 6.4](#)) in the Spectrum Explorer button bar to activate it. Right clicking anywhere in the Data Tree will give you a menu allowing you to:

- select/deselect, this is only offered if you have selected spectra in the plot
- expand/collapse all of the products in the Data tree
- display/hide all the spectra in the plot in the panel above the data tree
- copy the contents of a cell

The mosaic function, or raster panel, of the Spectrum Explorer is a useful tool to inspect large datasets. It is opened by clicking on the solid grid icon in the Spectrum Explorer button bar, and will open a new tab in the plot panel of the Spectrum Explorer from where you can select three ways to display the data in your HTP:

- *Grid*: all of the spectra in the HTP are displayed in order from top to bottom-right as a series of postage stamps. On mouse-over, a spectrum is displayed in the preview panel. You can adjust the x and y ranges of the data are viewed over using the slide bars at the top of the panel. This is the default mode the raster panel opens in.
- *Raster*: the spectra are displayed according to their position, RA and dec values are given on the left and top axes, respectively. The x and y ranges of the data viewed can be adjusted as for the grid view. Spectra that are from close-by sky positions may be plotted so close together that they overlap but each plot moves to the top on mouse-over, and is displayed in the preview pane. You can also zoom in and out on the raster display using the mouse wheel or track pad equivalent.
- *Location*: crosses mark the positions of spectra in the dataset, with RA and dec given on the left and top axes. On mouse-over, the spectrum of each point is shown in the preview panel and you can zoom in and out on the display using the mouse wheel or track pad equivalent.

You can reset the display with the reset button at the top right on the panel and return to the original scale after zooming with a right-click.

Prior to HIPE 8 (and even for some older maps in HIPE 8), the gridding of spectral maps was not optimal with the result that some cubes were created with pixels sizes different than that appropriate for the way the observation was carried out. This could result in cubes with a different number of pixels (and potentially map rows and columns) than calculated by HSpot. Using the Spectrum Explorer mosaic function to inspect the positions of spectra in the map can be helpful in providing a quick comparison with the cube produced by the pipeline.



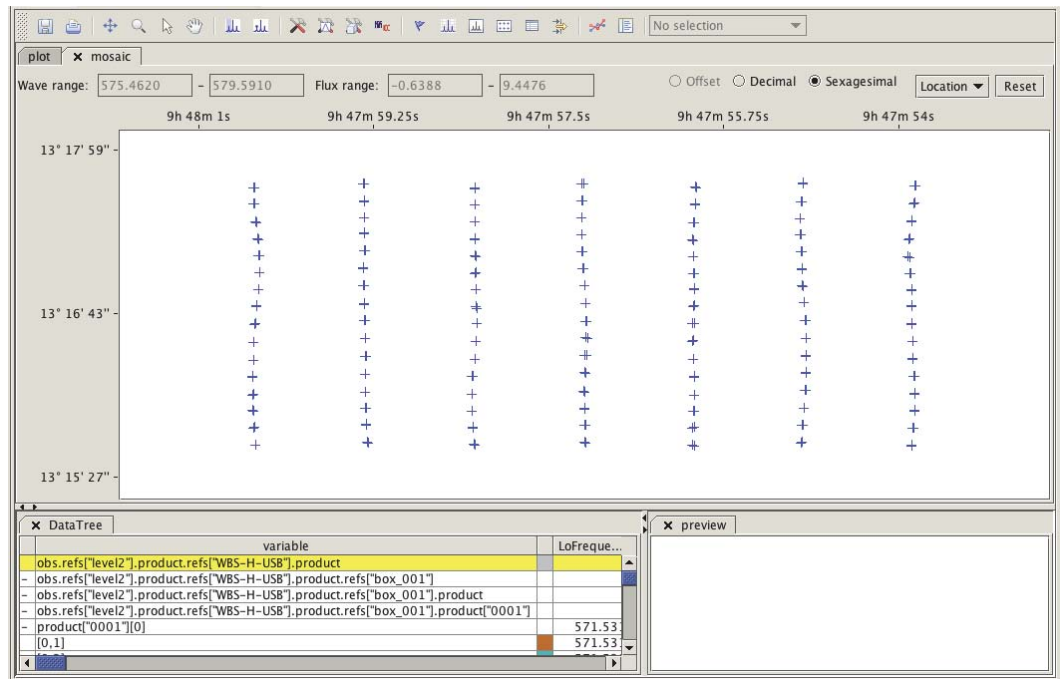


Figure 6.5. Using the location option of the Spectrum Explorer mosaic to see the positions of spectra in a map

## 6.3. Scripted plotting of spectral data with PlotXY

PlotXY() is the basic package to plot arrays of data points in the HCSS, and it can be used to plot HIFI spectra as well. It has a lot of options, making the plots highly configurable. Here is an example of plotting a HIFI spectrum:

- Get the spectrum (technically a HifiSpectrumDataset) from an ObservationContext. You can right click on the dataset in the ContextViewer and create a variable. Alternatively, you can adapt the following to be appropriate for the level, spectrometer and sideband of interest:

```
ds = obs.refs["level2"].product.refs["WBS-H-USB"].product.refs["box_001"].product["0001"]
```

- You need to extract the flux and frequency data separately in order to plot them against each other. The following example will extract the flux and frequency from the first subband (*Segment*) of the first spectrum (*PointSpectrum*) in a dataset:

```
freq = ds.getPointSpectrum(0).getSegment(1).getWave()
flux = ds.getPointSpectrum(0).getSegment(1).getFlux()
```

Note that numbering for the Segments begins at 1, while the numbering of the PointSpectra begins at 0.

- The simplest possible plot:

```
out=PlotXY(freq, flux)
```

- When plotting multiple spectrum datasets, say 'sd1' and 'sd2' in one figure:

```

# Get the wavelengths and fluxes to be plotted
freq1=sd1.getPointSpectrum(0).getSegment(1).getWave()
flux1=sd1.getPointSpectrum(0).getSegment(1).getFlux()
freq2=sd2.getPointSpectrum(0).getSegment(1).getWave()
flux2=sd2.getPointSpectrum(0).getSegment(1).getFlux()
#
# Create the plot variable
p=PlotXY()
# Define the layer variable
l1=[]
# Optional: remove any non-numbers (NaN's, Infinities etc.)
valid=flux1.where(IS_FINITE)
# Create layer for first plot
l=LayerXY(freq1[valid],flux1[valid])
# Append to layer variable
l1.append(l)
# Repeat the above for the 2nd plot to be overlaid
valid=flux2.where(IS_FINITE)
l=LayerXY(freq2[valid],flux2[valid])
l1.append(l)
# Define the plot layers that have just been created
p.layers=l1

```

- And this is how some common features of the plot are modified.

```

p.setYrange([0, 1.5])
p.setTitleText("This is an example plot")

```

## 6.4. Scripted plotting of spectral data with SpectrumPlot

All Herschel spectra types can be displayed with the `SpectrumPlot` package.

`SpectrumPlot` is built on `PlotXY` and so many of the features you would use in `PlotXY` you can also use for `SpectrumPlot`. The main distinction of `SpectrumPlot` is that it allows you to plot spectra directly, without needing to extract the `Wave` and `Flux` arrays separately before plotting.

Below are two simple examples.

```

# Plot using Spectrum Plot
#
# Get data
obs = getObservation("1342190183")
L2_wbs_h_u = obs.refs["level2"].product.refs["WBS-H-USB"].product.refs \
["box_001"].product["0001"]
L2_wbs_v_u = obs.refs["level2"].product.refs["WBS-V-USB"].product.refs \
["box_001"].product["0001"]
L2_hrs_h_u = obs.refs["level2"].product.refs["HRS-H-USB"].product.refs \
["box_001"].product["0001"]
#
#-----
# Example 1:
# Get the segments of one spectrum (L2_wbs_h_u) and plot all the same colour
#
spec1 = L2_wbs_h_u.getPointSpectrum(0).getSegment(1)
spec2 = L2_wbs_h_u.getPointSpectrum(0).getSegment(2)
spec3 = L2_wbs_h_u.getPointSpectrum(0).getSegment(3)
spec4 = L2_wbs_h_u.getPointSpectrum(0).getSegment(4)
#
# Plot
splot = SpectrumPlot()
splot.add(spec1)

```

```
splot.add(spec2)
splot.add(spec3)
splot.add(spec4)
#
# Remove the fourth spectrum
splot.remove(spec4)
#
# Set colours
# Note, layers are labelled with indices from 0 up (first layer has index 0)
splot.getLayer(0).setColor(java.awt.Color.RED)
splot.getLayer(1).setColor(java.awt.Color.RED)
splot.getLayer(2).setColor(java.awt.Color.RED)
#
# Limit the X range
splot.setXrange([575.5, 577.0])
#
# Clear all annotations, then set one and a title
splot.clearAnnotations()
splot.addAnnotation(Annotation(576.5,3, "My Annotation", \
color=java.awt.Color.BLUE, fontSize=14))
splot.setTitleText("Title")
#
# Save plot as png
splot.saveAsPNG("/Users/carolynmccoey/Desktop/splot.png")
#
#-----
# Example 2:
# Plot the line in all three spectra, using different line styles
#
# Note that Spectrum Plot treats each subband
splot = SpectrumPlot()
splot.add(L2_wbs_h_u) # Layers 0-3 (4 subbands). Line is in subband 3 (layer 2)
splot.add(L2_wbs_v_u) # Layers 4-7 (4 subbands). Line is in subband 3 (layer 6)
splot.add(L2_hrs_h_u) # Layer 8 (1 subband)
#
# Set line styles
splot.getLayer(2).setLine(0)
splot.getLayer(6).setLine(3)
splot.getLayer(8).setLine(2)
```

The first example will produce the following plot:

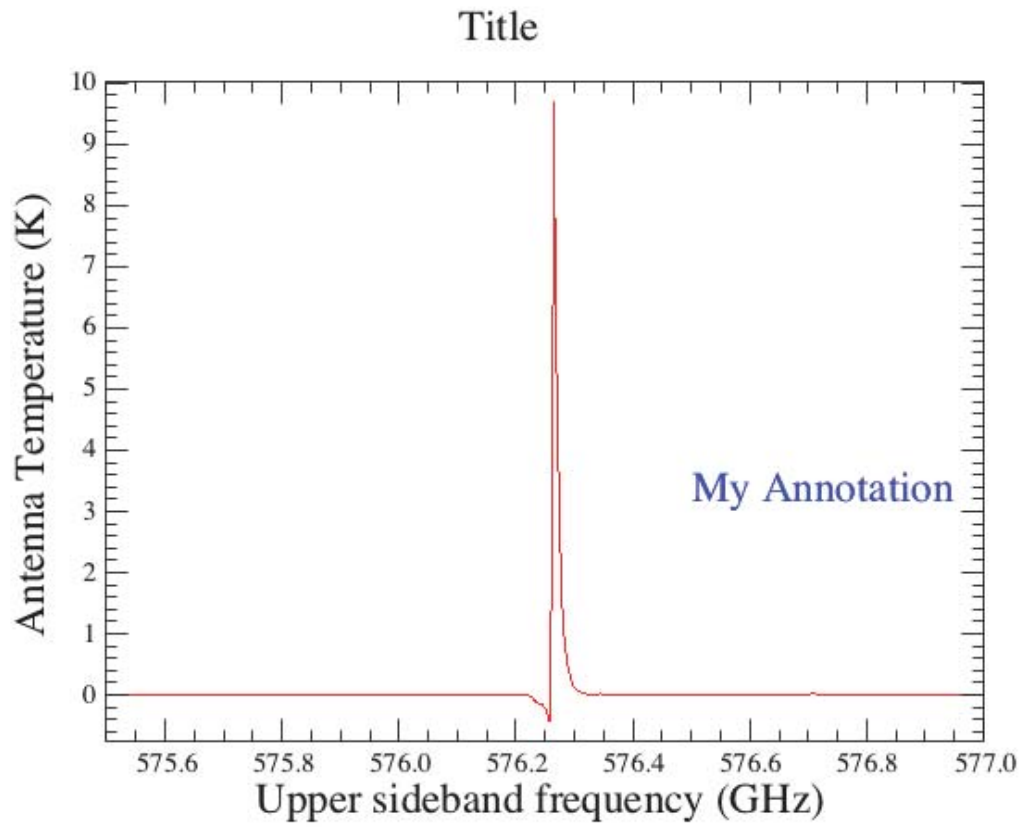


Figure 6.6. Plot produced with SpectrumPlot

# Chapter 7. Converting positions in data to offsets

Last updated: 28 April, 2015

## 7.1. Introduction

You may wish to inspect the position of your data relative to the requested position of the map centre, or in the case of Solar System Objects (SSOs) relative to the target position. The `DoOffset` task allows you to convert the values in the *longitude* and *latitude* columns in your datasets from absolute values to values relative to the requested positions (*raNominal* and *decNominal* in the Observation Context metadata). The task will also convert the relative values back to absolute positions again. You have the option to temporarily modify the longitude and latitude values or overwrite them.

## 7.2. Using the doOffset task

The `DoOffset` task will calculate positions relative to the values of the requested RA and dec, *raNominal* and *decNominal*, found in the metadata. Alternatively, you can pass values of RA and dec, in decimal degrees, to the task to be used to calculate the offset positions from.

In the case of a moving target (Solar-System object), the resulting offset coordinates are in a coordinate system co-moving with the target, centred on the nominal target position. For fixed targets the offset coordinates are centred on the nominal map centre. Therefore, this task allows you to inspect the offsets in all types of data from the requested position.

If you supply RA and dec values to the task it will write these values to the metadata of the resulting HTP as *ra\_centre* and *dec\_centre*. These metadata items will be used to calculate positions if you use the task again to convert the coordinates from offsets to absolute values, or vice versa.

The task works on HTPs and datasets, see the script examples below:

```
obs = getObservation(1342227194, useHsa=True)
htp = obs.getProduct("level2").getProduct("WBS-H-USB")
#
# Create a new htp2 with longitude and latitude values relative to offset
# coordinate,
# the original htp is not overwritten:
htp2=doOffset(htp=htp, overwrite=0, relative=1)
#
# Now convert the positions back to absolute coordinates:
htp3=doOffset(htp=htp2, overwrite=0, relative=0)
#
# To convert the positions (back) to relative values, and overwrite the positions in
# the original HTP:
doOffset(htp=htp3, overwrite=1, relative=1)
#
# If you now change the data back to relative positions, you will want to overwrite
# the original HTP again:
doOffset(htp=htp3, overwrite=1, relative=0)
#
# You can supply your own values of RA and dec, here 59.756165 and -71.1675896074
# from which to calculate the offset positions:
# Note that if ra_centre and dec_centre are present, they will be used instead:
htp2b=doOffset(htp=htp, overwrite=0, relative=1,ra=59.756165, dec=-71.1675896074)
#
#
# The task will also work with datasets, here we convert the positions in the first
# dataset in the HTP to relative values:
ds=doOffset(ds=htpv[1], overwrite=0, relative=1)
```

#

You can also use the GUI to operate the task. `DoOffset` will appear under *Applicable* in the *Tasks* View when you click on an HTP. To convert data to relative positions, check the *relative* box. To overwrite the original HTP with the new positions, check the *overwrite* box. You can supply an RA and dec, in decimal degrees, from which to calculate the offset positions in the text fields.

Viewing the converted HTP or dataset in the *location* view of the Spectrum Explorer will allow you to see the offsets of each position in RA and dec, [Figure 7.1](#) and [Figure 7.2](#) show the positions in `htp` (absolute) and `htp2` (relative), respectively.

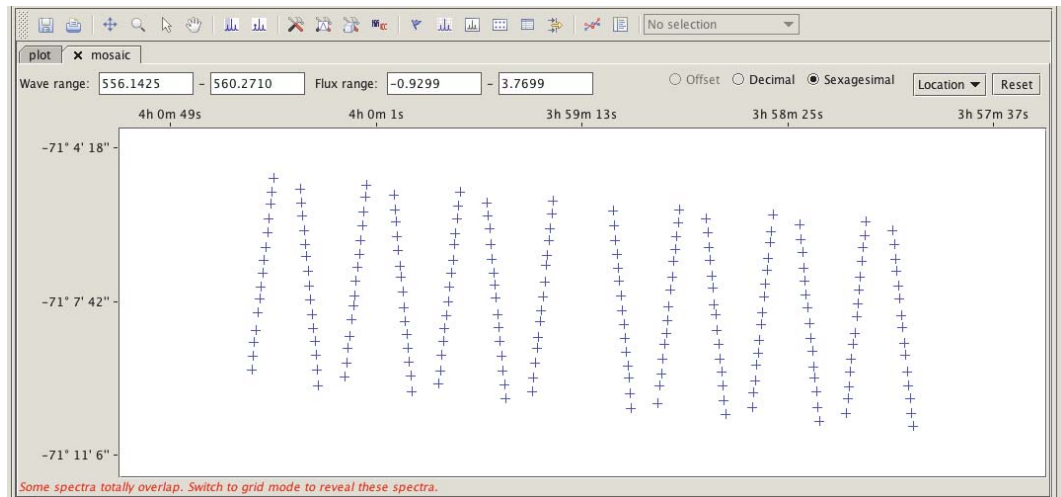


Figure 7.1. Absolute positions

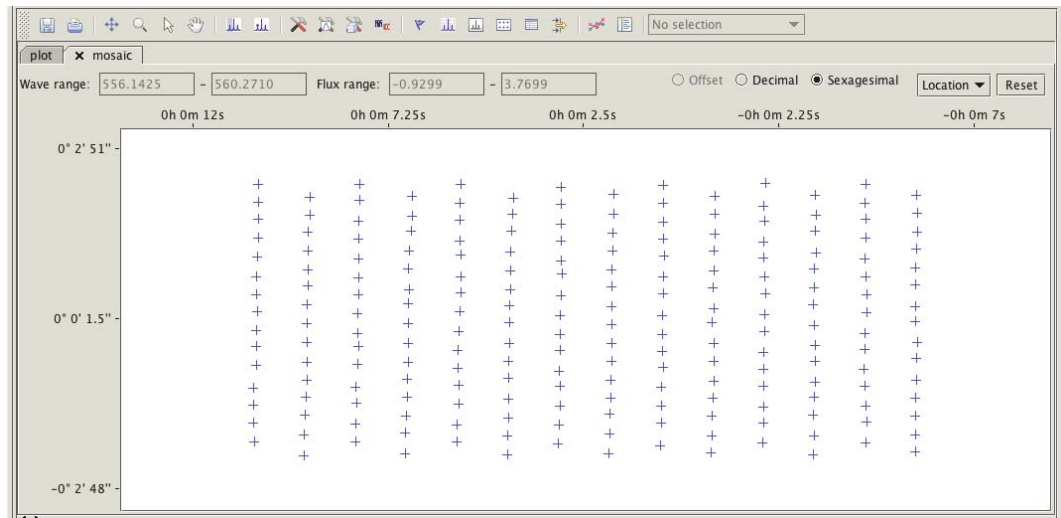


Figure 7.2. Relative positions, in a coordinate system co-moving with the Solar-System target

# Chapter 8. Understanding and using HIFI beam information in your data

Last updated: 6 February, 2015

## 8.1. Beam Metadata

Information about the HIFI beam and related information can be found in the `SpectrumDataset` metadata. The most useful for understanding what beam parameters and efficiencies have been applied to your data, as well as the efficiencies used in converting temperature scales are tabulated below.

Metadatum	Description	Introduced/modified by	Comment	FITs keyword
apertureEfficiency	Telescope aperture efficiency	doFluxHot - Cold		ETAA
aGeom	Telescope geometric aperture area	doFluxHot - Cold		AGEOM
forwardEfficiency	Telescope forward efficiency (recommended calibration value)	doAntennaTemp/doMainBeamTemp	Only generated if user supplies values for efficiency	ETALCAL
forwardEff	Telescope forward efficiency (applied value)	doFluxHot - Cold		ETAL
mainBeamEfficiency	Telescope main-beam efficiency (recommended calibration value)	doAntennaTemp/doMainBeamTemp	Only generated if user supplies values for efficiency	ETAMBCAL
beamEff	Telescope main-beam efficiency (applied value)	doFluxHot - Cold		ETAMB
hpbw	Azimuthally-averaged half-power beam width	doFluxHot - Cold		HPBW
temperatureScale	Temperature scale in use	doAntennaTemp/doMainBeamTemp	values T_A* or T_mb	
temperatureScale-Origin		convertK2Jy	Used by <code>convertK2Jy</code> in order to be able to make the conversion back to the original temperature scale. Values T_A*, T_mb or T_Aprime	

## 8.2. Tools to obtain and use the HIFI beam model

### `getHifiBeam`

The HIFI 2D beams are stored in the HIFI Calibration tree as tables for two frequencies per mixer. As detailed in the [beam release note](#), they need to be scaled to the appropriate frequency and rotated by the telescope roll angle at the time of the observation. Two auxiliary functions, `getHifiBeam` and `getHifiBeamAveraged` are provided in `herschel.hifi.pipeline.util`.

`getHifiBeam` generates a `SimpleImage` containing the 2D beam profile scaled to a given frequency and rotated to a given position angle. There are two equivalent ways of calling `getHifiBeam`:

- `getHifiBeam(band, backend, freq, posAngle, cal)`

where the user manually provides `band` (e.g., `1b`), `backend` (e.g., `WBS-V`), frequency in GHz, telescope roll angle (`posAngle`, which can be found in the metadata) in degrees, and a reference to a calibration tree containing the beam models (`cal`); or,

- `getHifiBeam(ds, cal, useLoFreq=True)`

where `band`, `backend`, frequency, and position angle are retrieved from the passed dataset `ds`. If the parameter `useLoFreq` is set to `True`, the beam is calculated for the LO frequency, otherwise the average frequency of the spectrum is used. A reference to a calibration tree containing the beam models is still required and can be obtained from an `observationContext` (`obs`) as follows:

```
cal = obs.getCalibration()
```

You must use an observation that has been processed at the HSA with HIPE 13 (and onwards) in order to get the beam models.

The resulting `SimpleImage` can be used seamlessly using tasks such as `imageMultiply`.

### `getHifiBeamAveraged`

`getHifiBeamAveraged` is analogous to `getHifiBeam` but deals with the azimuthally averaged beam instead of the full 2D beam. It generates a `TableDataset` containing the azimuthally averaged beam and the encircled-energy fraction (EEF) as a function of radial distance, scaled to the frequency of interest. There are two ways to call `getHifiBeamAveraged`, identical to those for `getHifiBeam`, except that `posAngle` is not required (nor accepted by the function; the symmetrised beam is independent of roll angle).

### Beam models from FITS files

The 2D beam models contained in the calibration tree are spatially cropped to an extent of roughly 10 HPBW across (i.e., from roughly -5 HPBW out to ~5 HPBW in each linear dimension). This should be sufficient for most scientific purposes. If you require full-size beam models, you are referred to the FITS files available from the [ESA Ancillary Data Product page](#).

This approach will also come in handy if you do not have access to a recent calibration tree, or if you do not wish to use it for whatever reason.

Like their counterparts contained in the calibration tree, the beam models contained in the FITS files need to be scaled and rotated appropriately. A script `getHifiBeam.py` performing the task of both the above-mentioned `getHifiBeam` and `getHifiBeamAveraged` can be found in the HIFI Useful Scripts menu of the HIPE Scripts menu; it can also be downloaded from the [HIFI Instrument and Calibration](#) web page ([direct link](#)). This script returns both a `SimpleImage` and a `TableDataset`.



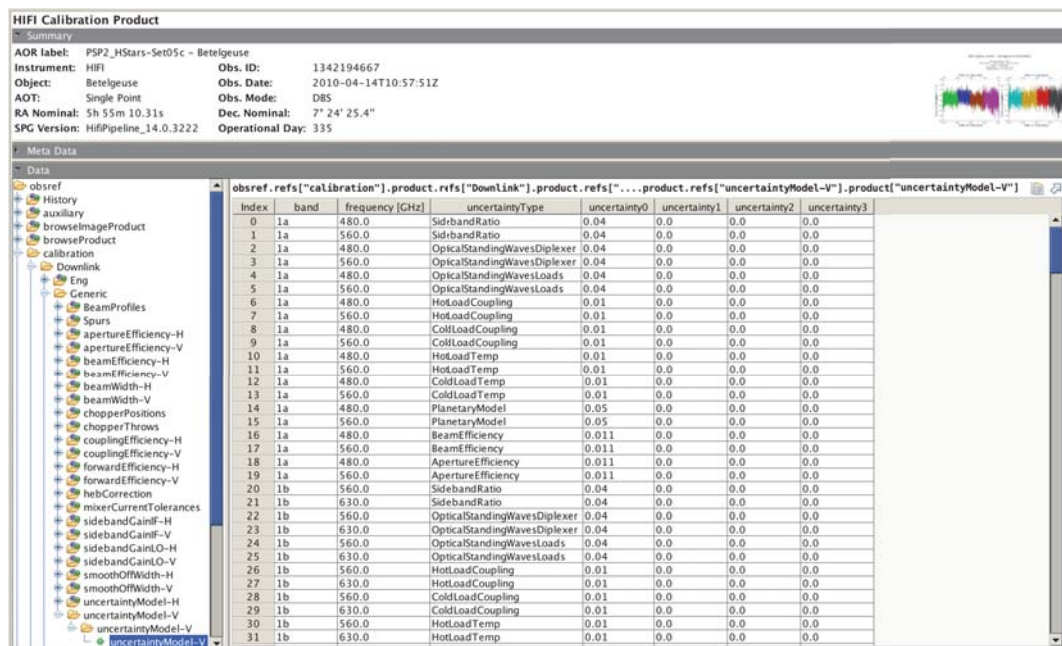
# Chapter 9. Understanding the uncertainty table information in your data

Last updated: 4 November, 2015

## 9.1. Uncertainty model

The HIFI flux calibration uncertainty is broken-down between the various components entering the general calibration equation (see [Ossenkopf 2003, ALMA memo no. 442.1](#)). For each component, the uncertainty is given as coefficients of a polynomial describing the possible dependency on the Intermediate Frequency (IF). In almost all cases, however, the uncertainty is flat over the IF and only the first coefficient is non-zero.

Uncertainties are LO-frequency dependent and given in percentages. Note that some uncertainties apply directly to the Level 2 or Level 2.5 products calibrated in Ta\*, while some others will only apply to data converted into a certain intensity scale (see [Chapter 18](#)) - see the following section for more details.



Index	band	frequency [GHz]	uncertaintyType	uncertainty0	uncertainty1	uncertainty2	uncertainty3
0	1a	480.0	Sid-bandRatio	0.04	0.0	0.0	0.0
1	1a	560.0	Sid-bandRatio	0.04	0.0	0.0	0.0
2	1a	480.0	OpticalStandingWavesDiplexer	0.04	0.0	0.0	0.0
3	1a	560.0	OpticalStandingWavesDiplexer	0.04	0.0	0.0	0.0
4	1a	480.0	OpticalStandingWavesLoads	0.04	0.0	0.0	0.0
5	1a	560.0	OpticalStandingWavesLoads	0.04	0.0	0.0	0.0
6	1a	480.0	HotLoadCoupling	0.01	0.0	0.0	0.0
7	1a	560.0	HotLoadCoupling	0.01	0.0	0.0	0.0
8	1a	480.0	ColdLoadCoupling	0.01	0.0	0.0	0.0
9	1a	560.0	ColdLoadCoupling	0.01	0.0	0.0	0.0
10	1a	480.0	HotLoadTemp	0.01	0.0	0.0	0.0
11	1a	560.0	HotLoadTemp	0.01	0.0	0.0	0.0
12	1a	480.0	ColdLoadTemp	0.01	0.0	0.0	0.0
13	1a	560.0	ColdLoadTemp	0.01	0.0	0.0	0.0
14	1a	480.0	PlanetaryModel	0.05	0.0	0.0	0.0
15	1a	560.0	PlanetaryModel	0.05	0.0	0.0	0.0
16	1a	480.0	BeamEfficiency	0.011	0.0	0.0	0.0
17	1a	560.0	BeamEfficiency	0.011	0.0	0.0	0.0
18	1a	480.0	ApertureEfficiency	0.011	0.0	0.0	0.0
19	1a	560.0	ApertureEfficiency	0.011	0.0	0.0	0.0
20	1b	560.0	SidebandRatio	0.04	0.0	0.0	0.0
21	1b	630.0	SidebandRatio	0.04	0.0	0.0	0.0
22	1b	560.0	OpticalStandingWavesDiplexer	0.04	0.0	0.0	0.0
23	1b	630.0	OpticalStandingWavesDiplexer	0.04	0.0	0.0	0.0
24	1b	560.0	OpticalStandingWavesLoads	0.04	0.0	0.0	0.0
25	1b	630.0	OpticalStandingWavesLoads	0.04	0.0	0.0	0.0
26	1b	560.0	HotLoadCoupling	0.01	0.0	0.0	0.0
27	1b	630.0	HotLoadCoupling	0.01	0.0	0.0	0.0
28	1b	560.0	ColdLoadCoupling	0.01	0.0	0.0	0.0
29	1b	630.0	ColdLoadCoupling	0.01	0.0	0.0	0.0
30	1b	560.0	HotLoadTemp	0.01	0.0	0.0	0.0
31	1b	630.0	HotLoadTemp	0.01	0.0	0.0	0.0

Figure 9.1. Table containing the uncertainty model (values are in percentages) (for the V polarisation in this example)

## 9.2. Flux calibration uncertainty budget

The contribution from each of the uncertainty components considered in the *uncertaintyModel* tables is propagated in the final uncertainty budget table using LO-frequency interpolation. Again, an IF-dependent description is considered by means of propagating the polynomial coefficients present in the input model. Additionally, the uncertainties are also directly computed at representative locations of the IF, namely:

- for the WBS, the uncertainty is computed at the IF centre (*uncertaintyIfMid*), and at the two IF edges (*uncertaintyIfLow* and *uncertaintyIfHigh*). For each case, it corresponds to the mean uncertainty in a window of 100 MHz wide.

- for the HRS, the uncertainty is computed in the centre of the IF for each individual HRS subband (*uncertaintyIfMid\_1*, *uncertaintyIfMid\_2*, etc)

The uncertainty table separates contribution from components assumed to be uncorrelated, from those that should be treated either in a systematical manner, or that will apply to a particular flux scale conversion.

- the uncertainties related to the sideband ratio, the optical standing waves, and the internal load properties are assumed to be independent and are added in quadrature into the uncertainty component called *SumUncorrelatedUncertainty*. Note that those are the uncertainties applied to data calibrated in the Ta\* scale.
- the planetary model error is a systematic error and is assumed to be constant at all HIFI frequencies
- additional uncertainties related to the extraction of the respective beam and aperture efficiencies (on top of the planetary model used in their derivation) are provided. They should not be considered together, instead they will apply depending on whether a conversion to Tmb scale or to flux scale is considered (see [Chapter 18](#)).

The uncertainty product is computed for each spectrometer, polarisation, and sideband, as well as for each LO frequency tuned in the observation. For spectral maps in particular, only one uncertainty budget table is provided per spectrometer, polarisation, and sideband because no distinction is made between the respective pixels of the map.

**Hifi Uncertainty Table**

Summary

AOR Label: PSP2\_HStars-Set05c - Betelgeuse  
 Instrument: HIFI Obs. ID: 1342194667  
 Object: Betelgeuse Obs. Date: 2010-04-14T10:57:11Z  
 AOT: Single Point Obs. Mode: DBS  
 RA Nominal: 5h 55m 10.31s Dec. Nominal: 7° 24' 25.4"  
 SPG Version: HifiPipeline\_14.0.3222 Operational Day: 335

Meta Data

Data

Index	uncertaintyType	uncertaintyLow	uncertaintyIfMid	uncertaintyIfHigh	uncertainty0	uncertainty1	uncertainty2	uncertainty3
0	HotLoadTemp	0.013	0.013	0.013	0.013283226654618407	0.0	0.0	0.0
1	ColdLoadTemp	0.0	0.0	0.0	4.257339823986842E-4	0.0	0.0	0.0
2	HotLoadCoupling	0.02	0.02	0.02	0.02	0.0	0.0	0.0
3	ColdLoadCoupling	0.02	0.02	0.02	0.02	0.0	0.0	0.0
4	SidebandRatio	0.04	0.04	0.04	0.04034345060962079	0.0	0.0	0.0
5	OpticalStandingWavesLoads	0.039	0.039	0.039	0.03895597675075099	0.0	0.0	0.0
6	OpticalStandingWavesDiplexer	0.068	0.068	0.068	0.06840875462574247	0.0	0.0	0.0
7	SumUncorrelatedUncertainty	0.093	0.093	0.093	0.094	0.0	0.0	0.0
8	BeamEfficiency	0.01	0.01	0.01	0.01	0.0	0.0	0.0
9	ApertureEfficiency	0.011	0.011	0.011	0.011	0.0	0.0	0.0
10	PlanetaryModel	0.05	0.05	0.05	0.05	0.0	0.0	0.0

Figure 9.2. Table containing the uncertainty budget (values are in percentages) (for the V polarisation in this example)

# Chapter 10. Flags in HIFI data

Last updated: 29 February, 2016

## 10.1. Introduction to flags

Flags (also called masks) are identifiers of specific issues with the data, such as saturated pixels or a possible spur, that can affect the quality of the final product. Flags are applied by the pipeline and used to identify potentially problematic data, and to make a caution during its processing.

A data flag has a defined name and a value, which specifies the nature of the flag. These flags are divided into two categories depending on whether they apply to an individual channel (pixel), or to a complete Dataframe. They are called *channel flags* and *column rowflags*, respectively. There are also *Quality Flags* which are found in the Quality Product in the ObservationContext and are used to provide you with means to make a quick assessment of the quality of your data. They are described in [Section 10.4](#).

## 10.2. Channel flags

Channel (or pixel) flags apply to individual pixels and are added as a column in the HTP. Their names are also added to the metadata of a dataset during processing, and this is used for the history of the pipeline. It also means that you can tell that, e.g., the WBS pipeline has been applied if you see things like "isMasked" and "checkZero" in the metadata.

For each pixel, there are 32 flags which can be set. The definition of the currently used mask bits and values in HIFI data is given below. For bit  $n$ , the value is computed according to  $value=2^n$ . For scripting, it is recommended that instead of using the bit values, you avail yourself of the software names defined in the HifiMask class. The section on [Chapter 11](#) has more details.

Flag Name	Bit	Software Name	Description
Bad pixel	0	HifiMask.BAD_PIXEL	Indicates a failed pixel on a WBS CCD. Not used in HRS.
Saturated pixel	1	HifiMask.SATURATED	If this bit is set, the sample was saturated.
Not observed	2	HifiMask.NOT_OBSERVED	If this bit is set, the sample is not observed. See Note 1.
Not Calibrated	3	HifiMask.NOT_CALIBRATED	If this bit is set, the sample is not calibrated. It is set on pixels that also have the SATURATION bit set. See Note 2.
Not used	4		Not used
Glitch detected	5	HifiMask.GLITCHED	Indicates a WBS data point affected by a cosmic ray.
Dark pixel	6	HifiMask.DARK_PIXEL	If this bit is set, the sample is used to measure the dark.
Spur candidate	7	HifiMask.SPUR_CANDIDATE	If this bit is set, the sample is a candidate to be a spur. It is a 'candidate' since not all things flagged by the spurfinder are necessarily spurs.
Spur warning	8	HifiMask.SPUR_WARNING	This bit is set to indicate that spurs have been observed at

Flag Name	Bit	Software Name	Description
			similar LO tunings for those channels, although it does not necessarily imply that a spur is actually present in this particular observation. This flag is essentially a warning, and therefore is not honoured by the <code>fitHifiFringe</code> , <code>fitBaseline</code> , <code>deconvolution</code> or <code>gridding</code> algorithms.
line	28	HifiMask.LINE	User set flag to denote a line.
Bright line	29	HifiMask.BRIGHT_LINE	User set flag to denote a bright line.
Ignore data	30	HifiMask.IGNORE_DATA	User set flag to ignore data.

NOTE 1 : The `NOT_OBSERVED` bit is typically used to represent data computed via interpolation or extrapolation. At Level 2, this is common at the end-points of a subband where the resampler has to extrapolate beyond the observed frequency range by under 1 pixel in order to ensure that the data sets are all the same size. It is also used at Level 0 and 0.5 to represent data that really are not observed. This occurs when the IF of HIFI covers a region narrower than the 8192 pixels that the CCD does. By design, the IF is well matched to the CCD, hence the number of unobserved pixels is very small. They are culled from the spectra in Level 1 and beyond.

NOTE 2 : The `NOT_CALIBRATED` bit is set for saturations in addition to the `SATURATED` bit. Saturated regions (such as strong spurs) can migrate in IF over time. This leads to a possible scenario where, when averaging data, a 'good' region from one spectra overlaps with a 'bad' region from another. The `NOT_CALIBRATED` flag is logically 'or-ed' between the good and the bad data, which means that the final output will properly reflect that the data in this overlap region should not be trusted.

NOTE 3 : Bits 28-30 are special because they are set by you and not the pipeline. The 'line' and 'bright line' will be used by `fitBaseline`, `fitHifiFringe`, and the `doDeconvolution` tools. The ignore data flag allows you to mask out bad spectral regions which can then be ignored by the pipeline.

NOTE 4 : It is possible to replace a flagged value with a NaN (Not a Number) or another value (e.g. fitted with an algorithm) using the `removePixel` task. In addition, if the mask is homogeneous for all scans within a dataset, it is possible to remove the channel(s) from the HTP.

```
from herschel.hifi.pipeline.product import HifiMask
from herschel.hifi.pipeline.util.tools import RemoveFlaggedPixelsTask

# replace bad pixels with NaN (default behaviour)
removePixel(htp=htp, mask=HifiMask.BAD_PIXEL)

# interpolate over bad and saturated pixels
removePixel(htp=htp,
            mask=[HifiMask.BAD_PIXEL, HifiMask.SATURATED], mode='interpolate')

# remove flagged channels (
removePixel = RemoveFlaggedPixelsTask()
```

## 10.3. Column rowflags

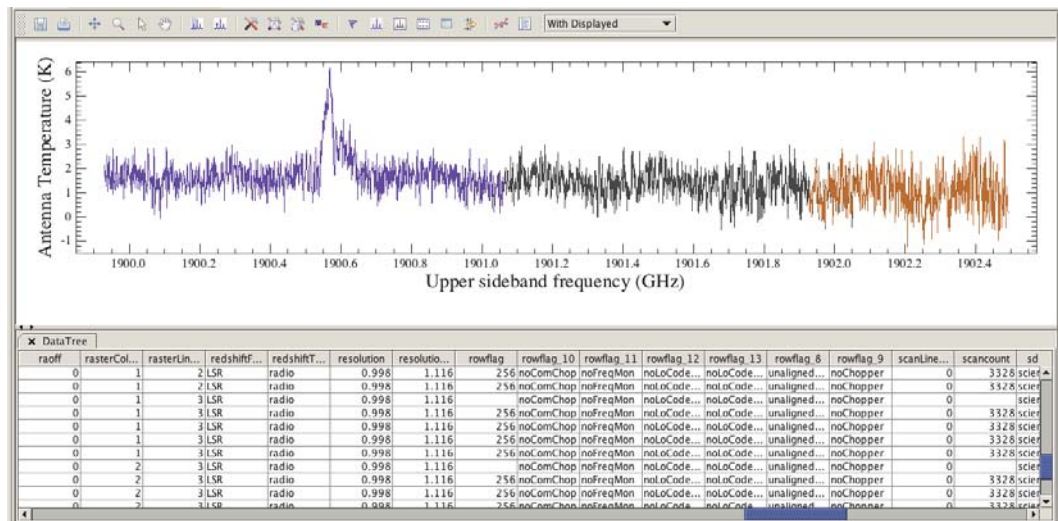
Column rowflags (the "rowflag" column in the HIFI spectrum `TableDataset`) apply to the complete `DataFrames` (DF) or rows in a `HifiSpectrumDataset` (HSD).

For bit  $n$ , the value is computed according to  $value=2^n$ . The first 5 bits are about the packets from which the `DataFrame` (DF) is reconstructed, and are unlikely to ever occur.

Below is a table showing the current names and values of HIFI rowflags. As for the channel flags, there is a class defined to make scripting with rowflags easier.

Flag Name	Bit	Software Name	Description
PacketOrder	0	RowMask.PACKET_ORDER	Error in the packet order while constructing the DataFrame.
PacketLength	1	RowMask.PACKET_LENGTH	Error in the packet length while constructing the DataFrame.
TooMuchData	2	RowMask.TOO_MUCH_DATA	More data than can be fit in a DataFrame.
FirstPacket	3	RowMask.FIRST_PACKET	Error in the start packet while constructing the DataFrame.
NoBlocks	4	RowMask.NO_BLOCK	No block information present while constructing the DataFrame.
Spare	5	Not defined	
Spare	6	Not defined	
Spare	7	Not defined	
UnalignedHK	8	RowMask.UNALIGNED_HK	HK could not be aligned with DataFrames. When the columns "df_transfer" and "hk_transfer" in the TableDataset are different, bit 8 is set.
noChopper	9	RowMask.NO_CHOPPER	No valid Chopper information. Set when the flagbit is zero in the DFs, extracted from the HK packets if possible.
noComChop	10	RowMask.NO_COM_CHOP	No valid Commanded Chopper information. Set when the flagbit is zero in the DFs, extracted from the HK packets if possible.
noFreqMon	11	RowMask.NO_FREQ_MON	No valid Frequency Monitor information. Set when the flagbit is zero in the DFs, extracted from the HK packets if possible.
noLoCodeOffset	12	RowMask.NO_LO_OFF	No valid LO code offset information. Set when the flagbit is zero in the DFs, extracted from the HK packets if possible.
noLoCodeMain	13	RowMask.NO_LO_MAIN	No valid LO code main information. Set when the flagbit is zero in the DFs, extracted from the HK packets if possible.
MixerCurrentDeviation (ref)	14	RowMask.MCD_REF	Difference in mixer currents exceeds tolerance when applying DoRefSubtract.
MixerCurrentDeviation (off)	15	RowMask.MCD_OFF	Difference in mixer currents exceeds tolerance when applying DoOffSubtract.

Flag Name	Bit	Software Name	Description
MixerCurrentDeviation (load)	16	RowMask.MCD_HOT	Difference in mixer currents exceeds tolerance when applying DoFluxHotCold or MkFluxHotCold.
NoHotColdCalibration	17	RowMask.NO_HOT_COLD	Division by the bandpass has not been carried through.
SuspectLO	18	RowMask.SUSPECT_LO	LO Frequency is listed in the Bad Frequency Table. Data is not necessarily corrupted.
Spare	19	Not defined	
IgnoreData	20	RowMask.IGNORE_DATA	User has the option to set this flag. Some tools (e.g. doDeconvolution) will honour it.
BadData	22	RowMask.BAD_DATA	Corrupted data in this data-frame. Data will be ignored when building the Level 2 products.
BbidCorrection	28	RowMask.BBID_CORR	Correction of BBID. No longer relevant. (It was during ground-based testing, but the onboard software has been corrected since.)



Example of a HIFI spectrum TableDataset, which contains the "rowflag" column with a value of 256 ( $2^8 = \text{UNALIGNED\_HK}$ ), and a few other rowflag columns.

Figure 10.1. Row flags in a HIFI spectrum

## 10.4. Quality Flags

Last updated: 3 November, 2015

Quality flags can be raised during standard processing of HIFI data from every processing stage of the pipeline, from the initial creation of the *HifiTimelineProduct* (Level 0), through to the final product of Level 2 processing. Quality flags indicating potential issues with the data are collected in the *Quality Report*. Thus the quality report is, by definition, a list of issues identified as have gone wrong, and an empty quality report indicates that there is no potential problem with the observation. Note that

quality flags are raised in a conservative fashion and the presence of quality flags in the quality report does not mean that your data is inadequate for your purposes. A quality report is found from the ObservationContext:

```
report1=obs.refs["quality"].product
print report1
```

The quality report can also be found via the HSA Science Archive, and via the QualityContext Explorer under the "quality" product. To check possible problems with the data, check the Quality Flag Report, the JQLogProductPanel with the flag "level SEVERE", and the possible comments.

The screenshot displays the Quality Context Explorer interface. The main window shows the state of the 'quality' product as 'PASSED' with no actions. A table of quality flags is visible, listing various issues and their values. A comment from user 'herdpops' dated 12:25 PM, Mar 7, 2012, states that about 80% of the H polarization data are affected by pumping issues. The console window at the bottom shows the command used to generate the report: `report1=obs.refs["quality"].product`.

Concept	Value
Event report	
FPU Check:The level of mixer current is Out Of Limit.	
Maximum number of saturated pixel detected in a spectrum	465
Out of limit in the number of bad pixels detected	419
Percentage of Dataframes which have unaligned HK	27.0
Runtime errors found in the telemetry	21
Spur lines detected in the cold spectra	

Figure 10.2. Example of a Quality flag Report

Relevant and pertinent data quality information is also visible in the *Quality Control Summary Report*. The summary report can be accessible through the *quality summary* field of the QualityContext via the Quality Context Explorer. The type of flags displayed here are so-called public flags.

The quality flags are organized into categories from class 1 to 3 by how severe the impact they may have on the science. In general, a lower quality category indicates a more severe problem:

- **Class 1** flags: indicate that the data is partially or totally unusable for science.
- **Class 2** flags: refer to data that is usable for science, but that in some cases could still be affected by residual instrument artefacts.
- **Class 3** flags: refer to data that is usable for science with no particular further action.

Below is a list of the current available quality flags for the HIFI pipeline for every class. The format below gives flag name, consequences for science data, and action to follow.

### Class 1 Flags

Quality Flags	Consequences for science data	Action
All COMBs have failed to be fitted	The frequency calibration is inaccurate	Do not use data
At least one Science data has been affected from failing frequency calibration due to a COMB fit failure	Data is probably not properly frequency calibrated	Do not use data
Bbtype not known	The corresponding data may not be properly processed	Do not use data
Data measured from hot and cold loads not sufficient for hot/cold calibration	Part of or all the data could not be properly calibrated due to missing information, and so the quality of the end product can be degraded and/or cannot be trusted.	Do not use affected data
Failure in HifiSpectrumDataset construction	Data could not be properly processed and should probably not be used	Do not use the data
Fast Quantization Distortion Correction processed. Not optimal	HRS data have probably not been properly processed	Do not use the data
FPU Check: Cold load temperature is Out Of Limit	Range: [4, 20 K]. Cold load temperature out of specification	Do not use the data
FPU Check: Hot load temperature is Out Of Limit	Range [90, 110 K]. Hot load temperature out of specification	Do not use data
FPU Check: Level 0 Temperature is Out Of Limit	Range [1.5, 2.5 K]. Serious problem with the thermal environment or with the readout	Do not use data
Hot/cold calibration not successful	Data may not have been properly calibrated	Do not use data
Inacceptable maximum drift in the frequency grid detected	The frequency scale is probably wrong	Do not use data
Intensity calibration not or not for all spectra carried through	Part of the data could not be properly calibrated due to missing information, and so the quality of the end product can be degraded and/or cannot be trusted.	Do not use data
LOU has been potentially deactivated for one active band	Data are likely to be useless	Do not use data
Max number of command acceptance or execution failures found in the telemetry	Indicative of a failure during the execution of the observations. The integrity of the data cannot be guaranteed.	Use only unaffected data or data with acceptable quality
More ON- than OFF-datasets found in the data - not all ON-datasets could be processed with OFF-dataset(s)	Part of the data could not be properly calibrated due to missing information, and so the quality of the end product can	Check if part of the data is usable. If applies, correct baseline.



Quality Flags	Consequences for science data	Action
	be degraded and/or cannot be trusted.	
No Power Correction could be processed	HRS data have probably not been properly processed	Do not use HRS data
No Quantization Distortion Correction could be processed	HRS data have probably not been properly processed	Do not use HRS data
Observing mode not recognised - consult the pipeline configuration xml file	The concerned HRS-H data are likely to be lost due to the data corruption	Do not use data
One or more ASICs configuration problems in HRS-H	Data could not be properly processed and should probably not be used	Do not use the affected HRS data
One or more ASICs configuration problems in HRS-V	The concerned HRS-V data are likely to be lost due to the data corruption	Do not use the affected HRS data
Pattern observed for the LoFrequency not as expected in all datasets	Possible problem with LO tuning	Do not use data
The LOU was disabled during this observation, science data are likely unusable	Data are likely to be useless	Do not use data

### Class 2 Flags

Quality Flags	Consequences for science data	Action
Erroneous scan counts in calibration data-frames	Some COMB may not be processable, leading to potential COMB failure	None, unless additional flags such as the COMB or ZERO ones are raised. In that case the frequency calibration can be affected.
Failure in LO tuning status	Possible LO tuning problem, however for the vast majority it is due an artifact of the slow readout in FastChop mode.	Probably none, however double check integrity of spectra
Failure to associate pointing values to the spectra	The science data will not have proper attitude information	Data are usable but their astrometry cannot be trusted
FPU Check: IF Amplifier values are Out Of Limit	Range: [-1.5 V, +0.5 V]. The baseline quality could be degraded	If applies, clean baseline artefacts
FPU Check: Mixer current is Out Of Limit	Range: [ $I_{leak} + 5\mu A$ , $2 \times nom\_value$ ] for SIS, [ $30\mu A$ , $55\mu A$ ] for HEB. Some of the data could be noisier than the nominal performance.	Use with caution if combined with other data
FPU Check: The variance of the mixer current is out of limit	The baseline quality could be degraded	If applies, clean baseline artefacts
FPU Check: Mixer Magnet Current is Out Of Limit	Range: [ $nom\_value \times 0.96$ , $nom\_value \times 1.04$ ]. The baseline quality could be degraded	If applies, clean baseline artefacts
Less data found than expected	Although the impact has been mitigated for most affected cas-	Check noise and baseline. If applies, correct baseline

Quality Flags	Consequences for science data	Action
	es, the resulting data noise and baseline quality could be degraded.	
LO multiplier current deviates from zero, which is indicating of possible LOU impurity	Data may be affected by spurs or, more rarely, purity issues. Range: It depends of the band and frequency. See table below.	Mask artifacts where applies and pay particular attention to unexpected line location

**louCurrentMin**

Band	Frequency [GHz]	louCurrentMin[mA]	louCurrentMin[mA]
1a	540.0	-0.01	1.7976
1a	554.0	-0.01	1.7976
3b	937.0	-1.7976	-0.0060
3b	955.0	-1.7976	-0.0060
7a	1710.0	-1.7976	-0.0060
7a	1720.0	-1.7976	-0.0060
7a	1750.0	-1.7976	-0.0060
7a	1764.0	-1.7976	-0.0060
7b	1719.0	-1.7976	-0.0060
7b	1912.5	-1.7976	-0.0060

Quality Flags	Consequences for science data	Action
Max number of channels marked as BAD due repeated saturations	Science data are potentially saturated	Check for saturation in the data and flag if needed
Max number of TM Runtime error found	IF the HI_runtime_err in the level0 quality is different from LOTUNE_NOBRCKT, data could be missing but this will show in other higher severity level flags.	Check the HI_runtime_err in the level0 quality
Maximum number of saturated pixel detected in a single spectrum	Science data are potentially saturated	Check for saturation in the data and flag if needed
More data found than expected	Although the impact has been mitigated for most affected cases, the resulting data noise and baseline quality could be degraded.	Check noise and baseline. If applies, correct baseline.
No off baseline could be calculated	This is expected in NoRef modes. In others, indicates an anomaly in the data processing.	If applies, correct baseline
No off baseline subtraction carried through since no off baseline data available	The noise and/or the baseline quality may not be as intended	Check noise and baseline. If applies, correct baseline
No or not sufficient velocity information available	The frequency scale is probably still in the observatory frame, not in the LSR	Frequency calibration needs to be redone off-line

Quality Flags	Consequences for science data	Action
Not all phase checks could be carried through, or completed	Amount of Data is different than the expected observation. It could lead to degraded baseline quality	If applies, correct baseline
ON/OFF datasets not in expected sequence (...-ON-OFF-ON-OFF-... or ...-ON-OFF-OFF-ON-ON-....	The noise and/or the baseline quality can be degraded	Correct residual baseline distortion if applicable
One of the two polarisations is noisier than the other by more than $\sqrt{2}$ %. Their noise ratio is given in the value column.	The two polarisations cannot be averaged in order to improve the noise	Discard one of the two polarisations in your data if necessary
Out of limit in Zeros spectra	Saturation may apply to some spectral ranges	If applies, mask saturated channels
Platforming present in overlapping subbands	The baseline levels between consecutive subbands in overlapping channel ranges differ by an amount larger than 1.3 times the noise in this range. Lines could fall in the overlapping channels and their profiles could be distorted.	Re-adjust baseline levels by using non-stitched (level 2) data
Problem in the calibration-tree: APE data not present in calibration tree. APE check not performed.	Usually refers to data taken in a non-standard configuration or while the instrument was OFF	If instrument or LOU band was OFF, do not use the data, otherwise ignore flag
Reference subtraction not processed - maybe identification of phases not successful	The noise and/or the baseline quality may not be as intended	If applies, correct baseline
Some data has been lost while computing the average over many datasets	The noise and/or the baseline quality may not be as intended	Use with caution if combined with other data
Some ON/OFF dataset pairs found with unequal number of rows	The noise and/or the baseline quality may not be as intended	If applies, correct baseline
Spectrum contains saturated dark	Science data are potentially saturated	Check for saturation in the data and flag if needed
Spur lines detected in the cold spectra	There are spurs in the data	Check that most of the spurs are already flagged in your data and absent from deconvolved spectra. Flag further otherwise.
Suspicious quality of the attitude reconstruction	The gyro-based pointing reconstruction was considered inadequate and the simple pointing reconstruction was used instead.	If data in Bands 1 or 2, none (if the anomalous pointing flag has been raised with a systematic offset of the observation by $> 3 \times \text{APE}$ , either a pointing problem or an error in the altitude reconstruction). If data in Bands 3 - 7 and $\text{flag} \leq 0.4$ , interpretations based on relative astrometry may be unreliable to any

Quality Flags	Consequences for science data	Action
		quantifiable level. For observations in Bands 3 - 7 which have the "gyroAttSuspicious" flag set to True and the gyroAttQuality > 0.4, be cautious when interpreting weird line ratios. For rest of observations, None.
The computed noise rms exceeds the predicted one by more than SQRT(2). Their ratio is given in the value column	The noise in the data may be larger than normally achieved at a particular frequency	Check for other flags that may reveal other anomalies. Verify that the spectra or portion of spectra are not affected by spurs or abnormally high noise.
The deconvolution could not be performed on at least one polarization	Indicates that the deconvolution algorithm failed to converge and create the corresponding level 2.5 product	None - in principle none of the HIFI obsids should be concerned anymore
The frequency throw is large compared to the HRS bandwidth	Folded line profiles in the HRS data may be distorted	Compare with the WBS data to figure out whether folded HRS spectra have been affected
The intended and the computed pointing differ by more than three times the APE (the ratio is given in the value column)	The data have been taken slightly OFF from the intended position	Take into account the reported position offset when interpreting the data
The noise ratio of the two polarisations exceeds that expected from the measured Tsys ratio by more than 10 %. Their ratio is given in the value column.	One of the two polarisations is under-performing and may suffer from very poor noise and baseline	Discard concerned polarisation if the data are unfit
Unable to apply the HEB electric standing wave correction to H polarisation	The polarisation H could not be corrected	If residual electrical standing wave remain, clean baseline
Unable to apply the HEB electric standing wave correction to V polarisation	The polarisation V could not be corrected	If residual electrical standing wave remain, clean baseline

### Class 3 Flags

Quality Flags	Consequences for science data	Action
A Zero scan could be impacted by a High Energy Cosmic Ray	None, unless additional flags such as the COMB or ZERO ones are raised. In that case the frequency calibration can be affected.	None
At least a Zero is out of limit	None as long as not all ZEROs are failed	None, unless additional flags such as the COMB or ZERO ones are raised. In that case the frequency calibration can be affected.
At least one COMB could not be fitted	None as long as not all COMBs are failed	None, unless additional flags such as the COMB or ZERO ones are raised. In that case the frequency calibration can be affected.

Quality Flags	Consequences for science data	Action
Bad (corrupted) hc data discarded	Data have been cleaned from corrupted data-frames	None
Bad (corrupted) science data discarded	Data have been cleaned from corrupted data-frames	None
FPU Check: chopper measured values differ from the commanded	Range: [nom_offset - 0.05 V, nom_offset + 0.05 V]. This is a HK downlink issue but data are not affected by this flag.	None
FPU Check: Diplexer Current is Out Of Limit	Range: It depends of the band. One or more science data-frames are using a mis-tuned diplexer, leading to insensitive and/or mis-calibrated data.	None. In cases where an instrument issue occurred this issue has been already taken into consideration
FPU Check: Diplexer Resistance is Out Of Limit	Range: [nom_value x 0.8, nominal_value x 1.2]. Usually no impact on data.	None
FPU Check: Mixer Magnet Resistance is Out Of Limit	Range: [nom_value x 0.8, nom_value x 1.2]. Usually no impact on data.	None
FPU Check: Mixer Voltage is Out Of Limit	Range: : [nom_value - 100 $\mu$ V, nom_value + 100 $\mu$ V]. Usually no impact on data.	None
Frequency checks and/or frequency grouping failed	Usually no impact on data	None
HD247194 (HL_ptv_checksum) out of limit	It applies to a parameter used at ESOC for real time uplink. No science data impact.	None
HM025193 (HWH_Laser1_C) out of limits	Usually limited to the obsid where the laser is switched on	None
HM029191 (HF_AH1_MXBIAS_V) out of limits	Usually limited to the obsid where the band is switched on	None
HM120191 (HF_AV1_MXBIAS_V) out of limits	Usually limited to the obsid where the band is switched on	None
Max percentage of Bbids corrected according to commanded Bbids	Only applicable to pre-launch data - no impact on flight data	None
Max percentage of Dataframes which have unaligned HK	This has no consequences on the data	None
Max percentage of DFs having no chopper information	Usually no impact on data	None
Max percentage of DFs having no commanded chopper information	Usually no impact on data	None
Max percentage of DFs having no frequency monitor information	Usually no impact on data	None
Max percentage of DFs having no LO Code main information	Usually no impact on data	None

Quality Flags	Consequences for science data	Action
Max percentage of DFs having no LO Code offset information	Usually no impact on data	None
Max percentage of DFs having zero values in the HRS Correlation Factors	One or more data-frames have corrupted correlation function at Level 0	None - this should have been taken care of by the pipeline
Maximum number of spikes detected in a Comb	Spikes may be present in comb	None, unless additional flags such as the COMB or ZERO ones are raised. In that case the frequency calibration can be affected.
Number of distinct buffer values not as expected in all datasets	Buffer read out may have been slower than actual chopper rotation speed. Usually no impact on data.	None
Number of distinct Chopper values not as expected in all datasets	Chopper readout may have been slower than actual rotation speed. Usually no impact on data.	None
Number of distinct LOF values not as expected in all datasets	In most cases this is because of timing issues in the OBSW in fast chop mode and the impact on data is null.	None
Pattern observed for the buffer not as expected in all datasets	Buffer read out may have been slower than actual chopper rotation speed. Usually no impact on data.	None
Pattern observed for the Chopper not as expected in all datasets	Chopper readout may have been slower than actual rotation speed. Usually no impact on data.	None
Problem occurred while computing channel-dependent weights. No weights added.	Negligible	None
Remaining bad (corrupted) science data at Level 2	One or more data-frame in the level 2 was corrupted and has been flagged as such. Those data have been discarded from the level 2.5 products.	None
Unordered or duplicate DataFrames found	When this occurs this should show as corrupted DF symptoms, treated elsewhere	None
WM409565 (HifiL-CU_R_L54_I) out of limits	No impact on the science data	None
WM508565 (HifiHRH_L1632_I) out of limits	No impact on the science data	None
WM608565 (HifiHRV_L67_I) out of limits	No impact on the science data	None

# Chapter 11. How to add and remove flags

Last updated 1 May, 2015.

## 11.1. Introduction

While chapter [Chapter 10](#) provides an overview of all the flags (channel flags, column rowflags, and quality flags) used to identify affected data, this chapter teaches you how to inspect your data to identify if any of it has been flagged by the pipeline. You will also learn how to set and clear flags both for a data point, and over a range.

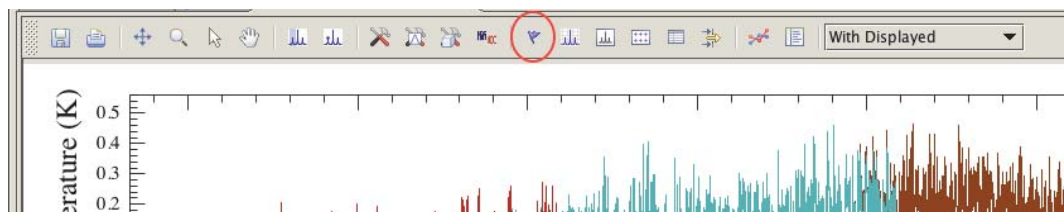


### Note

Although we provide a range of methods to flag data, we recommend that you use the task `flagTool` (see [Section 11.5](#)).

## 11.2. How to understand what flags are in your data

The pipeline may have raised a flag (or some flags) in your datasets during the different levels of processing. Using the Spectrum Explorer, you can view channel flags in your spectrum by clicking on the *blue flag* icon in the button bar at the top of the Spectrum Explorer window.



**Figure 11.1.** To view channel flags in your spectrum, click on the *blue flag* icon (highlighted by the red oval)

Coloured bars appear over the spectrum in the regions where data is flagged. The colour mapping applied to the flags is random. You can also see what kind of flags were applied by moving the mouse over the spectrum - the information is given in the bottom left of the plot window in form of [row, segment, channel, flag name] (see the next two figures).

Alternatively, you can right click in the window of the spectrum, then select *view*, and then *flags*.

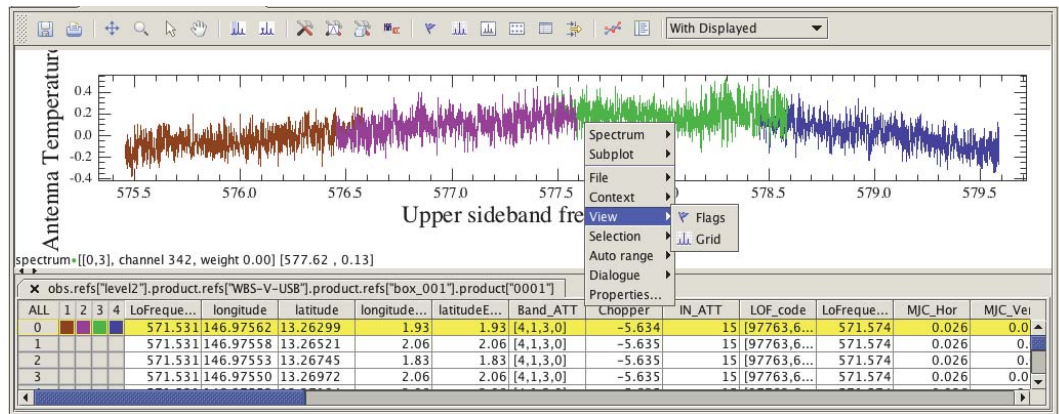


Figure 11.2. To view channel flags in your spectrum by using a pointing device such as a touchpad or a mouse

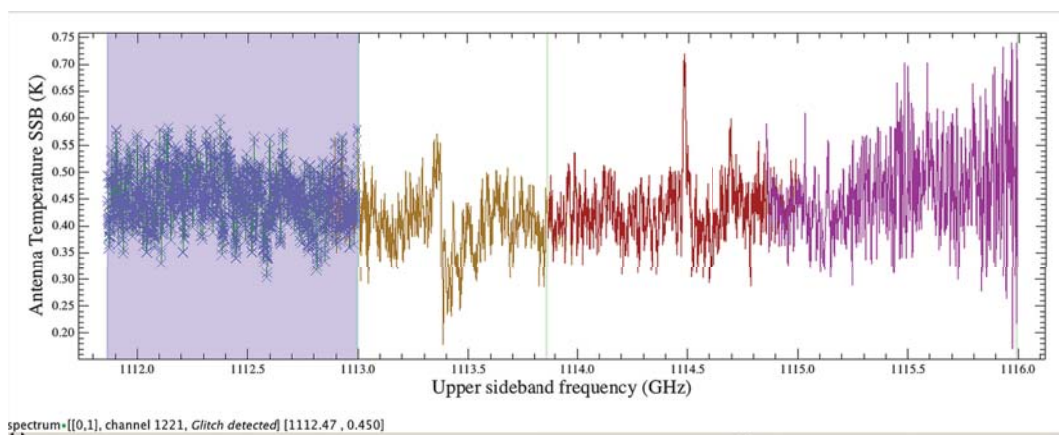


Figure 11.3. The region flagged is colour-coded

Column RowFlags are not directly overlaid on the Spectrum Explorer since, by definition, they affect the entire spectrum. Rowflags are stored in extra columns within your dataset, and are shown as a table with the Spectrum Explorer. In [Figure 11.2](#), you see an example of this table underneath the spectrum, with columns for LO frequency, longitude, Band, etc. Scrolling over, you will find the 'rowflag' column.

A flag value may just be a single bit with an already defined value (see [Chapter 10](#)), e.g. rowflag = 256 corresponds to bit = 8 (*Unaligned\_HK*) (i.e.  $value=2^8$ ), or it can also be a combination (by addition) of two or more flag values, e.g. rowflag = 262400 is a combination of the rowflag *Suspect\_LO*: bit = 18, value = 262144, and the rowflag *Unaligned\_HK*: bit = 8, value = 256.

To determine if a particular bit is set, you can employ the mask classes defined in [Chapter 10](#). For example, to test if a particular rowflag has the *UnalignedHK* bit set, you can type:

```
print RowMask.UNALIGNED_HK.isSet(rowflagvalue)
```

A script to test all rowflags is given below:

```
from herschel.hifi.pipeline.product import RowMask
#
## Extract the fourth spectrum from level 1 WBS-H in the
## Observation context
#
ds=obs.getProduct('level1').getProduct('WBS-H').get(4)
#
# Extract the rowflags and print out which are set:
```



```
#
flags=ds['rowflag'].data
print 'PACKET_ORDER: ', RowMask.PACKET_ORDER.isSet(flags)
print 'PACKET_LENGTH: ', RowMask.PACKET_LENGTH.isSet(flags)
print 'TOO_MUCH_DATA: ', RowMask.TOO_MUCH_DATA.isSet(flags)
print 'FIRST_PACKET: ', RowMask.FIRST_PACKET.isSet(flags)
print 'NO_BLOCK: ', RowMask.NO_BLOCK.isSet(flags)
print 'UNALIGNED_HK: ', RowMask.UNALIGNED_HK.isSet(flags)
print 'NO_CHOPPER: ', RowMask.NO_CHOPPER.isSet(flags)
print 'NO_COM_CHOP: ', RowMask.NO_COM_CHOP.isSet(flags)
print 'NO_FREQ_MON: ', RowMask.NO_FREQ_MON.isSet(flags)
print 'NO_LO_OFF: ', RowMask.NO_LO_OFF.isSet(flags)
print 'NO_LO_MAIN: ', RowMask.NO_LO_MAIN.isSet(flags)
print 'MCD_REF: ', RowMask.MCD_REF.isSet(flags)
print 'MCD_OFF: ', RowMask.MCD_OFF.isSet(flags)
print 'MCD_HOT: ', RowMask.MCD_HOT.isSet(flags)
print 'NO_HOT_COLD: ', RowMask.NO_HOT_COLD.isSet(flags)
print 'SUSPECT_LO: ', RowMask.SUSPECT_LO.isSet(flags)
print 'SPUR_DETECTED: ', RowMask.SPUR_DETECTED.isSet(flags)
print 'IGNORE_DATA: ', RowMask.IGNORE_DATA.isSet(flags)
print 'BBID_CORR: ', RowMask.BBID_CORR.isSet(flags)
print 'PERM_USER_FLAG: ', RowMask.PERM_USER_FLAG.isSet(flags)
print 'TEMP_USER_FLAG: ', RowMask.TEMP_USER_FLAG.isSet(flags)
```

## 11.3. Safe Usage of Flags

**Channel Flags.** Bits 0-7 of the *HifiMask* definitions (see [Chapter 10](#)) are reserved for use by the pipeline. Nothing will prevent you from setting and clearing these flags in your data, but there may be consequences if flags are manipulated at Level 0 or 1, and then the data is re-pipelined up to Level 2. For example, the *Bad Pixel* flag, when set, means that the data is completely ignored by the pipeline. The spur candidate bit on the other hand, is simply carried through from level to level without affecting how the pipeline treats the data. This may change in the future, and therefore it is recommended that you employ bits 29 and 30 as much as possible when flagging data by channel. Bit 29 is meant to represent data which should be ignored. Bit 30 is a temporary flag that will discard data only for dedicated functionalities (e.g. baseline fitting, etc...).

**Row Flags.** Bits 0-19 of the 'RowMask' definition (see [Chapter 10](#)) are reserved for use by the pipeline. Again nothing prevents you from setting or clearing them, but results when re-pipelining may not be predictable. You are encouraged to use bit 20 *IGNORE\_DATA* to flag spectra that are bad. Currently, only the deconvolution task honours the *IGNORE\_DATA* flag.

## 11.4. Setting and Clearing Flags with SpectrumExplorer

In the Spectrum Explorer button bar, you will find a button called *select one or more datapoints* which will allow you to flag one or more data points (see [Figure 11.4](#)).

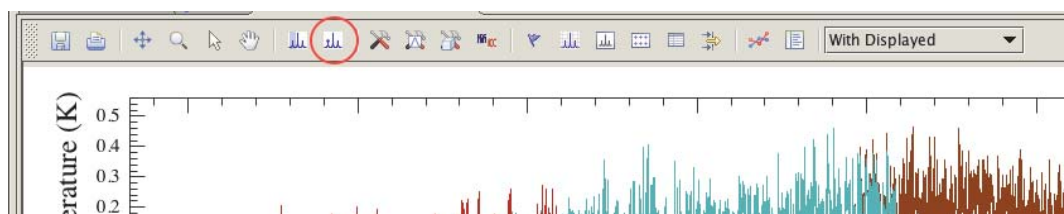


Figure 11.4. Data point selection from the Spectrum Explorer button bar (highlighted in the red circle)



### Note

Flagging data involves adding more information to it, so you are actually overwriting your data. The flagging task will do this so long as the data is in memory. However, if it comes

from a storage you must create a new variable and add the flags to it. When working with an observationContext from storage this means you should create a new variable for your dataset, flag it, and set the modified data back into the context. For the moment, this still must be done manually. An example is shown at the end of this section.

### To flag a range of data points:

- First, plot your spectrum dataset in SpectrumExplorer
- Select the *select one or more datapoints* button using a left-click
- Using the left-click again, click and drag a box around the region to be flagged. Take care to completely enclose all the points you wish to flag (see [Figure 11.5](#)).
- With a right-click, select *Point selection / flag*. A drop-down menu with an extended list of masks will appear. At the bottom of the list, you will also find the choice *manual*, which allows you to select a flag value not already present in the menu. If *manual* is selected, a small window requesting a flag numerical value will appear. The numerical value of the flag should be calculated as  $value=2^n$ , where  $n$  is the bit value. For example, for bit 7 you would enter 128 and for bit 29 the value is 536870912. Once the value is set, click *OK* to continue. Note that if you selected data points in a spectrum with more than one row (typical for Level 1 HIFI data), then you will be prompted for a flag value for each of N rows in the spectra (see [Figure 11.6](#)).
- To verify your newly added flag(s), select the *red flag* button
- Before moving to a new region to be flagged you need to *deselect* the region you just flagged. This is done by moving your cursor in the flagged region, right-click, and select *Point selection / Deselect*. The flagged region will no longer be highlighted but by overing your cursor in the flagged region, you will see the flag value written in the information box (lower left corner of the plot).

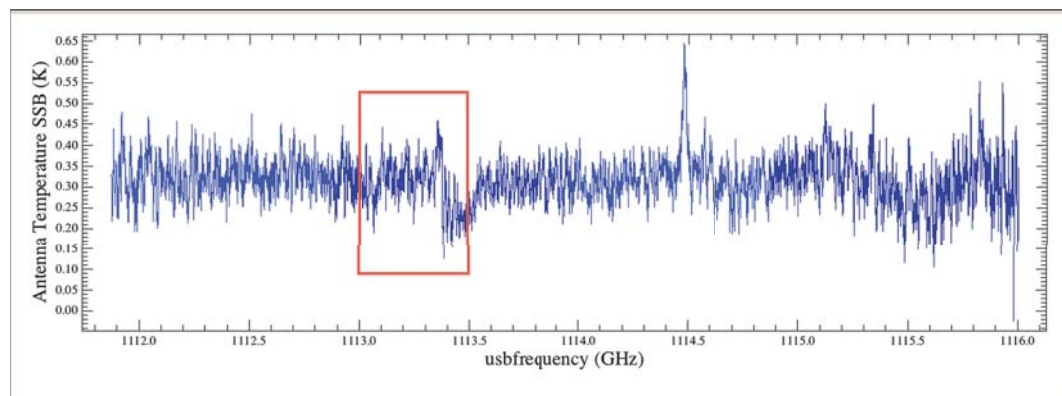


Figure 11.5. Selecting a region of the spectrum

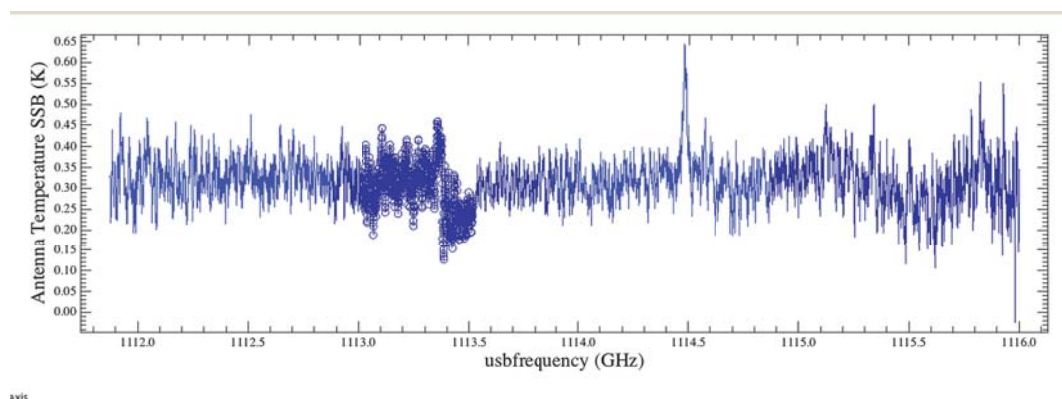


Figure 11.6. Results from flagging the selected region

**To flag a single data point:**

- Select the *select one or more datapoints* button
- Then click on the pixel of interest, a circle will be drawn around that point
- Right click on the circle and proceed as above for flagging a range

Clearing flags from a point or a range is done by selecting *manual* and using *0* as the numerical flag value.

**Example:** ObservationContexts and HTPs are passed by reference. Datasets are usually passed by copy. Thus to modify a spectrum, you first need to extract it from an HTP. After you are done flagging or working on the spectrum, it needs to be set back into the HTP. Since the HTP is a reference, this change will immediately modify the parent ObservationContext in memory.

```
## This procedure is an example of the method to extract a SpectrumDataset from a
  HifiTimelineProduct, flag some data points,
## and then re-insert the SpectrumDataset into the ObservationContext in order to
  save it.

# Create a variable of the HifiTimelineProduct

http = obs.getProduct('level2').getProduct('WBS-V-USB')

# Create a variable of the SpectrumDataset you need to flag: here, we extract from
  box_0001, spectrum 0007

sds7 = http.get(7)

## Follow the procedure described above to either flag a single data point, or a
  range of data points
## on the SpectrumDataset sds7.

## Re-insert your flagged SpectrumDataset into the ObservationContext via the
  HifiTimelineProduct

http.set(sds7,7)
```

## 11.5. Setting flags interactively for many spectra

The `flagTool` task allows you to interactively flag (via a graphical interface) rows and channels in HIFI for a given spectrometer. The task provides a user friendly environment and allows you to easily rerun the task and modify your masks. With the concept of an interactive table of datasets provided with the plot (left side of the plot), it is visually easy to see where your flags are (per dataset and per subband), and to choose which dataset to examine.

The `flagTool` task is applied on an ObservationContext (recommended) or a HifiTimelineProduct (`http`) and once selected, you will find `flagTool` in the *Applicable Tasks* menu where you can open the GUI from there (see [Figure 11.7](#)). The ObservationContext (or `http`) will then be loaded into the *product* bullet. You can also open the GUI from the *HIFI* list of tasks under *By Category*. Please note that we recommend using the `flagTool` task on an ObservationContext as opposed to a `http`. This is because the modifications applied on a `http` is not conserved unless you re-insert the `http` into the ObservationContext. This can be achieved by using the following command line:

```
obs.getProduct(level).setProduct(backend, http)
```

### Capabilities.

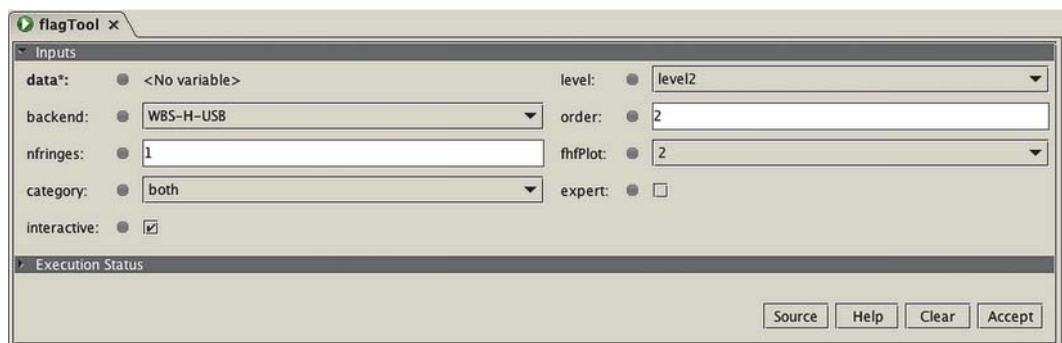
- Natural zooming enabled in the plot window (by drawing a box (left-click), or by using the mouse wheel (Mac trackpad: two-finger pinch, or two-finger drag across the pad))
- Easy channel flag switching (via four clickable choices situated at the top of the plot)
- Limited rowflag setting (SUSPECT\_LO and IGNORE\_DATA)
- Subband pseudo-gluing (the default) by choosing a dataset in the interactive table (left-click or right-click)
- Single subband plotting by choosing the subband in the interactive table (left-click or right-click)
- Easy drawing of a mask region (by holding *shift and left-click* on the left border of a region and dragging to the right border of the region)
- Category of flags to process (channel, rowflag, both (default))
- Embedded use of `fitHifiFringe` and `fitBaseline` for Level 2 data (via clickable choices at the bottom of the plot)
- Interactive and Expert modes

**(Mac users:** If you are using a trackpad instead of a 3-button mouse, you need to configure your trackpad for *One Finger* and *Two Fingers*. And, in order to emulate the middle-click mouse button, you just use *Ctrl + two-finger tap*.)

You can select the spectrometer (HRS or WBS), the polarisation (H or V), and the sideband (LSB or USB) to be processed by using the *backend* drop-down menu (e.g. WBS-H-USB, WBS-V-LSB, HRS-V, WBS-H, etc..) in the GUI (for the command line syntax, see examples at the end of this section). The level of the data to be processed must also be selected using the drop-down menu called *level*. Note that for Level 1, the *backend* choices should be HRS-H, HRS-V, WBS-H, or WBS-V i.e. spectrometer and polarisation. For Level 2, you must add the sideband choice. If your selection is wrong, e.g. if you choose *backend=WBS-H-USB*, and *level= level1*, the task will end and you will see a descriptive error message in the *Console* window. Simply reselect the proper level, are restart the task.

If you select the HRS backend, note that prior to HIPE 9.0, HRS data did not contain a flag column, making it impossible to flag HRS spectra. If the flag column does not exist, the task will send a comment *Column name 'flag\_1' not found* in the console window and the task will not start. Therefore, if you wish to flag HRS data, check that the SPG version (you can find this information in the ObservationContext metadata) begins with 9.0 or higher. If it does not, you will need to reprocess your observation from Level 0 using the pipeline from HIPE 9.0 onwards (see [Chapter 5](#)).

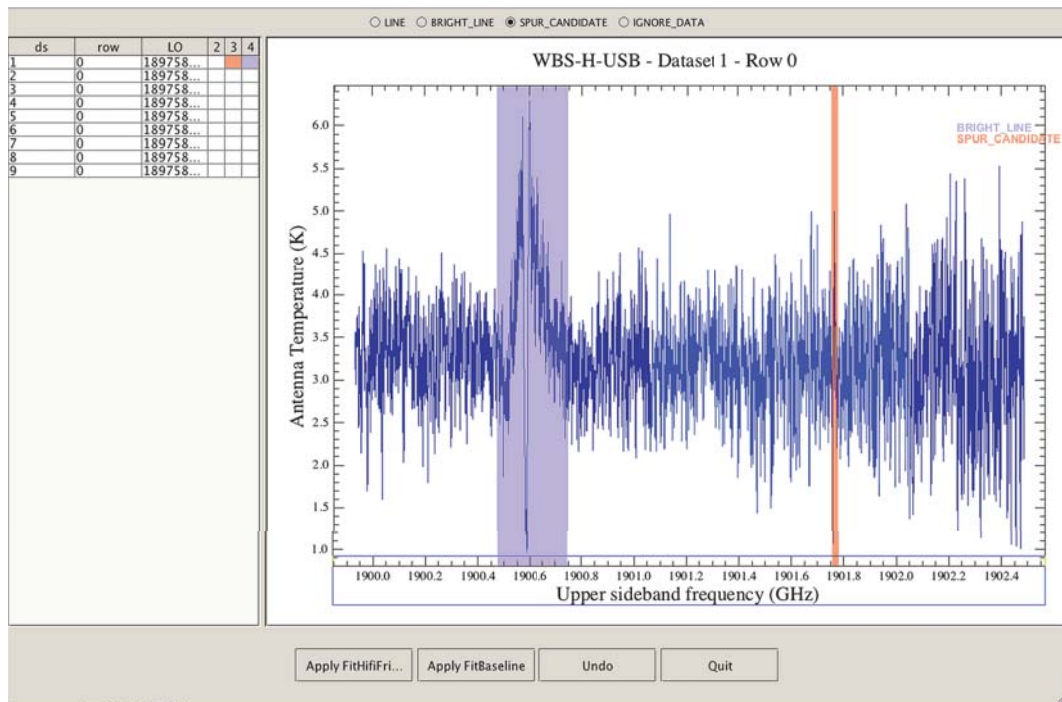
You are also able to choose to run `flagTool` with the flags category set to either *channel* only, *rowflags* only, or with both categories (the default setting), i.e. by selecting *both*.



**Figure 11.7. FlagTool task GUI**

By default, `flagTool` display the first dataset, and all subbands at once, thus allowing you to process all subbands of a dataset in one go. This feature allows you to process your data with significantly reduced time, and to flag regions where two subbands connect. But you can also choose to plot one subband at a time for a given dataset with one right-click (or left-click) in a subband box. It is fast and easy to choose which dataset to process by selecting (right-click or left-click in the `ds` column) a dataset in the interactive table on the left side of the plot (see [Figure 11.8](#)). Note that you work on one dataset at a time i.e. that flagging `ds 2, subband 3` will not be propagated to the other datasets. Therefore, you need to examine all datasets to establish which ones need flagging.

Once the task is invoked, you will be presented with a datasets table containing, at least, column 1: `ds` (dataset), column 2: `row`, and column 3: `LO` value associated with the dataset. Then, prior to running the task, if you chose `category = channel`, you will see additional columns associated with each subband. If you chose `category = rowflag`, you will see column 1: `ds` (dataset), column 2: `row`, column 3: `LO` value associated with the dataset, and additional columns with the name of all rowflags raised (in at least one spectrum) in your datasets except `UNALIGNED_HK` since this rowflag is very common in most datasets. In addition to the rowflags raised in your data, you will also see the two rowflags `SUSPECT_LO` and `IGNORE_DATA`. Note that these two rowflags are the only rowflags that you are permitted to modify. The rowflags raised by the pipeline are *read only* and will not be editable. A cross (or a marker) will indicate that the rowflag is present in a given dataset. If you chose `category = both`, you will see all columns from channel and rowflag displayed.



An interactive table listing all the datasets of an HTP is provided on the left side of the plot. This example shows the `category = channel` table.

**Figure 11.8.** FlagTool datasets table and plot showing two flagged regions (coloured 'curtains') using two different flags

```

Summary:
* level : level2
* backend : WBS-H-USB
* nfringes : 1
* order : 2
* fhfPlot : 2

Note: A backup of the maskTable is made every 5 masks.

=====

Available actions with:

* Radio buttons:
  1. Select the wanted flag
* Plot:
  1. Create masks by dragging with Shift-click
  2. Disable existing masks by clicking on them.
  3. Use mouse wheel to zoom in and out.
* Table:
  1. Plot all subbands of a given dataset by clicking on the corresponding row of the table
  2. Plot a single subband by right-clicking on the corresponding cell of the table
* Buttons:
  1. Apply FitHifiFringe or FitBaseline
  2. Undo the modifications done on the current dataset
  3. Exit FlagTool

=====

[Scan of the flags in WBS-H-USB...]
  Loading channel flags...
  Loading row flags...

```

Summary of the HTP being processed and a list of available actions appearing in the console.

**Figure 11.9. FlagTool messages in the console**

### Interactive and Expert modes.

By default the interactive option is set to 'True'. If you wish to use `flagTool` in batch mode, you can unselect ('False') this option. The interactive mode defines the verbosity of `flagTool`: if set to 'True', all actions on the flags are shown in the *Console* window.

By default, the expert option is set to 'False'. Expert mode set to 'True' affects only the row flags by focusing only on the `IGNORE_DATA` flags. If combined with `Interactive = 'False'`, the text in the *Console* window is kept to its minimum.

### Details of channel flagging.

The channel flagging technique consist of 1] choosing a flag type, and 2] defining a mask region. By default, the flag will be set to `LINE` but you can choose a flag from a sets of four types of masks via four clickable choices situated at the top of the plot. You have a choice of `SPUR_CANDIDATE` (value= $2^7$  Orange), `LINE` (value= $2^{28}$  Pink), `BRIGHT_LINE` (value= $2^{29}$  Blue), and `IGNORE_DATA` (value= $2^{30}$  Green). Once you start the task, the console will show a summary of the HTP being processed, and will show a list of available actions (see [Figure 11.9](#)).

You define a mask region (i.e. a frequency range) by holding *shift and left-click* on the left border of a region and by dragging to the right border of the region. The mask is then visible as a coloured 'curtain' (see [Figure 11.8](#)). To undo the most recent flag, use the *Undo* button at the bottom of the plot. Note that the *Undo* button only works right after you have assigned a mask, and if you have not moved to a new dataset. But, if you need to undo any masks, you can use one left-click within the coloured 'curtain'. If you choose a flag type and apply it but then decide that you need to re-assign a different type to the region, or if you need to modify the region, you must first disable the mask with a one left-click within the coloured 'curtain', choose a flag type, and then re-define the mask region.

To understand the implication of those flags, we strongly suggest that you consult the chapter covering flags in HIFI (see [Chapter 10](#)).

You may also zoom in and out by drawing a box around the region of interest or by using the mouse wheel (Mac trackpad: two-finger pinch, or two-finger drag across the pad). Using the right-click will allow you to use the default menu of PlotXY.

If a flag is set, `flagTool` will:

- draw a mask in the plot window with the colour associated with the flag
- colour the corresponding cell in the datasets interactive table with the colour associated with the flag
- flag all masked channels with the corresponding flag value
- create the corresponding mask in the `maskTable`

Note that `SPUR_CANDIDATE` and `IGNORE_DATA` prevail upon `LINE` and `BRIGHT_LINE`. As a consequence, the flag value of the channels you flagged as both `SPUR_CANDIDATE` (or `IGNORE_DATA`) and `LINE` (or `BRIGHT_LINE`) will only be `SPUR_CANDIDATE` (or `IGNORE_DATA`).

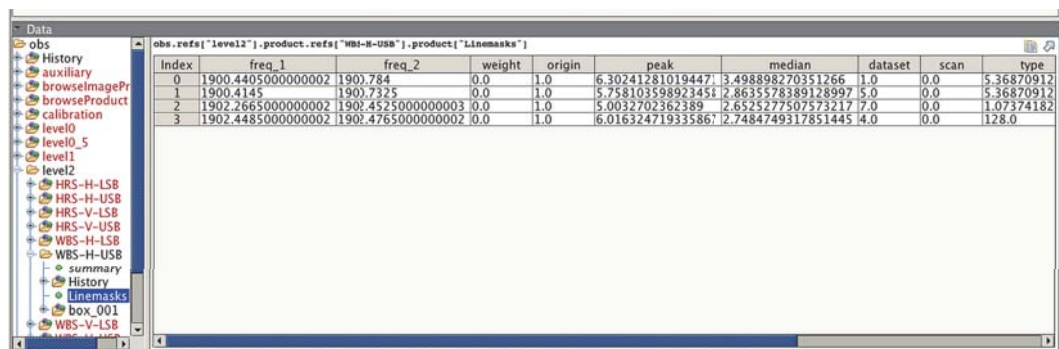
As you progress in flagging your data, please pay attention to the comments available from the Console window. Some warnings may be important about the choice of flags you may have used. Once you are done flagging your datasets, you may quite the task (*Quit* option at the bottom of the plot).

### Handling rowflags.

The purpose of listing, in the interactive table, all rowflags raised by the pipeline (except for `UNALIGNED_HK`), is to allow you to have a clear overview of potential problems in your data. As mention earlier, the only two rowflags that you should, if necessary, modify are `SUSPECT_LO` and `IGNORE_DATA`. You may raise a flag, or remove one raised by the pipeline if you judge that the quality of the dataset is good. A simple click will allow you to either select or deselect a rowflag in a given dataset.

### MaskTables: Linemasks and Rowmasks files.

`flagTool` stores the channel flags masks and row flags masks in mask tables. Channel flags masks are saved in the variable *Linemasks* and row flag masks are saved in the variable *Rowmasks*. You can find both of them in the HTP you processed, and view your masks by selecting the variable (see [Figure 11.10](#)). Mask tables have the same format and the same characteristics as `TableDatasets` so you should not be disoriented. They are specific data structures to handle masks and provide useful functions as we will see.



Index	freq_1	freq_2	weight	origin	peak	median	dataset	scan	type
0	1900.4405000000002	1901.784	0.0	1.0	6.30241281019447	3.498898270351266	1.0	0.0	5.36870912
1	1900.4145	1901.7325	0.0	1.0	5.758103598923451	2.8635578389128997	5.0	0.0	5.36870912
2	1902.2665000000002	1902.4525000000003	0.0	1.0	5.0032702362389	2.6525277507573217	7.0	0.0	1.07374182
3	1902.4485000000002	1902.4765000000002	0.0	1.0	6.016324719335867	2.7484749317851445	4.0	0.0	128.0

**Figure 11.10. FlagTool Linemasks table**

If the *Linemasks* and *Rowmasks* products are already present in an `ObservationContext`, you will see these masks already in the data and you can modify them (remove or select a different mask) and/or add new masks by just running the task again.

`flagTool` also automatically creates a backup FITS file in your working directory from a channel flag mask table for every 5 masks that you set. This feature was implemented in order to avoid losing an extensive flagging session due to an unexpected termination of HIPE. The backup file will be named *backup\_obsid\_backend.fits*. If your flagging session was less than 5 masks then you will not see this backup file.

**Note**

You may not be able to use mask tables created from older HIPE versions due to an incompatibility in the table format. Work is in progress to implement a functionality to allow compatibility between the old and the new format.

While you can handle the mask tables like any other TableDataset (drag and drop, Send to FITS file...) `flagTool` provides you with specific functions to add, load, remove, and save channel flag and row flag mask tables.

The three operations that alter the mask tables (add, load, remove) use the same syntax:

```
<operation> MaskTables({backend: maskTable})
```

They should be called **before** the `process` method because they take into account the mask table provided, and analyze it before flagging the data.

**Add new masks from a mask table**

```
ft = FlagTool()
ft.addMaskTables({backend: maskTable})
ft.process(obs, backend)
```

Only the difference between the mask table in the data and the mask table provided is taken into account. If the new mask table contains a mask that is not in the data, this mask will be added to the data:

**How addMaskTables work**

In the data	In the maskTable	Masks to be added	Final maskTable
A	A		A
B			B
	C	C	C
D			D
E			E

**Load all the masks from a mask table**

```
ft = FlagTool()
ft.loadMaskTables({backend: maskTable})
ft.process(obs, backend)
```

All the previous user row flags (if `maskTable` contains row flags) or all the channel flags (if `maskTable` contains channel flags) will be removed and replaced by the new mask table provided:

**How loadMaskTables work**

In the data	In the maskTable	Final maskTable
A	A	A
B		
	C	C
D	D	D
E		



### Remove the masks contained in a mask table

```
ft = FlagTool()
ft.removeMaskTables({backend: maskTable})
ft.process(obs, backend)
```

All the masks in your data that match the masks in *maskTable* will be removed:

#### How removeMaskTables work

In the data	In the maskTable	Masks to be removed	Final maskTable
A	A	A	
B			B
	C		
D			D
E			E



#### Note

- The rule about row flags: You can add or remove user row flags but not row flags set by the pipeline (see [Section 10.3](#)).
- The rule about channel flags: You can remove any channel flag but you can only add user channel flags (see [Section 10.2](#)).

### Save the masks to a FITS file

The last operation (save) should be called **after** the *process* method, when the flagging is done, and all the masks are updated in the mask table. It allows you to save your channel flag or row flag mask tables to a FITS file:

```
ft = FlagTool()
ft.process(obs, backend)
ft.saveMaskTables(filename)
```

*saveMaskTables* looks for *Linemasks* and *Rowmasks* in the HTP. If any of these 2 mask tables is found, it is saved to a FITS file. Note that *filename* is optional. If it is provided, the file will be suffixed by *\_Linemasks* or *\_RowMasks*. If you do not provide a filename, the file will be saved in your working directory under the name *<obsid>\_<backend>\_Linemasks.fits* or *<obsid>\_<backend>\_Rowmasks.fits*.

### A simple use case to handle mask tables

An example will make things clearer. Let's imagine you were setting channel flags with *flagTool* on the obsid 1342191559, on WBS-H-USB but something went wrong and your session was closed. Don't panic. *flagTool* made a backup of your channel masks in your working directory. You just have to reload your obsid and add these masks back into WBS-H-USB:

```
# Load the observation context
obs = getObservation(1342191559)

# Create a FlagTool object
ft = FlagTool()

# Add the mask table named "backup_1342191559_WBS-H-USB.fits" into WBS-H-USB
ft.addMaskTables({"WBS-H-USB": "backup_1342191559_WBS-H-USB.fits"})

# Process WBS-H-USB
ft.process(obs, "WBS-H-USB", level=2, category="channel")
```

FlagTool will read all the masks in the mask table *backup\_1342191559\_WBS-H-USB.fits* and set the corresponding flags. Messages in the HIPE console will list all the masks added.

After you set all the flags you wanted, you will probably want to save your masks:

```
ft.saveMaskTables(filename="finished_1342191559_WBS-H-USB.fits")
```

FlagTool appended *\_Linemasks* to your filename because it found only *Linemasks* in WBS-H-USB.

One backend is finished. Let's do the same for WBS-V-USB. But this time, we want exactly the same masks as in WBS-H-USB. You cannot use *addMaskTables* because it will only add the masks that are not already in WBS-V-USB, which means that the masks you removed in WBS-H-USB will not be removed in WBS-V-USB. What you want is to load all the masks as if no other masks previously existed in the data. This is when *loadMaskTables* comes in handy:

```
ft = FlagTool()
ft.loadMaskTables({"WBS-V-USB": "finished_1342191559_WBS-H-USB_Linemasks.fits"})
ft.process(obs, "WBS-V-USB", level=2, category="channel")
```

All the channel flags existing in WBS-V-USB are now removed and replaced by the masks in *finished\_1342191559\_WBS-H-USB\_Linemasks.fits*.



#### Note

As you can imagine, *loadMaskTables* can be dangerous and must be used carefully because it removes all the channel flags including those set by the pipeline. However, this behaviour is coherent with what you can do interactively in *flagTool* because it will only remove or add user row flags, preventing you from altering row flags set by the pipeline.

You are done! But if, for some reason, you had to remove some masks in WBS-V-USB, and let's say that these masks are stored in *spurs\_in\_1342191559\_WBS-V-USB.fits*, there is a command for that:

```
ft.removeMaskTables({"WBS-V-USB": "spurs_in_1342191559_WBS-V-USB.fits"})
ft.process(obs, "WBS-V-USB", level=2, category="channel")
```

FlagTool will read the masks in *spurs\_in\_1342191559\_WBS-V-USB.fits* and remove them from WBS-V-USB. Now you know everything about mask tables and how to handle them in *flagTool*.

### FitHifiFringe and FitBaseline.

For a given dataset of Level 2, you may also choose to perform simultaneously *fitHifiFringe* and/or *fitBaseline*. The default values are to NOT perform those options. *FlagTool* allows you to define some of the main options of these two tools:

- For *FitHifiFringe*: (for more details on these options, see [Chapter 12](#))
  - *nfringes*: the number of sine waves to be fitted (default value: 1)
  - *fhfPlot* (named *plot* in the original tool): sets if, and how the results should be plotted. The dropdown menu allows you to select 0 (no plotting), 1, or 2. The default value is 2.
- For *FitBaseline*: (for more details on these options, see [Chapter 13](#))
  - *order*: the polynomial order for the fit

All other options relative to these two tools are set to their default value. This means that the option *doglue* (used in both *fitHifiFringe* and *fitBaseline*) will be set to the *flagTool* default i.e. to *True*. Some data show offsets between the subbands. If this is the case in your data, it is then unwise to use *fitBaseline* within the *flagTool* task because you will want to fit your baselines one subband at a time.

In order to apply `fitHifiFringe` or `fitBaseline` to the current dataset, select the corresponding button at the bottom of the GUI. `FlagTool` will run `fitHifiFringe` or `fitBaseline` with the options provided in the GUI. Note that the two fitting tools work exactly the same way as in *standalone mode*. Please refer to the corresponding documentation: [Chapter 12](#) and [Chapter 13](#).

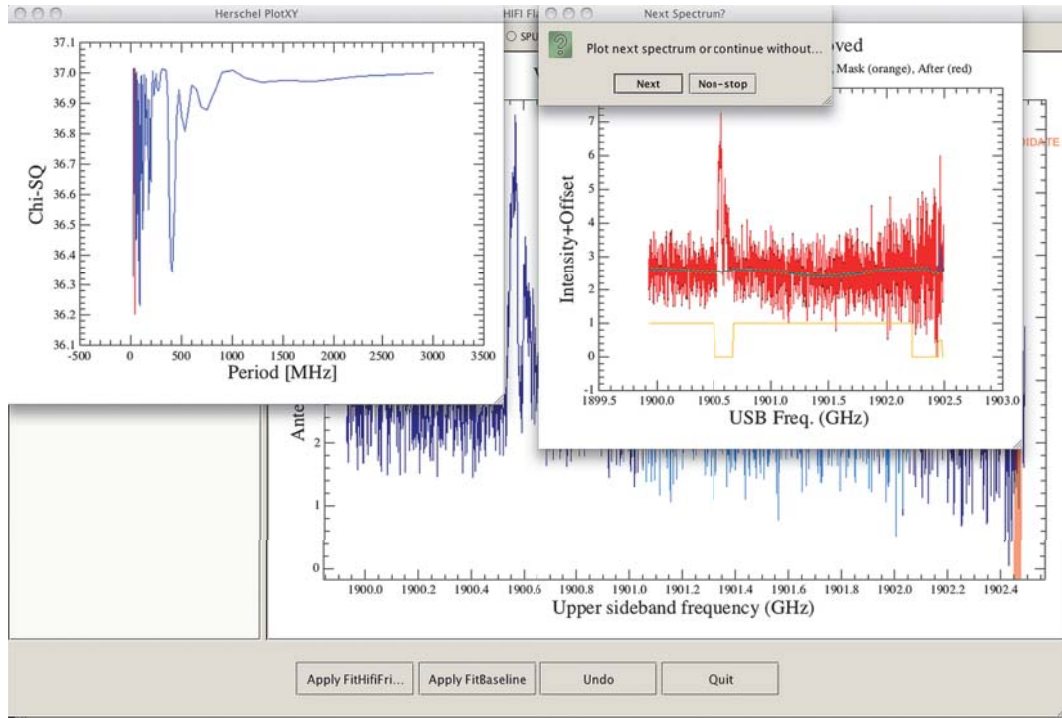


Figure 11.11. `flagTool` using `fitHifiFringe` as an option

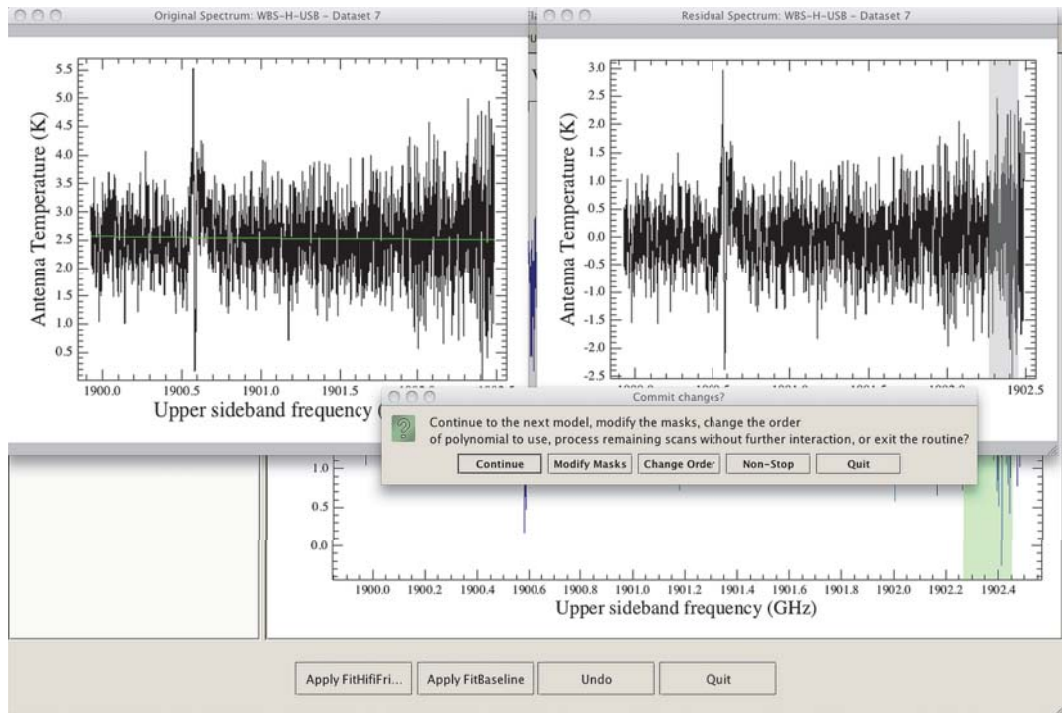


Figure 11.12. `flagTool` using `fitBaseline` as an option

`FlagTool` can also be used from the command line:

```

# Flag WBS-H data at Level 1 - by default, the category is set to "channel" flags
flagTool(obs, "WBS-H", level=1)

# Flag WBS-H-USB from the default Level 2
flagTool(obs, "WBS-H-USB")

# Flag WBS-H-USB data using a polynomial of order=3 to fit the baseline
flagTool(obs, "WBS-H-USB", order=3)

# Flag WBS-H-USB and use 2 sine waves for fitHifiFringe
flagTool(obs, "WBS-H-USB", nfringes=2)

# Flag WBS-H-USB but do not display the plot for FitHifiFringe
flagTool(obs, "WBS-H-USB", fhfPlot=0)

# Flag WBS-H-USB with the category rowflag
flagTool(obs, "WBS-H-USB", category="rowflag")

# Flag both channel flags and rowflags in WBS-H-USB
flagTool(obs, "WBS-H-USB", category="both")

```

## 11.6. Setting and Clearing Flags with command line tools

When you set or clear flags with the Spectrum Explorer, the command-line version appears in the console window and can be cut and paste for inclusion in a script. An example is shown below. We recommend that you use the following four flags, and that you use the flag name (e.g. HifiMask.BRIGHT\_LINE.value):

Flag Name	Bit	Software Name / Recommended syntax	Description	Flag value
Spur candidate	7	HifiMask.SPUR_CANDIDATE / HifiMask.SPUR_CANDIDATE.value	If this bit is set, the sample is a candidate to be a spur. It is a 'candidate' since not all things flagged by the spurfinder are necessarily spurs.	128
line	28	HifiMask.LINE / HifiMask.LINE.value	User set flag to denote a line	268435456
Bright line	29	HifiMask.BRIGHT_LINE / HifiMask.BRIGHT_LINE.value	User set flag to denote a bright line	536870912
Ignore data	30	HifiMask.IGNORE_DATA / HifiMask.IGNORE_DATA.value	User set flag to ignore data.	1073741824

```

# You cannot work from the observationContext therefore, you must create a variable
  when working
# with flagPixels.
# Here we first extract the HTP
htp = obs.refs["level2"].product.refs["WBS-V-USB"].product

# and then extract dataset 0004 from the HTP i.e. this would correspond to
# obs.refs["level2"].product.refs["WBS-V-
USB"].product.refs["box_001"].product["0004"]
sds=htp.get(0004)

```

```
# Flag some data where mask is mask={subband:[channel_numbers]} with channel numbers
# listed one by one (note
# that you cannot give a range of channels)
# and where selection[3] represents the third spectrum (row) of the spectrum
# datasets i.e.
# obs.refs["level2"].product.refs["WBS-V-
USB"].product.refs["box_001"].product["0004"].getPointSpectrum(3)
flagPixels(ds=sds, selection=[3], mask={3:[1239,1240,1241]},
  flag=HiFiMask.BRIGHT_LINE.value)

# Re-insert flagged dataset sds back into htp
htp.set(sds,4)

# Followed by a re-insertion of the htp into the obsContext
obs.level["level2"].setProduct("WBS-V-USB",htp)
```

Command-line example if we choose *flag & remove*:

```
flagPixels(ds=sds,selection=[3], mask={3:[1679]}, setFluxToNaN=1,
  flag=HiFiMask.BRIGHT_LINE.value)
```

This flagging task is not very user friendly. The *mask* represents pixel values within the spectrum. Because it is easier to identify regions by frequency ranges, we recommend that you use the new task `flagTool` (see [Section 11.5](#)).

Alternatively, you may wish to flag many spectra within an HTP where you have identified a problem at a specific frequency range.

```
# We first define the flagging routine using, as an example, the recommended
"ignore" flag
# "HiFiMask.IGNORE_DATA" (see table above)

def flagData(htp,htpindex,freq1,freq2) :

    spectrum=htp.get(htpindex)
    spectrumType=spectrum.meta["sds_type"].value # determine type (COMB, HC, SCIENCE,
TUNE)

    nrows=spectrum.getLength() # get number of spectrum in this dataset
    for dsidx in range(nrows) : # loop through them
        for sb in (spectrum.getSegmentIndices()): # loop through the subbands
            segment = spectrum.getPointSpectrum(dsidx).getSegment(sb)
            flag=segment.getFlag() # fetch the flags
            freq=segment.getWave() # fetch the frequencies
            idx=freq.where((freq>=freq1).and(freq<=freq2))
            if idx.length()>0 :
                flag[idx]=HiFiMask.IGNORE_DATA.set(flag[idx]) # set the flag
                segment.setFlag(flag) # reinsert into the spectrum
            htp.set(htpindex,spectrum,spectrumType) # reinsert modified spectra back into the
HTP
        return htp

# START OF SCRIPT

# read in your data using obs = getObservation("1342xxxxxx")
obs = getObservation("1342205481")

# extract the Level 2 tree
lev2=obs.getProduct("level2")

# from that, extract the HTP for the WBS-H backend
htpwbs=lev2.getProduct("WBS-H-USB")

# At this stage, you have a HTP that needs some cleaning.
#
# Inspect the data and identify those affected datasets, then run the cleaning
function flagData.
```

```
# In our example, we wish to flag the spectra from 3 to 6 in the frequency range
1900.49 to 1900.63

htpwbsH=flagData(htpwbsH,3,1900.49,1900.63)
htpwbsH=flagData(htpwbsH,4,1900.49,1900.63)
htpwbsH=flagData(htpwbsH,5,1900.49,1900.63)
htpwbsH=flagData(htpwbsH,6,1900.49,1900.63)

# Now with everything flagged, re-insert the data (i.e. the HTP) back into the
observation context "obs"

lev2.setProduct("WBS-H-USB",htpwbsH)
obs.setProduct("level2",lev2)

# Using Spectrum Explorer, verify, in the obsContext, that the HTP was re-inserted
into the obsContext and that
# the flags were applied accordingly (see Section 11.2).
```

### Level 2.5 'stitched' cube

Command line example to set a flag in a given spectrum of a Level 2.5 WBS 'subbands stitched' cube (pipeline product). Here, the parameter *selection* is the pixel number within the cube. Pixel numbers are labelled from 0 (lower-left corner), and increase from left to right and from bottom to top. Please be aware, however, that flags with values above 32767 (short integer limit) will not be honoured by the task in cubes as those latter are set as short integer arrays.

```
flagPixels(ds=obs.refs["level2_5"].product.refs["cubesContext"].product.refs["cubesContext_WBS-
V-USB"].product.refs["cube_WBS_V_USB_1"].product, \
selection=[31], mask={0:[1268]}, flag=HiFiMask.SPUR_CANDIDATE.value)
```

## 11.7. Scripting Techniques for bulk clearing of flags

It may happen that you will find an entire subband is flagged by either the pipeline or various tools you use to process HIFI data. It may be necessary to clear these flags to ensure that tools such as *fitBaseline* work properly. This is just one example of a case where it may be easier to write a script to deal with flagging rather than doing it via command-line tasks or the spectrum explorer. Below you will find a task that clears the *glitch* flag from ALL data within the provided HTP. It can be modified to set/clear this or any other channel flag. Indeed the loop structure (loop through the HTP, rows, and segments of spectra) has many practical applications and can be used in your own scripts.

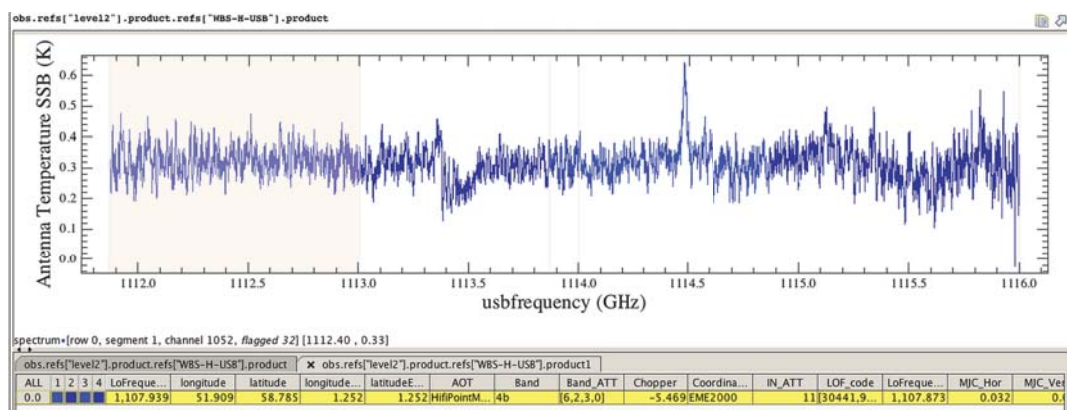


Figure 11.13. The flagged subband is colour-coded in this example

```
# This routine clears all flags in all datasets in a HTP and it is specific to the
GLITCH flag.
```

```

# In this example, we use observation "1342217724"

def clearAllFlags(htp):
    GLITCH = HifiMask.GLITCHED

# inspect summary table to get the index of all data, then loop through them
for htpindex in htp["summary"]["dataset"].data :

# fetch the spectrum from the HTP
spectrum= htp.get(htpindex)

# get number of spectrum in this dataset
nrows=spectrum.getLength()

# loop through them
for dsidx in range(nrows) :

# loop through the subbands
for sb in (spectrum.getSegmentIndices()):
    segment = spectrum.getPointSpectrum(dsidx).getSegment(sb)

# fetch the flags
flag=segment.getFlag()

# clear the spur flag
flag=GLITCH.unset(flag)

# reinsert into the spectrum
segment.setFlag(flag)

# reinsert modified spectra back into the HTP
htp.set(spectrum,htpindex)

# END OF FUNCTION DEFINITIONS

# START OF SCRIPT

obs = getObservation("1342217724")

# extract the Level 2 tree
lev2=obs.getProduct("level2")

# from that, extract the HTP for, e.g., the WBS-H and -U backends
htpwbsh = lev2.getProduct("WBS-H-USB")
htpwbshlsb = lev2.getProduct("WBS-H-LSB")
htpwbsv = lev2.getProduct("WBS-V-USB")
htpwbsvlsb = lev2.getProduct("WBS-V-LSB")

# now clear all the "GLITCH" flags
clearAllFlags(htpwbsh)
clearAllFlags(htpwbshlsb)
clearAllFlags(htpwbsv)
clearAllFlags(htpwbsvlsb)

# END OF SCRIPT

```

## 11.8. Scripting techniques for setting row flags

HIPE comes installed with the `UserFlagTask`, which sets bit 21 to user-specified rows within datasets. You can select other bits, or use the task to clear rather than set flags.

The task can be run on either a dataset, or an HTP and an index to a dataset. Example usage is given below. For more information you are directed to the [UserFlagTask](#) in *HIFI User's Reference Manual* HIFI URM (User Reference Manual) entry.

```
obs = getObservation("1342217724")
htp = obs.getProduct("level1").getProduct("WBS-H") # extract the Level 1 WBS-H tree

# In this HTP, select the 4rth dataset, and flag out rows 0 and 1.
UserFlagTask() (htp=htp,sdsnr=4,dataframes=Int1d([0,1]))

# Or, extract the dataset from the htp and operate on it:
sds=htp.get(4)

# Clear the flags
UserFlagTask() (sds=sds,dataframes=Int1d([1,2]),erase=True)

# Reinsert spectra in the HTP
htp.set(sds,4)
```



# Chapter 12. Standing Wave Removal

Last updated: 15 September, 2015

## 12.1. Introduction to Standing Wave Removal

At many locations in the HIFI instrument, standing waves are generated. The standard (SPG) pipeline and AOTs are designed to minimise the amplitude of these waves in Level 2 spectra. However, for certain observing modes, some mixer bands, and some observation characteristics (sky signal strength, signal-to-noise level) particular residual waves may still be present:

1. All bands: a 98 MHz wave from the cold black body to mixer cavity
2. All bands: a 92 MHz wave from the hot black body to mixer cavity
3. All bands: a 100 MHz wave from the local oscillator unit to mixer cavity
4. Diplexer bands (3,4,6,7): a 620 MHz wave from the diplexer rooftop to mixer cavity
5. HEB bands (6,7): a ~320 MHz wave from within the HEB mixers

Depending on the science application, the user may want to remove these waves. Different techniques may need to be applied for the different waves. These are described in the next sections.

## 12.2. Modified Passband Calibration Method

The hot and cold load spectra that are taken during every observation and that are used for intensity and passband calibration contain 92 and 98 MHz waves. In the standard pipeline they are divided into the sky spectra. Thus it is desired to remove the waves from the load measurements beforehand. This is possible by running the pipeline in a modified way and activating the `DoFilterLoadsTask` step:

```
# Obtain a params object
http = obs.level.get("level0").getProduct("WBS-H")
params = herschel.hifi.pipeline.generic.PipelineConfiguration.getConfig(http)
# Configure it
params.setParameter("doFilterLoads", "ignore", False)
params.setParameter("doFilterLoads", "filterMethod", "cubic_splines")
# Run the pipeline while passing the params object to it:
obs = hifiPipeline(obs=obs, fromLevel=0.0, upToLevel=2.0, params=params,
  apids=["WBS-H"])
```

## 12.3. Sine Wave Fitting Method (`fitHifiFringe`)

### 12.3.1. Introduction to `fitHifiFringe`

Periodic signals in HIFI Level 2 spectra can be removed with the sine wave fitting tool `fitHifiFringe`. This tool allows for automatic or manual masking of lines before fitting a user-defined number of sine waves. As it makes use of the general task `fitFringe`, details on the algorithms can be found in the [FitFringe manual](#).

The cookbooks ([Chapter 2](#)) demonstrate use of the tool on specific observations.

### 12.3.2. Running `fitHifiFringe`

`FitHifiFringe` is automatically registered to `ObservationContexts` and `HifiTimelineProducts` (i.e. when clicking on such variables in the Variable window of HIPE, `fitHi-`

fitHifiFringe shows up as an applicable task). However, you can also process SpectrumDatasets, SimpleSpectrums, and HIFI spectral cubes (SpectralSimpleCubes) by opening the GUI (see [Figure 12.1](#)) under *All Tasks* and then dragging the variable to the appropriate bullet. Alternatively, run it on the command line as follows:

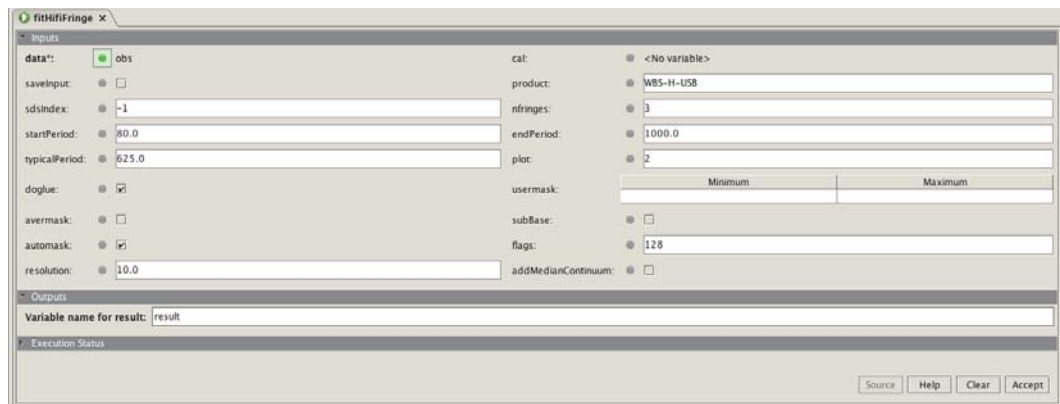


Figure 12.1. FitHifiFringe GUI

For ObservationContexts and HifiTimelineProducts:

```
result=fitHifiFringe (data=obs_in,nfringes=2,typicalPeriod=150., product='WBS-H-USB')
```

or for SpectrumDatasets:

```
result=fitHifiFringe (data=sds_in,nfringes=2,typicalPeriod=150.)
```

or for SpectralSimpleCubes:

```
result=fitHifiFringe (data=ss_in,nfringes=2,typicalPeriod=150.)
```

or for SimpleSpectrums:

```
result=fitHifiFringe (data=ss_in,nfringes=2,typicalPeriod=150.)
```

Note that in the latter case, the SimpleSpectrum input can be constructed from any spectrum of type SpectrumId as follows:

```
ss_in=SimpleSpectrum (spectrumId)
```

This allows FitHifiFringe to be applied to a Single Sideband spectrum taken from the doDeconvolution output product and to a spectrum extracted from a data cube using the extractRegionSpectrum task.

For HifiTimelineProducts, SpectrumDatasets, SimpleSpectrums, and SpectralSimpleCubes the output result is a list of products: a product that is identical to the input, but with the fitted sine waves subtracted from the flux columns (result[0]) and the summary standing wave tables (result[1]). These products are easily retrieved, for example:

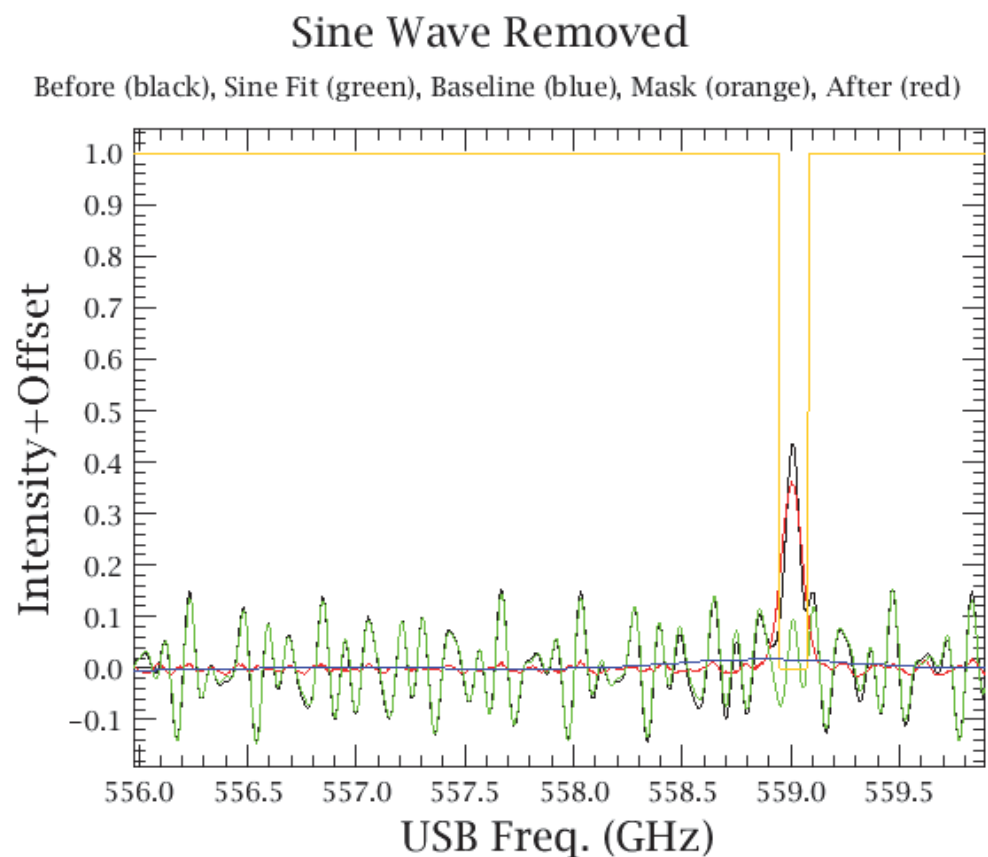
```
result=fitHifiFringe (data=obs)
myNewObs=result [0]
```

```
mySwTable=result[1]
```

Remember that because of the nature of an `ObservationContext` the original input will have the sine waves subtracted as well. If you wish to compare *before* and *after* results, you will need to read the input data separately into a different variable name.

When `subBase=1` is selected, `fitHifiFringe` subtracts the fitted baseline (shape+continuum level) together with the (multi-)sine fringe model. At the same time, a median baseline level is computed (per spectrometer subband, or as a whole, depending on the input on `doglue`) and reported in the output table `swTab` (see [Figure 12.3](#)). The parameter `addMedianContinuum=True` allows you to re-add the median baseline value computed by the task in the baseline subtraction process. This will allow you to benefit from baseline correction (shape) but preserve the overall continuum.

By default, `fitHifiFringe` produces two plots for each fitted spectrum. The first plot (not shown here) shows the chi-square values as a function of sine wave period. The second plot (see [Figure 12.2](#)) shows the input data, fitted sine waves, baseline, line mask, channel flags and output data for the user to inspect. The orange line represents all data ignored in the fitting process: values of 0.0 for line masks and values of 0.5 for channel flags. Before processing the next spectrum, you are asked for confirmation through a pop-up GUI. For products with many spectra, you can also decide to continue non-stop (without further plotting and pop-up GUIs).



**Figure 12.2. Fitted sine waves, baseline, line mask, channel flags, and output data**

The amplitudes, periods, phases, and chi-squares of the fitted sine waves are printed on the HIPE console. In case the input was an `ObservationContext`, the table is saved in the context as indicated in the screenshot below (see [Figure 12.3](#)):

Sine wave fit parameters.

Summary

AOR Label: PSP2\_HScan-0008 - 0001  
Instrument: HFI  
Obs. ID: 1342196511  
Object: LD1157-81  
Obs. Date: 2010-05-13T02:45:03Z  
AOR: Spectral Scan  
Obs. Mode: DRS  
RA: 20h 39m 10.2s  
Dec. Nominal: 68° 1' 10.5"  
SPG Version: SPG v13.0.0  
Operational Day: 364

Meta Data

Data

obs	obs.refs["calibration"]	product.refs["pipeline-out"]	product.refs["StandingWaves_WBS-H-LSB"]	product["swTable"]	relAmplitude [%]	
index	sdsindex	pointindex	segmentindex	period [MHz]	amplitude [K]	relAmplitude [%]
0	1	0	(1, 2, 3, 4)	96.96634145561879, 91.75140798757169	(0.009531623432116378, 0.007665169121788479)	-17.605196712108105, -14.1577991760
1	2	0	(1, 2, 3, 4)	85.9789716818368, 93.90746881098366	(0.011370984834261896, 0.009529619846090301)	-17.415627891897948, -14.59440431364
2	3	0	(1, 2, 3, 4)	103.81819256866531, 118.9271585635586	(0.016270520295474683, 0.01088339528562369)	-908.1010885308826, -607.4536430434
3	4	0	(1, 2, 3, 4)	96.9125309726086, 86.34835945648996	(0.0108449607778168, 0.011542570641074865)	-13.379913355187947, -14.24058585717
4	5	0	(1, 2, 3, 4)	89.19953736730616, 88.87718123330144	(0.012245044797447197, 0.010164246978407988)	-20.767608511890735, -17.2285232806
5	6	0	(1, 2, 3, 4)	120.0, 110.9885498842944	(0.011911851188861628, 0.01213788812041345)	-28.793753539972, -50.25918886129
6	7	0	(1, 2, 3, 4)	109.8849587020083, 93.15716535097251	(0.00970074698252898, 0.00955655561102827)	-04.79421019279515, 44.05803447395
7	8	0	(1, 2, 3, 4)	102.31985829129437, 104.64298790520835	(0.01005258501184864, 0.01001636553584409)	-36.11972473433466, -35.98938531437
8	9	0	(1, 2, 3, 4)	107.6908793521943, 109.8948897415329	(0.008081840834669443, 0.00873896898276805)	-19.898125824653714, -21.516011816145
9	10	0	(1, 2, 3, 4)	111.92547753287419, 84.9846442009376	(0.012219318027518742, 0.01162026088196682)	-14.5332580517746, -17.6248163755
10	11	0	(1, 2, 3, 4)	85.02419804845289, 112.58629144243827	(0.008807051449927236, 0.008625206423544458)	-34.61211434754005, -31.8924550902
11	12	0	(1, 2, 3, 4)	81.91011787680059, 98.89581959145139	(0.009684730791580136, 0.00884185080194581)	-35.40746541740254, -32.32588839172
12	13	0	(1, 2, 3, 4)	88.9865799707117, 120.0	(0.014682026797140245, 0.010281559440295232)	-16.14002790826506, 11.302602811987
13	14	0	(1, 2, 3, 4)	80.35714265714285, 90.7730266435179	(0.01105497903258632, 0.0102758410502346)	-62.810602839783606, 58.38380779608
14	15	0	(1, 2, 3, 4)	118.04202995274512, 88.01469878124492	(0.011266312408394992, 0.01019765102047924)	-45.47900367989885, 41.165111659882
15	16	0	(1, 2, 3, 4)	98.97893616313802, 103.4984775998442	(0.01081650219229275, 0.00931673800818904)	-8.930342639455858, -7.74144296393
16	17	0	(1, 2, 3, 4)	115.8255129932948, 120.0	(0.012172528174250805, 0.007129101871589954)	-31.89605679120571, 19.084227507524
17	18	0	(1, 2, 3, 4)	101.63554546896788, 109.77480416674723	(0.01141543664130852, 0.009498091809001784)	-37.626767697905581, -31.30895685866
18	19	0	(1, 2, 3, 4)	97.1589577088826, 86.2919345157547631	(0.011283719985431744, 0.00915758767226572)	-17.87790976785897, -14.5092687785847
19	20	0	(1, 2, 3, 4)	104.5405334028256, 90.16404483195013	(0.00916716647784717, 0.007404761884920609)	-24.26218992386184, -19.61150237692
20	21	0	(1, 2, 3, 4)	88.2237989852728, 101.11761010190155	(0.012937665943233607, 0.011983853841751172)	-14.66025115753581, -13.5794289282
21	22	0	(1, 2, 3, 4)	86.3258044413254, 82.54638456310944	(0.01146048796422789, 0.00954305950684541)	-13.734259286185857, -11.4364112642
22	23	0	(1, 2, 3, 4)	90.812277344885, 101.026532114161121	(0.00936117116674663, 0.010185632644108024)	-25.488790674000473, -22.7316514548
23	24	0	(1, 2, 3, 4)	116.68523216789453, 109.55501014673614	(0.009469547972021784, 0.010128523948091822)	-121.19562612959088, -129.629503465
24	25	0	(1, 2, 3, 4)	117.44894170183704, 107.32001715301092	(0.00746694453188133, 0.00736628130828046)	-82.5576594945778, 81.422783676433

Figure 12.3. Table containing information resulting from the fitting

The following input parameters are allowed:

- data*: input can be ObservationContexts (Level 1 or 2), HifiTimelineProducts (Level 1 or 2), SpectrumDatasets (WBS or HRS), SimpleSpectrums (Waveunit needs to be in GHz), and spectral cubes (SpectralSimpleCubes) (Waveunit needs to be in GHz).
- product*: if the input is an ObservationContext, indicate which Level 2 product needs to be processed. Options are: 'WBS-H-USB' (default), 'WBS-H-LSB', 'WBS-V-USB', 'WBS-V-LSB', 'HRS-H-USB', 'HRS-H-LSB', 'HRS-V-USB', 'HRS-V-LSB'.

Note that standing waves are automatically corrected in both sidebands of whichever spectrometer you select so, for example, if you select 'WBS-H-USB' the standing waves will also be corrected in the WBS-H-LSB.

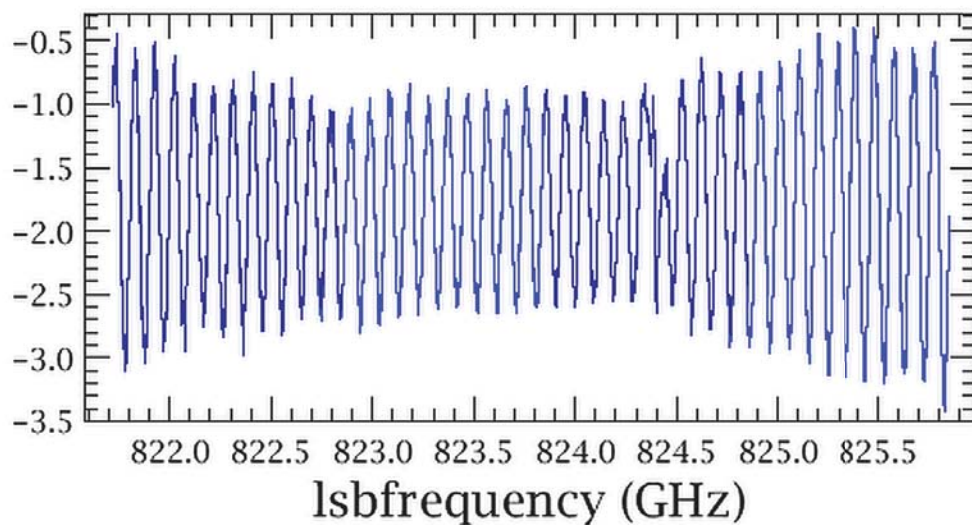
- sds\_index*: index of a single SpectrumDataset in a product to process. The index can be determined by listing the product in the Context viewer. This option is irrelevant if the input is an sds. [DEFAULT: sds\_index=-1, which means all SpectrumDatasets in a product will be processed].
- nfringes*: number of sine waves to be fitted. By default, nfringes=2 for HIFI bands 2,5, nfringes=3 for bands 1,3,4,6,7, and nfringes=1 if the HIFI band cannot be determined.
- startPeriod*: shortest-period standing wave (in MHz) to search for. By default, startPeriod=80 MHz for HIFI bands 1,2,3,4,5,6,7, and startPeriod=20 MHz if the HIFI band cannot be determined.
- endPeriod*: longest-period standing wave (in MHz) to search for. By default, endPeriod=120 MHz for HIFI bands 2,5, endPeriod=200 MHz for HIFI band 1, endPeriod=1000 MHz for bands 3,4,6,7, and endPeriod=3000 MHz if the HIFI band cannot be determined.
- typicalPeriod*: typical standing wave period (in MHz) in the data. This is used for the baseline determination. Features with much longer periods are considered baseline structure and will not be removed with sine waves. By default, typicalPeriod=95 MHz for HIFI bands 1,2,3,4,5, typicalPeriod=600 MHz for band 6, typicalPeriod=625 MHz for band 7, and typicalPeriod=150 MHz if the HIFI band cannot be determined.
- plot=...*: Sets if and how the results should be plotted.
  - plot=0: no plotting at all.
  - plot=1: two plots per scan: (1) period versus Chi<sup>2</sup> (2) the before/after plot and the subtracted sine wave and the line mask. WARNING: If the observation contains many scans, many plot windows will appear on the screen!

- `plot=2`: as `plot=1`, but after each scan, show a pop-up window to process the next scan and delete previous plot windows. One can also opt to continue without further plotting.
- [DEFAULT: `plot=2`]
- `doglue=False`, Determine SW on individual subband spectrum. This is desired for HRS, but often not for WBS, as long period SW can only be determined on the combined spectrum. [DEFAULT: `doglue=True`]
- `automask=True`: automatically mask datapoints using the sigma-clip algorithm described in the SmoothBaseline manual. This mask is added to any user-defined mask ('usermask') that is provided and any line flags (LINE or BRIGHT\_LINE) that were defined before `fitHifiFringe` was started (e.g., using `flagTool`). [DEFAULT: `automask=True`]
- `usermask=...`, mask frequency ranges in addition to the automatically determined mask ('automask') and any line flags (LINE or BRIGHT\_LINE) that were defined before `fitHifiFringe` was started (e.g., using `flagTool`). Example: `usermask=[(537.0,538.0), (539,539.5)]` marks the ranges 537-538 GHz and 539-539.5 GHz [DEFAULT: only use automatically determined mask]
- `avermask=True`, determine the mask on the average of all scans and then apply this mask to each scan. This reduces the amount of work for large maps a lot. It also should give more accurate masks as the average spectrum has better signal-to-noise. Spectra with different LO settings will not be averaged, so this option does not work for spectral scans. [DEFAULT: `avermask=False`, i.e. determine mask on each individual spectrum]
- `subBase=True`, subtract the smooth baseline that was determined in the sine wave fitting process from the input data (i.e., in addition to subtracting the fitted sine waves). Note that this can be dangerous as the smooth baseline may include weak lines or wings of bright lines that were not properly masked. [DEFAULT: do not subtract the baseline]
- `flags=128`, data points with these flag values will be excluded from the calculations. Multiple flags are added, e.g., `flags=384` means that data points with flag values 128 and 256 will be excluded. Note that data with flag values 1, 2, 8, 64, and 1073741824 ( $=2^{**30}$ ) will ALWAYS be ignored. [DEFAULT: 128 (=SPUR\_CANDIDATE)]
- `resolution=10`, used to internally rebin the spectra in order to speed up the task. The default of 10 MHz is sufficient for HIFI spectra, but should be reduced for processing of non-HIFI spectra with standing wave periods less than about 30 MHz. [DEFAULT: 10 MHz]
- `addMedianContinuum=True`, allow the median continuum to be added back into the baseline fit flux. [DEFAULT: False]
- `saveInput=True`, will save the `fitHifiFringe` parameter inputs to a calibration product with a table dataset and metadata, and store this in the pipeline-out product, i.e. `obs.refs["calibration"].product.refs["pipeline-out"].product.refs["fitHifiFringe"].product`. This product can be generated only when the input `data` is an `ObservationContext`. The calibration product may be used in later applications of `fitHifiFringe` to fix the task parameters, using the `cal` input parameter. [DEFAULT: False]
- `cal`: An optional calibration product containing a `fitHifiFringe` product saved from a previous application of `fitHifiFringe` with `saveInput = True`. When a calibration product (e.g. `cal = obs.refs["calibration"].product`) is supplied by *drag and drop* into the GUI, or at the command line, the task parameters are fixed by the contents of the `fitHifiFringe` product (changes to task parameters in the GUI or command line will have no effect). If the calibration product lacks the `fitHifiFringe` product, a warning message will pop-up when using the GUI, prior to executing the task. If a calibration product is provided in a command line application of the task, and it is missing the `fitHifiFringe` product, only a warning message is given, the `cal` variable is then ignored, and the task will run with the default or current task parameter inputs. [DEFAULT: Null (no cal input)]

### 12.3.3. Example of using fitHifiFringe

In Level 2 data, standing waves are mixed with astronomical signals, and separating them from each other requires the astronomer's judgment. `fitHifiFringe` is an interactive user tool to do this, and a typical way of using it is described here.

To get started, the most important parameter to supply to `fitHifiFringe` is `typicalPeriod`. This sets the shape of the baseline for the standing waves. Its value depends on the HIFI band that is being processed. A reasonable default is automatically selected by `fitHifiFringe`. To optimise the value for a particular observation, you have to either inspect the plot before applying `fitHifiFringe`, or run `fitHifiFringe` once to get an estimate of the typical period(s). As an example, see the next plot. This is a Load Chop observation in band 3A taken without a sky reference. The typical period is about 100 MHz.



**Figure 12.4.** Example of a typical period (at about 100 MHz) in a Load Chop observation in band 3A (taken without a sky reference)

Subsequently, run the tool by clicking on *Accept*. Alternatively, you can enter the following command line:

```
result = fitHifiFringe(data=obs,product="WBS-H-USB",nfringes=1,typicalPeriod=100.0)
```

Here, only one sine wave is fitted to the data (`nfringes=1`). If one does not give this input parameter, `nfringes` is automatically adjusted depending on the HIFI band. Two plots are generated: one with the sine wave period ([Figure 12.5](#)) as a function of chi-square, and one with before/after spectra ([Figure 12.6](#)).

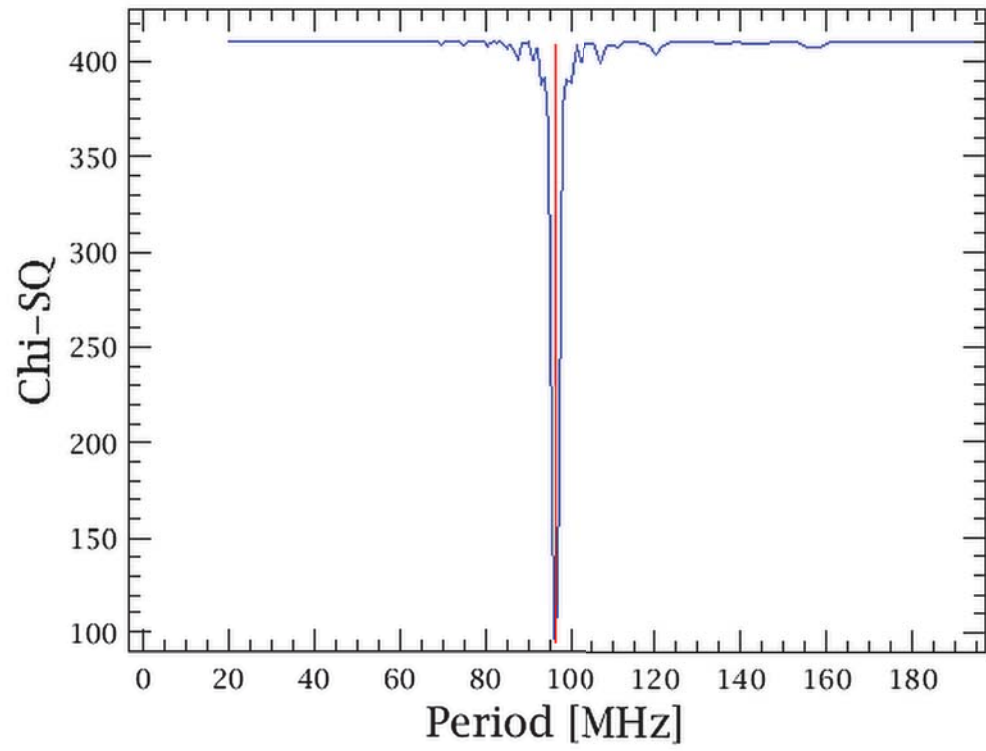
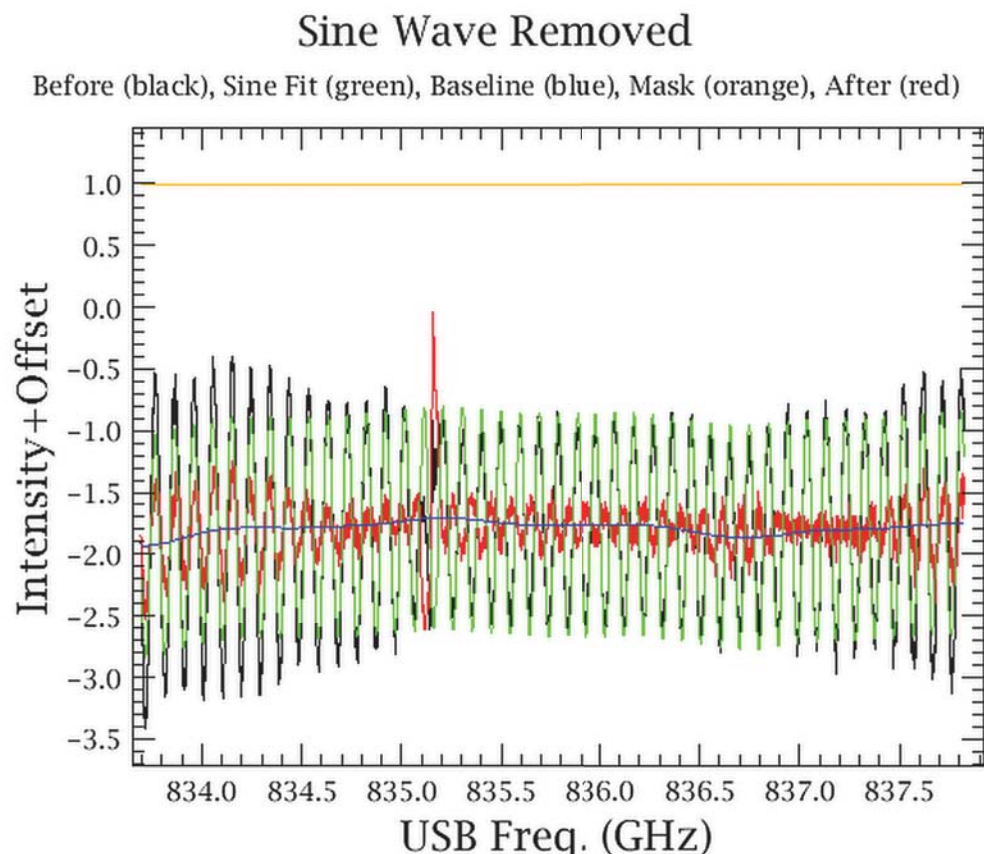


Figure 12.5. Sine wave fitted using only one sine wave (i.e. nfringes=1)



**Figure 12.6. Resulting fit showing the before and after spectra**

To optimise the fit, the following questions need to be answered by inspecting the plots:

- Does the smooth baseline (blue curve in the spectrum plot) look reasonable as a baseline for the standing waves? You may need to zoom in to see, do this by drawing a box around the region of interest with the left mouse button. In this example it looks OK. If it does not (e.g., if it follows the waves), run `fitHifiFringe` again with a different value of `typicalPeriod` (if the input was an `ObsContext`, you will need to read the original data from disk again).
- Are any lines properly masked? In this example, no lines are masked (the orange mask curve only shows values of 1), but there is clearly a spectral line between 835.04 and 835.24 GHz. You can mask this line by hand using the `usermask` input. Line masks are shown with values of 0.0 in the orange line, and channel flags taken into account with values of 0.5. Only channels where the orange line is 1.0 are taken into account in the calculations.
- How many peaks are present in the chi-square versus period plot? You may need to zoom in by drawing a box with the left mouse button. In this case, a strong 96 MHz standing wave is found (indicated by a vertical red line), but the spectrum shows, especially near the IF band edges, that the fit is not so good. The chi-square plot does not show additional peaks, but the main peak is rather broad. Increase the number of sine waves to be fitted (`nfringes`). Also, sometimes minima are found at periods that are irrelevant for the science, e.g., with very long periods. The fit solutions can be influenced by limiting the period search range better than the default 20-3000 MHz range. Use the `startPeriod` and `endPeriod` parameters for this. Do not choose a very narrow range, or otherwise the chi-square peaks cannot be determine accurately enough, or you might miss waves that are hard to see by eye in the spectrum.
- The smooth baseline (blue line in the spectrum plot) is only used as a baseline for the standing waves. It is not actually subtracted from the data. In some cases, such as in this example which



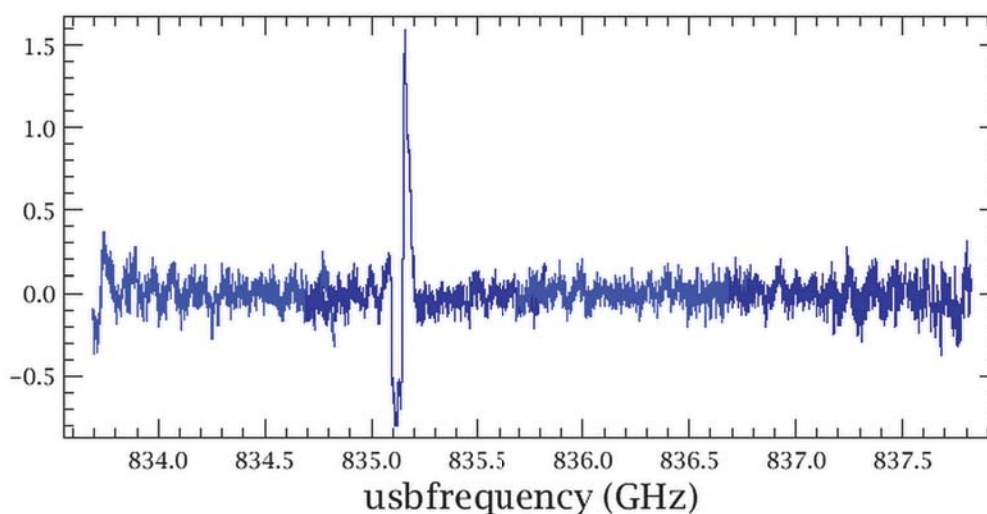
shows a negative baseline level, it is actually a very reasonable baseline choice. You can subtract it using the `subBase=True` option. Then you can omit fitting polynomial baselines at a later stage (using `fitBaseline`).

- By default the WBS subbands are fitted at once. Sometimes the fit improves by fitting them individually. In this case use the `doglue=False` option.

Taking the above considerations into account, the optimum `fitHifiFringe` parameters are:

```
result = fitHifiFringe(data=obs,product="WBS-H-
USB",nfringes=3,startPeriod=80.0,endPeriod=300.0,\
typicalPeriod=100.0,subBase=True,doglue=False,usermask=[(835.04, 835.24)])
```

The end result is shown here [Figure 12.7](#):



**Figure 12.7. Resulting fit using the optimum `fitHifiFringe` parameters**

If it is desired to save these input parameter values, the option `saveInput` can be used, and a `fitHifiFringe` product will be saved in the calibration tree of the observation context. In the same example:

```
result = fitHifiFringe(data=obs,product="WBS-H-USB",nfringes=3,startPeriod=80.0,
endPeriod=300.0, \
typicalPeriod=100.0,subBase=True,doglue=False,usermask=[(835.04, 835.24)],
saveInput=True)
```

Later, if the output product result (or `mynewObs = result[0]`) is saved, the same `fitHifiFringe` parameters can be retrieved from the data tree and reapplied, either to the same, or a different observation with similar standing wave characteristics:

```
myCal = obs.refs["calibration"].product
result2 = fitHifiFringe(data=obs2,cal=myCal)
```

Other considerations for removing standing waves with `fitHifiFringe`:

- The band 6 and 7 "HEB" waves are not pure sine waves: the phase drifts over the IF band pass. You may still approximate fits by using a combination of 2-3 sine waves with periods around 320 MHz.

For the WBS backends, fitting the waves per subband may give better results (*doglue=False*). The next [Section 12.4](#) deals with the Electrical Standing Wave correction in HEB bands.

- Waves in bands 3 and 4 show increasing amplitudes toward the IF band edges. Approximate fits can be obtained by fitting a combination of sine waves, and sometimes by fitting per WBS subband (*doglue=False*).

## 12.4. Electrical Standing Wave correction in HEB bands

Last updated: 15 September, 2015

Bands 6 and 7 have Hot Electron Bolometer (HEB) detectors, which produce strong electronic standing waves with characteristics that depend on the signal power. If the power on the source and reference position is not the same, due to sky signal strength or instrument instabilities, the Level 2 spectra will show residual waves. These waves have periods of ~320 MHz, but they are not pure sine waves. Applying the `fitHifiFringe` task is not always satisfactory for science analysis. Therefore a new technique is being developed that matches standing waves toward the target with a database of spectra at different power levels.

### 12.4.1. Introduction

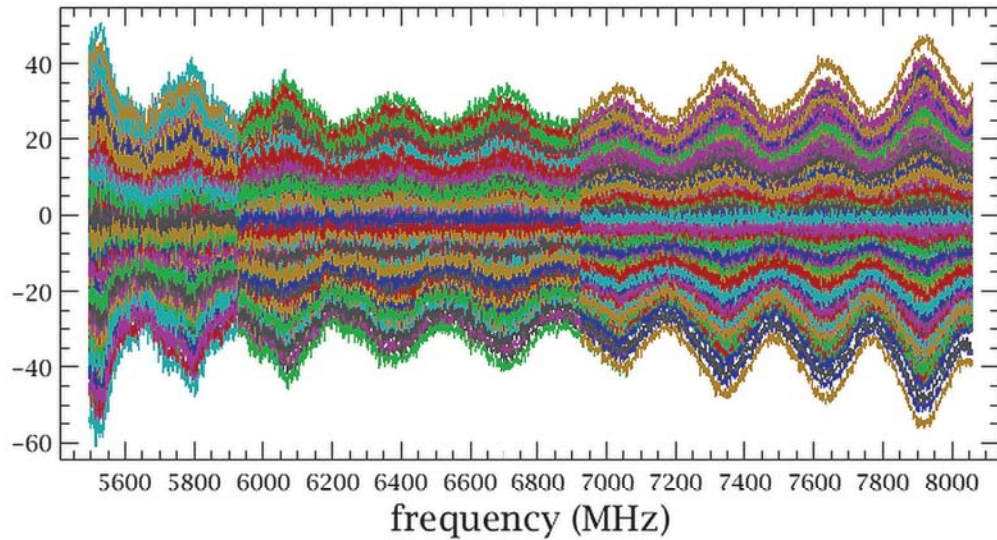
ESWs originate in the electric circuits of the Hot Electron Bolometer (HEB) bands 6 and 7. It was hoped that the ESW would be stable enough to be calibrated out with the usual off-source subtraction, which is also needed to remove the imprint of the telescope from the observation. However, the ESW was changing so quickly that there were residuals, sometimes quite significant, left in the spectra. Standard removal methods like FFT or sine fitting cannot work because the waves are not regular even in either amplitude or frequency.

A task was introduced in HCSS 12.0 to remove the troublesome electrical standing wave (ESW) affecting spectra in HIFI bands 6 and 7 (the HEB bands). Since then, solutions for all HEB observations have been prepared and placed in the HIFI calibration product, and in HCSS 13.0 are automatically applied by the pipeline. The correction improves most observations. We are following-up the relatively few cases that are not well-corrected and we will add the solutions in a future calibration release.

The pipeline task `doHebCorrection` applies the corrections stored in the HIFI calibration. It can also remove already-applied solutions. It does not compute the solutions. The task `HebCorrection`, which computes and applies the correction, has several useful features. The most important is the automatic disregard of line emission or absorption in the spectrum by means of robust fitting. Another is the option to plot individual *PointSpectra* for evaluation of the process. The task also creates a *TableDataset* which contains all the model parameters generated. However, before we go into these features, we will describe how the process of ESW removal is implemented.

### 12.4.2. Catalogue

In the database, there are very long stability calibration observations with numerous emission-free off-source spectra. Creating differences from many permutations of these spectra produces many example ESWs similar to those seen in normal data after on-off differencing, but before averaging. Thousands of example difference spectra were generated and collected in a catalogue. One such catalogue of ESWs is shown in [Figure 12.8](#).



**Figure 12.8. Example of a WBS HEB Catalogue**

From inspection of the figure, three properties are evidenced. Firstly, the ESWs are very similar indeed, except for a multiplicative factor. Secondly, there are not only waves in the ESW but also a continuum part. Thirdly, the continuum offset scales with the modulation amplitude, resulting in fact in both positive and negative offsets, accompanied by an apparent phase inversion in the modulation.

### 12.4.3. Spline Model

We want to know whether our notion of a multiplicative family is indeed true. The example ESWs were ordered according to their average absolute amplitude. We modelled the largest with a 72 knots cubic splines model. This relatively large number of knots was chosen as the smallest that would still catch all the finer details in the ESW. This largest one is our first splines model. The next example ESW was modelled in two ways: with a 72 knots cubic splines (like before) and as a multiplication of a previous splines model. If the latter is as good as the former, then this example ESW can be explained as one of a family. Otherwise, the splines model is added to the list, and it starts a new family of models. And so on, resulting in a list of example ESWs and splines models which are unique to the catalogue.

For all 8 HEB bands (6aH, 6aV, 6bH, 6bV, 7aH, 7aV, 7bH, and 7bV) we created catalogues and applied this procedure. The unique example ESWs were normalised and split into a slowly changing continuum part and a fast, waves-only part. The results are shown in [Figure 12.9](#) and [Figure 12.10](#) for all 8 bands. Note that the number of families is quite different among the bands.

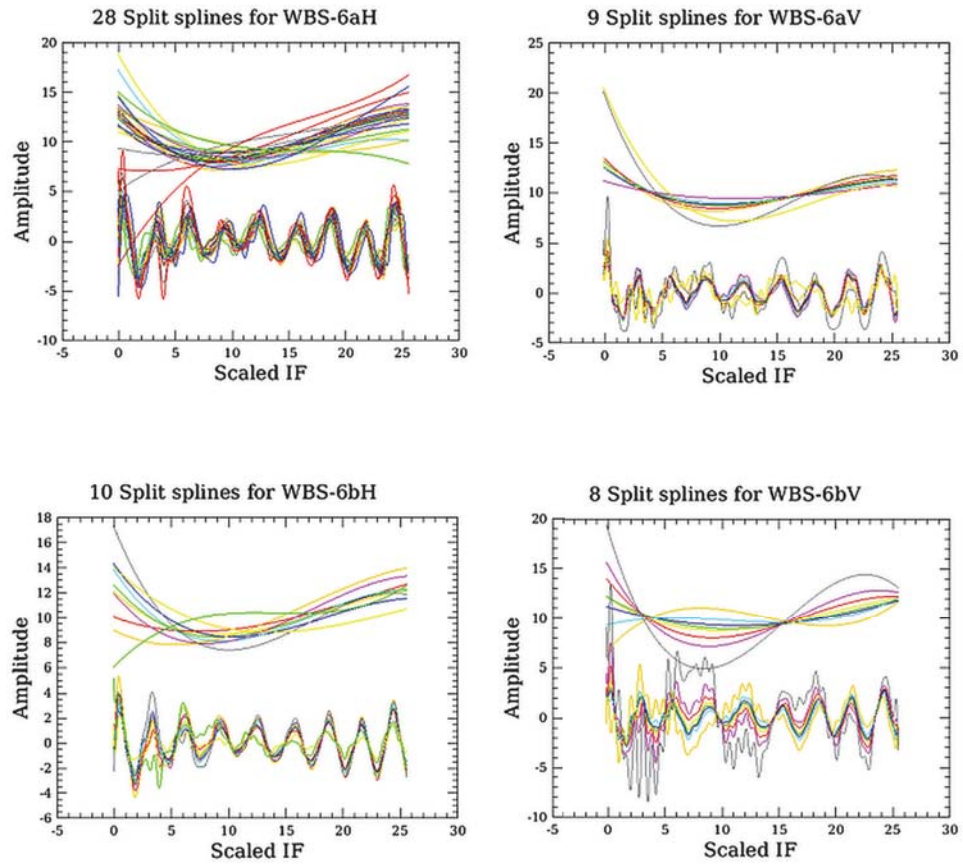


Figure 12.9. WBS Band 6a and 6 b (H and V)

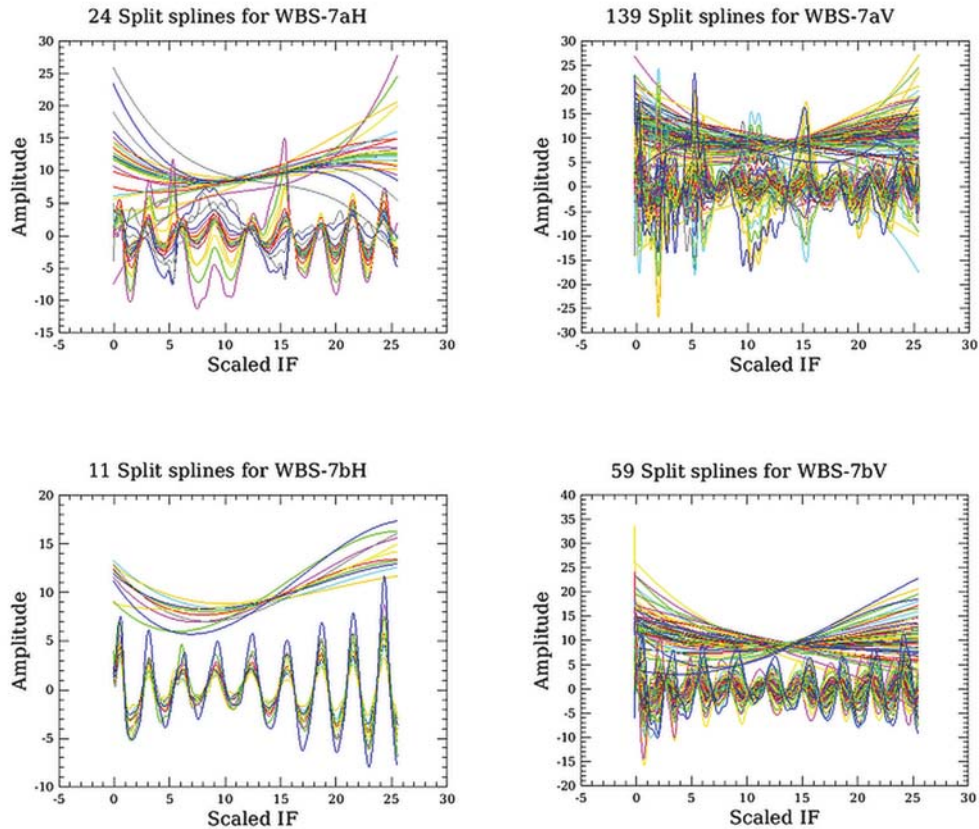


Figure 12.10. WBS Band 7a and 7 b (H and V)

## 12.4.4. Removal

The ESWs are a leftover effect of the on-off subtraction in the pipeline just before reaching Level 1. Most observations are aimed at some spectral line which might start to be visible in the individual spectra at Level 1. There might also be a continuum belonging to the source, and of course ESWs which must be removed without affecting the line or the continuum.

### Continuum only.

We start with a simple case where there are no (visible) lines present. In [Figure 12.11](#), there are 2 spectra, one with clear ESWs and one almost without. Note that the latter has a small continuum between 0 and 1 K.

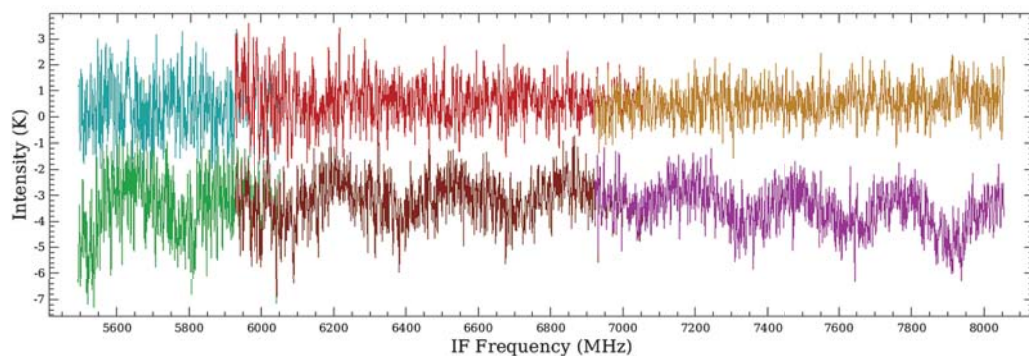


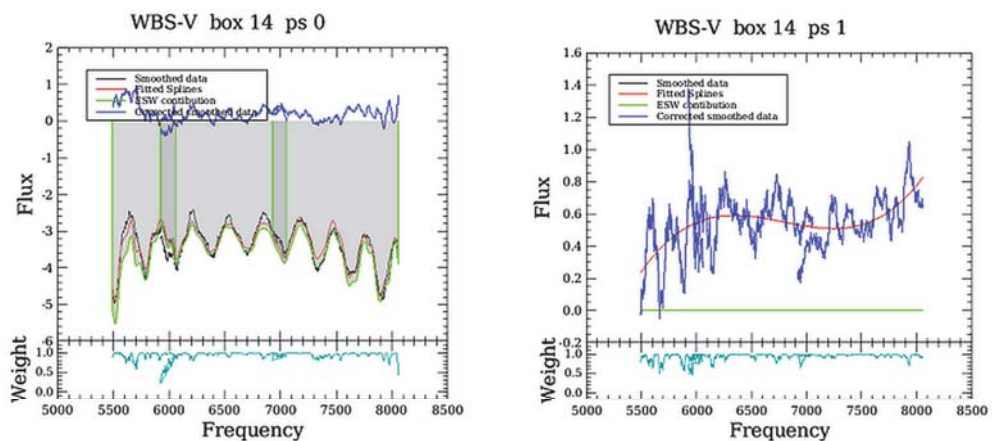
Figure 12.11. RDor Band 6a V-polarisation

Both spectra were corrected for ESWs. First we define a background model, another cubic splines with a few knots. This background model is added successively to a scaled version of each of the applicable fast model splines. The combination is fitted to the data and the one with the largest evidence is chosen as the best fit. No ESW at all is also a possibility when the evidence is largest for the background model alone. The scale found in this procedure is the amplitude of the ESW, both the slow and the fast parts. To correct the spectrum we subtract the slow and fast model splines multiplied by the scale factor.

In [Figure 12.12](#), the procedure is displayed. The spectrum data is in black. Note that it has been smoothed with a boxcar filter of 101 points to reduce the noise in the original. In red we find the best fit for the background and one of the fast splines models. In green we find the scaled sum of the selected fast model plus its associated slow model. Although here it looks almost the same as the red line, it is not. It can be quite different. The gray parts are removed from the spectrum, resulting in a corrected spectrum, displayed in blue. The lower part of each panel, labeled "weight", will be addressed in the next section.

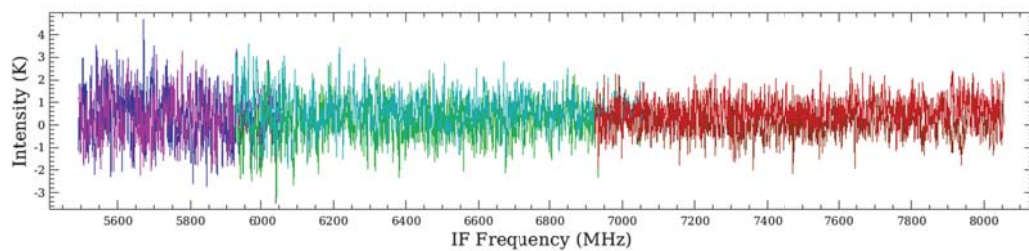
In [Figure 12.12](#) (right panel), we show a spectrum that does not show any ESW. The fit of the background alone had the largest evidence; hence it was chosen and no correction was applied. Black is blue.

Note that the corrected spectrum in [Figure 12.12](#) (left panel) lies slightly above zero. It is now much more like the spectrum without any ESW, i.e. [Figure 12.12](#) (right panel) which did not need any correction.



**Figure 12.12.** Example of a spectrum with ESW (left panel), and a spectrum with no ESW (right panel)

After correction, the 2 spectra fall nicely on top of each other ([Figure 12.13](#)).

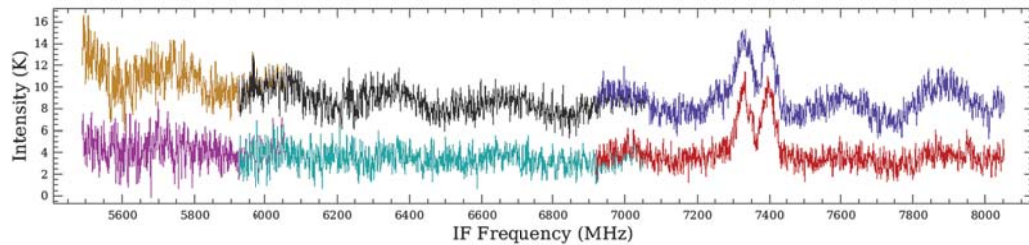


**Figure 12.13.** Results after correction

**Continuum with lines.**

Most observations are targeted on some line. The situation above should be more of an exception than the rule. Let's look at another observation where there is also line flux to consider.

Again we present 2 spectra, one with quite some ESW (Figure 12.14 upper spectrum) and one where it is doubtful (Figure 12.14 lower spectrum). In both spectra there is clearly a double-peaked line feature around 7400 MHz.

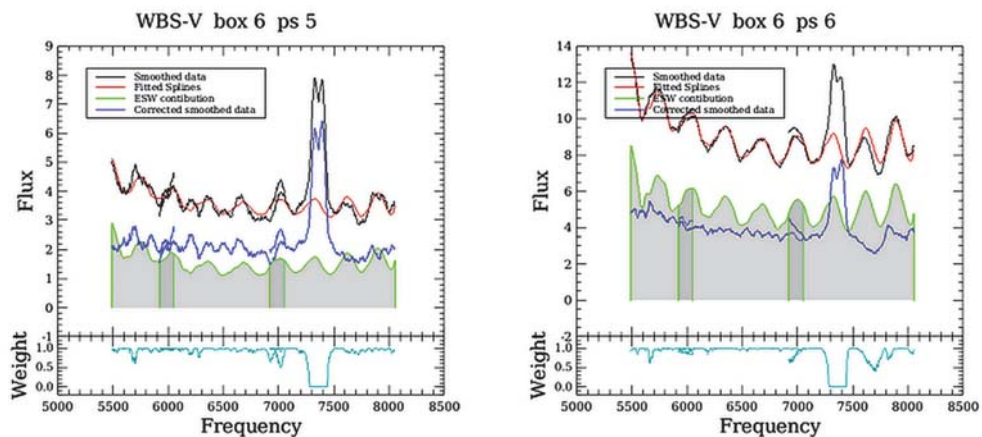


**Figure 12.14. Example of a spectrum with significant ESW (upper spectrum) and a spectrum with doubtful ESW (lower spectrum)**

There are two options to exclude the lines from the fit. One option is to mask regions manually using the `exclude_if` or `exclude_sky` parameters (pass IF frequencies using the former, or sky frequencies using the latter (either USB or LSB, the task automatically masks both sides)). The other option is to use [Robust Statistics](#) to automatically down-weight the lines. The latter option will be shown here. Robust statistics is an iterative scheme where outlying points are given less weight in the fit. Usually it converges after a handful of iterations.

Figure 12.15 shows the results for the two spectra. The layout is as before. Now it is clear that the lower part of each panel represents the resulting weights of the robust procedure. At most places it is near 1, only at the positions of the lines does it dip to 0, meaning that the lines are excluded from the fit.

It is now also clear why the boxcar smoothing was done before fitting: it makes all features stand out much more clearly. The ESWs in Figure 12.15 (left-panel) are now clearly defined as are the lines. In fact, without smoothing, the lines are not prominent enough to start the robust procedure. Unless smoothed, the lines "drown" in the noise.



**Figure 12.15. Results of the spectra going through the procedure**

After correction (Figure 12.16) the spectra are much more similar, although not completely. The wavy parts of ESW have completely disappeared, but the continuum is not completely corrected. H and V spectra can differ for reasons other than ESW, but large, relative baseline offsets are probably due to imperfect correction. Have a look at the baselines of all the corrected spectra and decide if the

variations look random (calibration uncertainty) or if there are outliers. A baseline offset relative to other integrations and a residual wavy pattern indicate poor ESW correction.

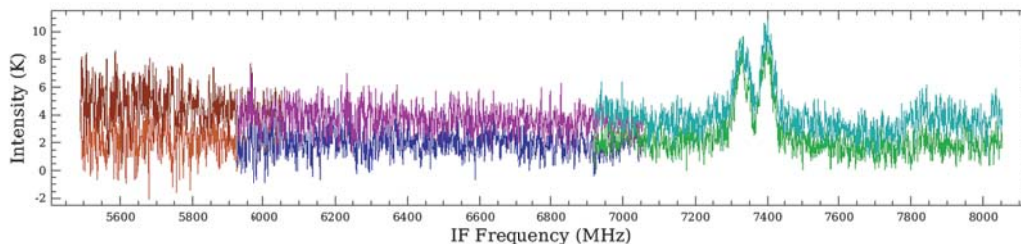


Figure 12.16. Results after correction

## 12.4.5. Demonstration

From HCSS 13.0 onward, pre-computed corrections stored in the HIFI calibration tree are automatically applied by the HIFI pipeline. Observations downloaded from the HSA processed with HCSS 13.0 or higher will most likely not require further electrical standing wave mitigation.

The Level 1 pipeline task `doHebCorrection` reads and applies the correction model parameters from the HIFI calibration tree. The same task can remove an already-applied correction. If you want to try re-running the ESW fit using the `hebCorrection` task as described below, you should first remove the corrections applied automatically by the pipeline.

```
# Example script to *remove* Electrical Standing Wave (ESW) corrections.
# This is necessary if you plan to try re-correcting the ESW
# yourself by masking spectral ranges, for example
#
# Read an observationContext

obsid = 1342219332
obs= getObservation(obsid,useHsa=1)

# undo the ESW correction

for backend in ['WBS-H', 'WBS-V', 'HRS-H', 'HRS-V']:
    http = obs.getLevel1().getProduct(backend)
    httpUnCorr, tds = doHebCorrection( http=http, catalog=obs.calibration,
    undo=obs.calibration)
    obs.level["level1"].setProduct(backend,httpUnCorr)

# metadatum 'hebCorrection' is set to True if data are ESW-corrected

print obs.refs["level1"].product.refs["WBS-
H"].product.refs["box_001"].product["0005"].meta['hebCorrection']

# {description="Heb Standing Wave Correction has been done", boolean=false}

# If you want to recompute Level 2 and higher products, run the pipeline as below.
# But note, this is not necessary if you want to now attempt to re-fit the ESW
# with the task hebCorrection, because it works by default on Level 1 data

obs = hifiPipeline( obs=obs, fromLevel=1.0 )
```

The `hebCorrection` task will attempt to fit models of Electrical Standing Waves (ESW) to the WBS science spectra at Level 1, and then produce corrected spectra. The HRS data, if present, will be corrected using the solutions from WBS. The ESW models are stored in the HIFI *calTree*.

If you are dissatisfied with the ESW correction, you might improve the result by masking a spectral range, if there is an unruly baseline or a spectral line of width/amplitude similar to an ESW. Excluding a spectral range or ranges is the most effective parameter affecting the ESW fit that you can tune.



The `hebCorrection` task corrects the `observationContext` you submit as parameter `obs` in-place; the data will be changed. In addition, the output of the `hebCorrection` task is a `TableDataset` that contains all the parameters used in the correction. It is recommended to save this table, as it can be used to perform the correction again without redoing the CPU-intensive fitting. Another available output is a list of plots showing the spectra pre- and post-correction, and the models fitted. The form of the plots depends on the parameter `plot`. See the example script below for usage.

- `plot = 0` no plots (default)
- `plot = 1` erase previous plot when new one appears
- `plot = 2` make plotlist
- `plot = 3` every plot in its own window. (This might generate too many plot windows for your windowing system. X11 has been known to crash)

The option to mask out spectral ranges is `exclude_sky`. The parameter value should be a `DoubleId` of even length containing `[lowF1,highF1,lowF2,highF2,...]` values of frequency ranges to exclude during fitting. This option takes sky frequency ranges (either USB or LSB allowed, the task automatically masks both sidebands). Robust fitting is still used by default, though if the unmasked baseline regions are clean you can consider shortening `runtime` by turning it off with the robust option set to `False`.

Alternatively, you can specify the mask region on the Intermediate Frequency (IF) scale using the task parameter `exclude_if`. The difference with `exclude_sky` is only in spectral scans: `excluding_if` will affect all spectra, whereas `exclude_sky` will only affect those with the appropriate LO to bring those frequencies into the IF.

```
# Read an observationContext

obsid = 1342196583
obs = getObservation( obsid, useHsa=1, useCache=0 )

# undo the ESW correction

for backend in ['WBS-H', 'WBS-V', 'HRS-H', 'HRS-V']:
    http = obs.getLevel1().getProduct(backend)
    httpUnCorr, tds = doHebCorrection( http=http, catalog=obs.calibration,
    undo=obs.calibration)
    obs.level["level1"].setProduct(backend,httpUnCorr)

# If, somehow, you are lacking the ESW model set in the calTree, uncomment this
# following line
# obs.refs["calibration"].product.refs["Downlink"].product.refs["Generic"]. \
# product.refs["hebCorrection"].product.refs["WBS-Splines"].product

# you should download the observation again from the HSA!
#
# Deprecated Alternative: the model set is available online:
# wget http://herschel.esac.esa.int/twiki/pub/Public/HifiCalibrationWeb/
# hebCorrectionModels_HICAL15.fits
# It can be read into HIPE and passed to the task as a parameter:
# catalog = simpleFitsReader( '/path/to/hebCorrectionModels_HICAL15.fits' )
# tds = hct( obs=obs, offsource=True, catalog=catalog, plot=2 )

# Process (remove ESWs from) the observation with default input parameters
# Data are corrected at Level 1

hct = hebCorrection
tds = hct( obs=obs, offsource=True, plot=2 )

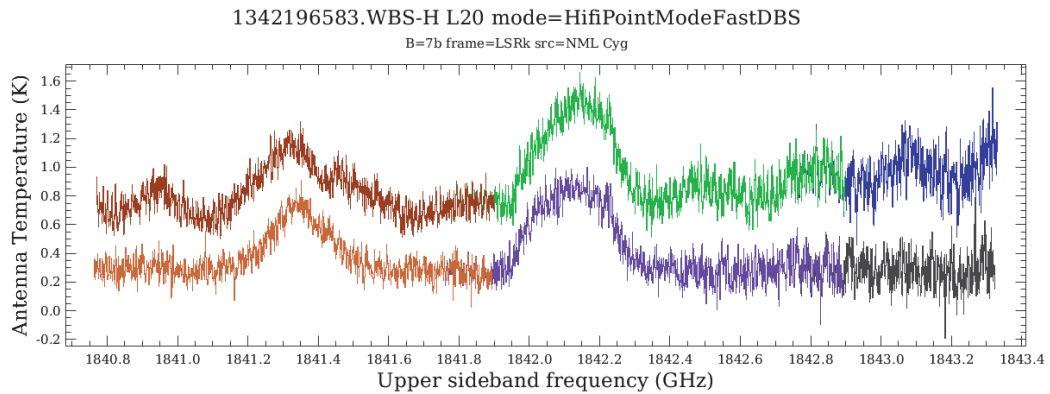
# Have a look at the plots if interested

plotList = hct.plotList

# Recreate the final pipeline products if you are satisfied with the correction at
# Level 1
```

```
obs = hifiPipeline( obs=obs, fromLevel=1.0 )
#Done!
```

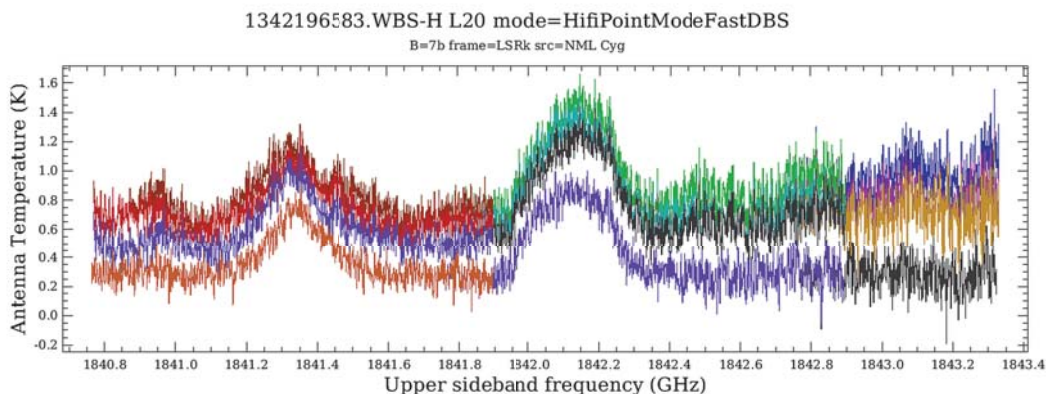
The example script demonstrates the use of `HebCorrectionTask` on the relatively short observation 1342196583. There are two broad lines in the spectrum, see [Figure 12.17](#) for the uncorrected pipeline output.



**Figure 12.17. Uncorrected pipeline output**

Observation 1342196583 is an example of an observation that cannot be satisfactorily corrected with the current version of the software; the fraction of such observations is not large, and we are improving our models based on these exceptions.

After running the script above, the final spectra are as shown in [Figure 12.18](#). In the figure, the flat-test spectrum is the original, uncorrected WBS-H spectrum. The corrected H spectra (not shown) are almost identical. The three spectra above are all WBS-V, in decreasing order of baseline level: (a) uncorrected, (b) corrected with default input parameters, (c) corrected with manual excision of the lines by means of the `exclude` parameter. Final spectra after running the script:



**Figure 12.18. Final spectra after running the script**

```
obs2 = getObservation( obsid, useHsa=1, useCache=0 )

# undo the ESW correction

for backend in ['WBS-H', 'WBS-V', 'HRS-H', 'HRS-V']:
    htp = obs2.getLevel1().getProduct(backend)
    htpUnCorr, tds = doHebCorrection( htp=htp, catalog=obs2.calibration,
    undo=obs2.calibration)
    obs2.level["level1"].setProduct(backend,htpUnCorr)

hct2 = hebCorrection
```

```
tds2 = hct2( obs=obs2, offsource=True, plot=2, exclude_if=DoubleId([6461., 6941.,
7241., 7741.] ) )
```

Note that the frequency ranges are specified on the IF scale in MHz. The relation between IF frequencies and sky frequencies, F, for bands 6 and 7 are:

LSB:  $IF = F - LO + 10.4$  (all in GHz)

USB:  $IF = LO - F + 10.4$  (all in GHz)

Using the manual line excision results in a different set of models being matched to the data. This can be seen in [Figure 12.19](#) and [Figure 12.20](#), the output tables containing details of the fitted models. The column titled *callIndex\_WBS-V* (or *callIndex\_WBS-H*) lists the best-fitting model found for the Level 1 spectrum; a value of -1 indicates no model was found preferable over a null correction. In the default inputs case, a correction to every WBS-V spectrum was performed, but in the manual line excision run, only 7 of the 24 Level 1 spectra were corrected.

index	box_WBS-V	psindex_WBS-V	callIndex_WBS-V	evidence	param_WBS-V	backg_WBS-V	box_WBS-H	psindex_WBS-H	callIndex_WBS-H	evidence	param_WBS-H	backg_WBS-H
0	5	0	20	-1869.7580	[-0.0378752742	[-0.29342434328	5	0	-1	-1864.4099	[0.0]	[0.3316740776
1	8	0	27	-1880.0068	[0.0277021059	[0.791770531802	8	0	-1	-1874.9861	[0.0]	[0.2352474456
2	9	0	23	-1868.7826	[-0.0161790210	[-0.26891328015	9	0	4	-1869.1537	[-0.00726010	[-0.023711912
3	15	0	41	-1790.1166	[0.0053187139	[0.566499597925	15	0	1	-1871.6008	[0.019077059	[0.2616628683
4	16	0	8	-1890.0189	[0.0617075460	[2.306060901178	16	0	8	-1869.4667	[0.018558016	[0.353137795
5	22	0	56	-1864.0686	[0.0153633466	[0.752339047696	22	0	-1	-1873.6407	[0.0]	[-0.142583373
6	23	0	58	-1874.3804	[-0.0088426933	[0.086597271702	23	0	4	-1869.8962	[-0.02827188	[-0.226909444
7	26	0	34	-1861.4924	[-0.0530123128	[-0.57628345058	26	0	8	-1872.5973	[0.011984038	[0.5415369620
8	27	0	30	-1865.4355	[-0.0185141937	[-0.02830839821	27	0	-1	-1881.9197	[0.0]	[0.0131187661
9	33	0	42	-1880.3823	[0.0125250900	[0.521151423409	33	0	-1	-1871.9337	[0.0]	[0.0451651754
10	34	0	13	-1883.8316	[-0.0193222684	[-0.05295139518	34	0	-1	-1879.2723	[0.0]	[-0.119601084
11	40	0	14	-1889.1109	[0.0661705381	[1.792068315248	40	0	8	-1880.0607	[0.024388831	[0.7491796352
12	41	0	56	-1839.6312	[0.0161499537	[0.431305064027	41	0	4	-1878.6916	[-0.02045194	[-0.254923800
13	44	0	53	-1857.5676	[0.0046556233	[0.632725813832	44	0	9	-1869.1954	[-0.00838303	[-0.184256483
14	45	0	19	-1863.4053	[-0.0173687662	[0.150734089616	45	0	7	-1876.9221	[-0.01318420	[-0.035871499
15	51	0	30	-1876.1370	[0.0239427619	[0.865534120273	51	0	9	-1862.0438	[0.006293988	[0.2750104682
16	52	0	44	-1871.7859	[-0.0044390678	[0.042743860675	52	0	8	-1864.7873	[-0.01307120	[0.2580614243
17	58	0	30	-1870.4798	[-0.0520422663	[-0.75221869651	58	0	8	-1872.8908	[-0.00738462	[-0.129564584
18	59	0	51	-1879.7287	[0.0173481289	[0.506597255470	59	0	-1	-1880.6436	[-0.01641364	[-0.230518631
19	62	0	56	-1876.3691	[0.0148813671	[0.33022245565	62	0	-1	-1876.3709	[0.0]	[-0.035397087
20	63	0	32	-1882.1232	[0.03288902491	[1.006571416924	63	0	-1	-1878.9627	[0.0]	[-0.4999780490
21	69	0	46	-1864.9530	[0.0163815683	[0.976664677144	69	0	8	-1861.6454	[0.0]	[0.2047522593
22	70	0	41	-1872.9733	[0.0128821813	[0.651448214894	70	0	-1	-1880.3893	[0.019768795	[0.4208772803
23	76	0	46	-1881.6319	[0.0361380656	[1.448308765049	76	0	-1	-1857.3669	[0.0]	[0.2289854881

Figure 12.19. HebCorrection output table with default input parameters

index	box_WBS-V	psindex_WBS-V	callIndex_WBS-V	evidence	param_WBS-V	backg_WBS-V	box_WBS-H	psindex_WBS-H	callIndex_WBS-H	evidence	param_WBS-H	backg_WBS-H
0	5	0	-1	-1303.1828	[0.0]	[-0.26567054	5	0	-1	-1304.1696	[0.0]	[0.337463315
1	8	0	-1	-1302.7661	[0.0]	[0.77986984	8	0	-1	-1305.7221	[0.0]	[0.232861822
2	9	0	-1	-1299.7983	[0.0]	[-0.18517763	9	0	-1	-1303.3227	[0.0]	[-0.016411352
3	15	0	-1	-1299.7796	[0.0]	[0.56142426	15	0	-1	-1305.5208	[0.0]	[0.256808206
4	16	0	30	-1304.4275	[0.0194457711	[2.19612952	16	0	-1	-1307.2157	[0.0]	[0.355731540
5	22	0	-1	-1302.3244	[0.0]	[0.73887230	22	0	-1	-1303.3917	[0.0]	[-0.124664663
6	23	0	-1	-1301.4193	[0.0]	[0.08610500	23	0	-1	-1306.3999	[0.0]	[-0.230080486
7	26	0	34	-1302.2479	[-0.045647417	[-0.56013460	26	0	-1	-1305.0801	[0.0]	[0.539216046
8	27	0	-1	-1298.7690	[0.0]	[-0.01581113	27	0	-1	-1304.3931	[0.0]	[0.023787993
9	33	0	-1	-1300.9862	[0.0]	[0.52895682	33	0	-1	-1303.9168	[0.0]	[0.070707445
10	34	0	-1	-1301.2542	[0.0]	[-0.04473555	34	0	8	-1304.1696	[0.0]	[-0.093179549
11	40	0	14	-1303.5276	[0.0174699061	[1.80982316	40	0	-1	-1309.1698	[0.02788649	[0.764283415
12	41	0	-1	-1303.0479	[0.0]	[0.41550775	41	0	-1	-1305.4438	[0.0]	[-0.248794982
13	44	0	-1	-1299.7349	[0.0]	[0.63299633	44	0	-1	-1304.3512	[0.0]	[-0.171449456
14	45	0	-1	-1299.9689	[0.0]	[0.16049321	45	0	-1	-1305.3607	[0.0]	[-0.017947330
15	51	0	-1	-1301.3614	[0.0]	[0.85127021	51	0	-1	-1303.9955	[0.0]	[0.276870502
16	52	0	-1	-1298.4800	[0.0]	[0.04765621	52	0	-1	-1304.2953	[0.0]	[0.267269718
17	58	0	31	-1302.9450	[-0.046317055	[-0.73153440	58	0	-1	-1304.0772	[0.0]	[-0.111890788
18	59	0	-1	-1301.4724	[0.0]	[0.50581301	59	0	-1	-1304.4342	[0.0]	[-0.224777526
19	62	0	-1	-1303.6512	[0.0]	[0.32274945	62	0	-1	-1305.1787	[0.0]	[-0.020633791
20	63	0	18	-1303.1313	[0.0049377055	[1.02243382	63	0	-1	-1305.8030	[0.0]	[0.519973890
21	69	0	-1	-1301.7410	[0.0]	[0.97116455	69	0	-1	-1303.9574	[0.0]	[0.204767142
22	70	0	9	-1303.5046	[0.033622069	[0.65879324	70	0	8	-1308.3369	[0.02412346	[0.433415507
23	76	0	31	-1304.0171	[0.077877932	[1.45595602	76	0	-1	-1303.4602	[0.0]	[0.237570014

Figure 12.20. HebCorrect output table with exclude parameter

In fact, for this observation and the current model set, there is little to choose between the solutions for default and manual-line-excision. We do not recommend turning off robust fitting. Although the task will run faster, your observation will not be as well served.

# Chapter 13. HIFI Baseline Removal

Last updated: 29 February, 2016.

## 13.1. Introduction

HIFI spectra produced by the Level 2 pipeline often show baselines that are offset, sloped, curved, and/or rippled. In many cases, these are leftovers from instrumental instabilities inherent to heterodyne techniques, and not completely removed by reference or sky subtraction in the HIFI pipeline. Depending on the science application, you may want to flatten the baseline, either by subtraction, or, in the presence of real continuum emission, by division. This can be done in HIPE at the Level 2 stage, using the tasks described in the next sections.

## 13.2. The FitHifiFringe Task

Ripples (also known as standing waves or fringes) can be fitted with sine waves and subsequently removed using the `FitHifiFringe` task, which is described in [Chapter 12](#). In the sine wave fitting process, a line clipping and smoothing algorithm produces a smooth version of the baseline. In addition to subtracting the fitted sine waves, you may choose to subtract this smooth baseline, by setting the `sub_base` option to `True` in `FitHifiFringe`. This is an efficient way to subtract baselines, but it should be used with care as the smooth baseline shape could be affected by weak lines or wings of broad lines that are not properly automatically masked. In such cases, you may first remove the ripples using `FitHifiFringe`, and then use the `FitBaseline` task described in the next section to mask lines manually, and fit the remaining baseline offsets and slopes using polynomials.

## 13.3. The FitBaseline Task

### 13.3.1. Introduction to FitBaseline

`fitBaseline` is a task that allows you to interactively fit, and subtract or divide out polynomial baselines in HIFI spectra. The task loops over all spectra (individual or 'glued' WBS subbands or HRS modules) available in HIFI ObservationContexts, TimelineProducts or SpectrumDataSets, and asks you to mask as many emission lines as needed before fitting the polynomial. You can select different polynomial orders for each spectrum, if needed. The line masks are stored in a table that can be used if you want to redo the baseline fitting at a later time, e.g., when the observation has been re-processed by a new version of the pipeline. At that time, you can un-mask lines, or change the polynomial order. Also, all fitted polynomial coefficients, as well as all original and baseline subtracted spectra, are stored so that you can review the fits after the fact. Finally, `fitBaseline` also allows you to replace the 'theoretical' channel weights assigned by the HIFI pipeline with actual weights determined from the channel-to-channel RMS noise after rectifying the baseline with the polynomial fit.

When `subBase=1` and `basemode=sub` are selected, `fitBaseline` subtracts the fitted baseline (shape+continuum level). At the same time, a median baseline level is computed (per spectrometer subband, or as a whole, depending on the input on `doglue`) and reported in the output table `fitProduct/fitTable` for each spectrometer subband. The parameter `addMedianContinuum` allows you to re-add the median baseline level computed by the task in the baseline subtraction process. This will allow you to benefit from baseline correction (shape) but preserve the overall continuum. **Note** that this only performs well for `basemode='sub'` and not necessarily for `basemode='div'`. For the latter, the baseline-corrected spectrum is normalised to 1, and with `addMedianContinuum=True`, the corrected spectrum is multiplied by the median baseline level, to recover the original continuum level.

## 13.3.2. Running FitBaseline

By default, `fitBaseline` is referenced to `ObservationContexts`, i.e., it will show up as an applicable task when you click on an `ObservationContext` variable. Double-click on `fitBaseline`, and the following GUI (Figure 13.1) will show up:

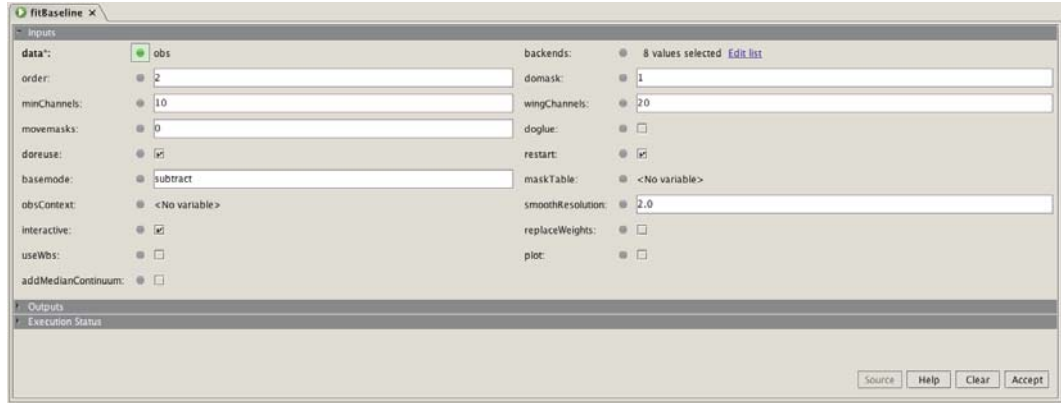


Figure 13.1. FitBaseline GUI

You may also run the task on the command line:

```
fitBaseline (data=obs_in)
```

The original spectra in the `obsContext` `obs_in` are always overwritten by the baseline-subtracted spectra, even if the task is called as:

```
obs_out=fitBaseline (data=obs_in)
```

For HIFI TimelineProducts, and WBS or HRS SpectrumDataSets, the input data are preserved and output variables have to be defined:

```
htp_out=fitBaseline (data=htp_in)
```

```
sds_out=fitBaseline (data=sds_in)
```

You will always see 2 plots for each baseline that needs to be fitted (see Figure 13.2).

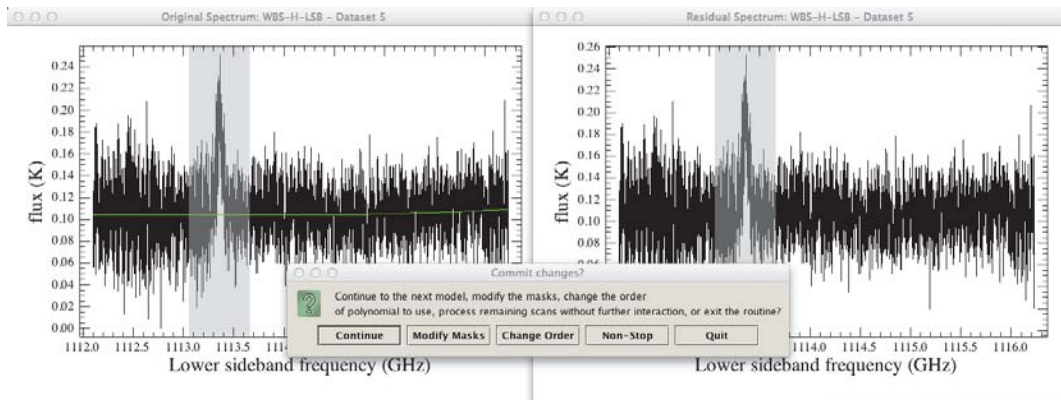


Figure 13.2. Original and Residual plots produced by `fitBaseline`

In `interactive=True` mode, the left plot is used to define masks by dragging (left-to-right, or right-to-left) over regions to be excluded, using left-click. The mask is then visible as a curtain. Dragging

back the curtain will remove the mask. When you're done with defining masks, click on the plot window, but outside of the plot axes. The green line is then the fitted polynomial, and the right plot displays the baseline subtracted spectrum, with the used masks in grey. You can zoom in on the plot to assess the baseline fit in the usual PlotXY way - draw a box with the left mouse button around the area you want to zoom in on. A selection window will pop up, in which you can opt to change the order of the polynomial, or add or remove the masks. When you click on 'Continue', the next dataset will be selected. 'Quit' will save the results so far. If the option `restart=False`, when you run `fitBaseline` the next time, `fitBaseline` will start from where you left off before. You can also select the option to continue with no further interaction with the option `non-stop`.

**Please note** that if the data contain SPUR\_WARNING flags, the mask will be displayed (usually in pink) but this flag is **not** being honoured because this is, as the flag name indicates, just a warning that the data may suffer from a spur. If you feel that the data must be flagged after all, you must flag the data yourself by using the flag IGNORE\_DATA. The display of the SPUR\_WARNING flag is just indicative.

In [Figure 13.2](#) we have also invoked `addMedianContinuum=True` to demonstrate this capability. The sequence of commands that we have used in this example is:

```
obs_out = fitBaseline(data=obs, backends=['WBS-H-LSB'], doglue=True,
    addMedianContinuum=True)
```

When `fitBaseline` is finished, and an `ObservationContext` was selected as input, the new item `fitProduct` will have been added, as can be seen in [Figure 13.3](#). It contains, for each dataset, the original, the baseline subtracted, the polynomial fit, and the mask spectra, which can all be inspected using the Spectrum Explorer. In addition, under the Level 2 product list, a `Linemasks` table has been created, as indicated in [Figure 13.3](#). This table contains all information about the applied masks. If the `origin` column has an entry of '-1', it means that this mask has been disabled. If it is '2' it means that the mask was determined automatically, using the `domask=2` option.

Index	freq_1	freq_2	weight	origin	peak	median	dataset	scan
0	1108.065...	1108.355...	0.0	2.0	15.09211...	-0.02192...	1.0	0.0
1	1108.375...	1108.635...	0.0	2.0	0.418335...	-0.02096...	1.0	0.0
2	1108.460...	1108.780...	0.0	2.0	0.602091...	-0.03450...	1.0	0.0
3	1109.065...	1109.325...	0.0	2.0	0.344165...	-0.04434...	1.0	0.0
4	1109.15	1109.41	0.0	2.0	0.295889...	-0.03484...	1.0	0.0
5	1110.440...	1110.700...	0.0	2.0	0.274137...	-0.04776...	1.0	0.0
6	1110.745...	1111.025	0.0	2.0	0.310046...	-0.07519...	1.0	0.0
7	1109.392	1109.702...	0.0	2.0	0.570176...	0.106969...	2.0	0.0
8	1109.587...	1109.877...	0.0	2.0	0.441724...	0.058142...	2.0	0.0
9	1110.090...	1110.25	0.0	2.0	0.319975...	-0.05301...	3.0	0.0
10	1113.195	1113.52	0.0	2.0	0.821896...	0.042120...	3.0	0.0
11	1111.107...	1111.297	0.0	2.0	0.409716...	0.050375...	4.0	0.0
12	1113.197...	1113.527	0.0	2.0	0.741245...	0.101865...	4.0	0.0
13	1114.167...	1114.427...	0.0	2.0	0.340295...	0.089456...	4.0	0.0
14	1112.756	1113.016	0.0	2.0	0.243310...	-0.05067...	5.0	0.0
15	1113.136...	1113.571...	0.0	2.0	0.642654...	-0.01191...	5.0	0.0
16	1113.391	1113.651	0.0	2.0	0.384915...	-0.03998...	5.0	0.0
17	1113.096	1113.636	0.0	2.0	0.811122	0.029594	6.0	0.0

Figure 13.3. Example of a Linemasks table

### 13.3.3. Re-running FitBaseline

Masking the emission or absorption lines in an observation can be tedious and time consuming, in particular for spectral scans. If you need to subtract the baselines again for a given observation (e.g., after pipeline reprocessing), the line masking does not need to be repeated, provided `fitBaseline`

was applied to an `ObservationContext`. The previously created `Linemasks` table can be re-used. There are two ways of doing this:

1. By storing the `Linemasks` table as a fits file to disk, and reusing it. The sequence of commands is typically this:

```
fitBaseline(data=obs1)
linemask=obs1.refs["level2"].product["Linemasks"]
# optional: simpleFitsWriter(product=linemask, file='linemask.fits')
# optional: linemask=simpleFitsReader(file='linemask.fits')
fitBaseline(data=obs2, maskTable=linemask)
```

2. By providing the 'old' `ObservationContext`, and reusing its `Linemasks` table and `FitProduct`. The sequence of commands is typically this:

```
fitBaseline(data=obs1)
# optional: localStoreWriter(product=obs1, store="obs1store")
# optional: obs1=getObservation(obsid1, poolName="obs1store")
# optional: obs2=getObservation(obsid2, poolName="obs2store")
fitBaseline(data=obs2, obsContext=obs1)
```

The advantage of option 2 is that the previously used polynomial order is retrieved from the `FitProduct`, and applied to the other observation. For option 1, you have to re-enter the polynomial orders.

### 13.3.4. FitBaseline Options

`FitBaseline` has a large number of user options, which are described here:

- `data`: input can be `ObservationContexts` (Level 1 or 2), `HifiTimelineProducts` (Level 1 or 2), `SpectrumDatasets` (WBS or HRS), `SimpleSpectrums` (Waveunit needs to be in GHz), and spectral cubes (`SpectralSimpleCubes`) (Waveunit needs to be in GHz).
- `backends=..`: Backends to process. Subset of default list ['WBS-H-LSB', 'WBS-H-USB', 'WBS-V-LSB', 'WBS-V-USB', 'HRS-H-LSB', 'HRS-H-USB', 'HRS-V-LSB', 'HRS-V-USB']. On the command line more options are allowed: 'WBS-V', 'WBS-H', etc., which imply that both the LSB and USB products of a given back will be processed.
- `order=..`: order of polynomial to fit, [DEFAULT: 2]
- `domask=..`
  - 0: for no masking
  - 1: for manual masking [DEFAULT]--ignored in interactive=False mode
  - 2: for automatic masking
- `movemasks=..`
  - 0: masks are only used for spectrum they are defined in [DEFAULT]
  - 1: move masks between spectra of same sideband
  - 2: move masks between spectra and sidebands by determining which sideband the masked line is in
- `doreuse = False`: Do not use masks specified previously for the LSB datasets for the equivalent USB datasets [DEFAULT: True]
- `doglue = True`: Fit all subbands in a spectrum simultaneously. Warning: if the WBS subbands are offset with respect to each other this option should not be used. [DEFAULT: False]



- *restart* = False: continue fitting baselines where you left off previously [DEFAULT: True, so start from beginning]
- *basemode* = ..
  - 'sub': subtract fitted baseline from original spectrum (resulting continuum is 0) [DEFAULT]
  - 'div': divide original spectrum by fitted baseline (so resulting continuum is 1)
- *addMedianContinuum* = True: allow the median continuum to be added back into the baseline fit flux; this only performs well for *basemode* = 'sub' and not *basemode* = 'div' [DEFAULT: False]
- *maskTable* = ...: MaskTable product containing previously-determined masks [DEFAULT: use maskTable found in ObsContext]
- *obsContext* = ...: ObservationContext from which the maskTable will be read, and in which the maskTable and fitProduct will be saved. This is useful if you provide a HifitimelineProduct or SpectrumDataSet as 'obs' input, because the maskTable and fitProduct can only be stored in ObservationContexts. [DEFAULT: -if obs is ObservationContext: read/write maskTable and FitProduct from/to obs -if obs is HTP or SDS: no maskTables or FitProducts are read or written]
- *smoothResolution* = ...: Only applies if domask=2 has been set. Resamples the data to resolution in MHz, which makes the automated masking faster. The drawback is that larger values lead to masking of noisy peaks rather than real lines. [DEFAULT: 2 MHz]
- *interactive* = False: With this option, the task will run without any user interaction. Line masks are taken from any existing mask table (in the ObsContext or user-supplied maskTable or separate ObsContext) and previously defined line flags. No new line masks can be defined, and thus the user parameter 'domask' is ignored. [DEFAULT: True, i.e. do stop after each spectrum, but the default is False when doreuse=True]
- *useWbs* = True: With this option, the task will subtract polynomials already fitted to the WBS spectra (stored in the fitProduct of the obsContext) from corresponding HRS spectra. This is particularly useful if there are broad emission or absorption lines that fill most of the HRS band width. [DEFAULT: False, i.e., fit polynomials for HRS separately]
- *minChannels* = ...: When automated masking is enabled (domask=2), ignore masks that consist of less channels than minChannels [DEFAULT: minChannels=10]
- *wingChannels* = ...: When automated masking is enabled (domask=2), extend the mask on either side by wingChannels channels to take into account weak wings [DEFAULT: wingChannels=20]
- *replace\_weights*: When set to True, the weight value for each channel is replaced by  $1/\text{rms}^2$ , with rms measured within a segment excluding the masked lines and after the baseline has been subtracted. The user can verify whether the weights in a SpectrumDataset were replaced by checking the metadata keyword 'replace\_weights'. These new, empirical weights may improve the deconvolution process for spectral scans. [Default: False]
- *plot*: Force plotting in interactive=False mode. [Default: False if interactive=False, True if interactive=True]

### 13.3.5. Caveats

If the spectra being processed has negative signals (due possibly to unflagged spurs or earlier processing which results in an absorption feature having negative flux), then `fitBaseline` will produce an improper correction.

Negative fluxes are non-physical, and `fitBaseline` does a first order correction by adding the negative signal into the final spectrum as follows:

```
(originalFlux + abs(minFlux) ) / ( baselineFlux + abs(minFlux) )
```

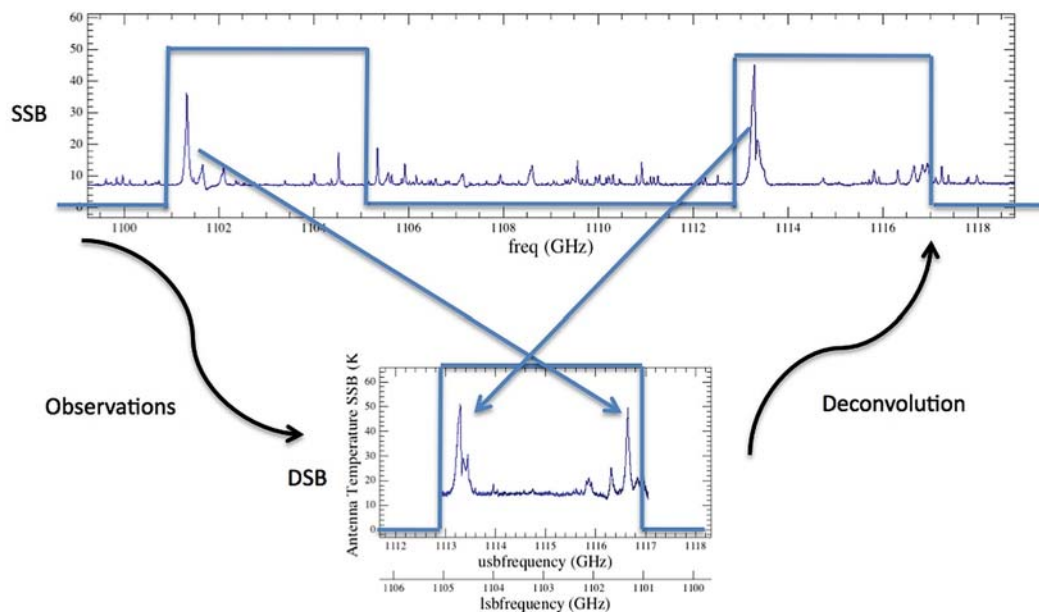
This will over-correct the intensities in the final spectrum. It is imperative that you flag all data points that are not real before applying `fitBaseline` in order to avoid this scenario.

# Chapter 14. Sideband Deconvolution

Last updated: 8 June, 2015

## 14.1. Introduction to doDeconvolution

doDeconvolution is the Level 2.5 pipeline task that separates (or *unfolds*) double sideband (DSB) data that is inherently produced by HIFI's heterodyne process into a single sideband (SSB) result. [Figure 14.1](#) shows an example of how the spectral ranges of the upper and lower sidebands of HIFI are folded together during an observation, causing the spectra to overlap and add, causing features to blend. Also notice that the continuum doubles.



**Figure 14.1. Folding of the upper and lower sidebands**

Fluxes ( $F_{\text{DSB}}$ ) in the DSB spectrum can be expressed in terms of the LO frequency and the IF frequency (where, for bands 1-5, the IF frequency goes from 4 to 8 GHz):

$$F_{\text{DSB}}(\text{IF}) = 2 * \text{usbGain} * F_{\text{sky}}(\text{LO}+\text{IF}) + 2 * \text{lsbGain} * F_{\text{sky}}(\text{LO}-\text{IF})$$

Here,  $F_{\text{sky}}$  are the true input fluxes *from the sky*, and  $2 * \text{usbGain}$  and  $2 * \text{lsbGain}$  are the sideband gain (imbalance) factors, typically close to 1.0. The deconvolution is usually used to reduce HIFI Spectral Scans, which are collections of overlapping observations taken at many LO settings. But in principle, any set of spectra taken at differing nearby LO settings, such that the frequency coverage of each overlaps with the next, may be together deconvolved. Observations taken at multiple LO settings serve to constrain the SSB solution. Given the observed  $F_{\text{DSB}}$  fluxes at multiple LO settings, the deconvolution solves for the unique  $F_{\text{sky}}$  solution that best models the observed multiple  $F_{\text{DSB}}$  observations through iterative chi-square minimisation using the *Conjugate Gradient Method* ([Comito and Schilke 2002, A&A, 395, 357](#)). The method can also be used to simultaneously solve for the  $\text{usbGain}$  and  $\text{lsbGain}$  gain factors, as described below. No detailed knowledge of the algorithm is required by you in order to operate the deconvolution task. With good input data, as few as 3 DSB spectra may be sideband-deconvolved, as shown in an example in [Figure 14.2](#).

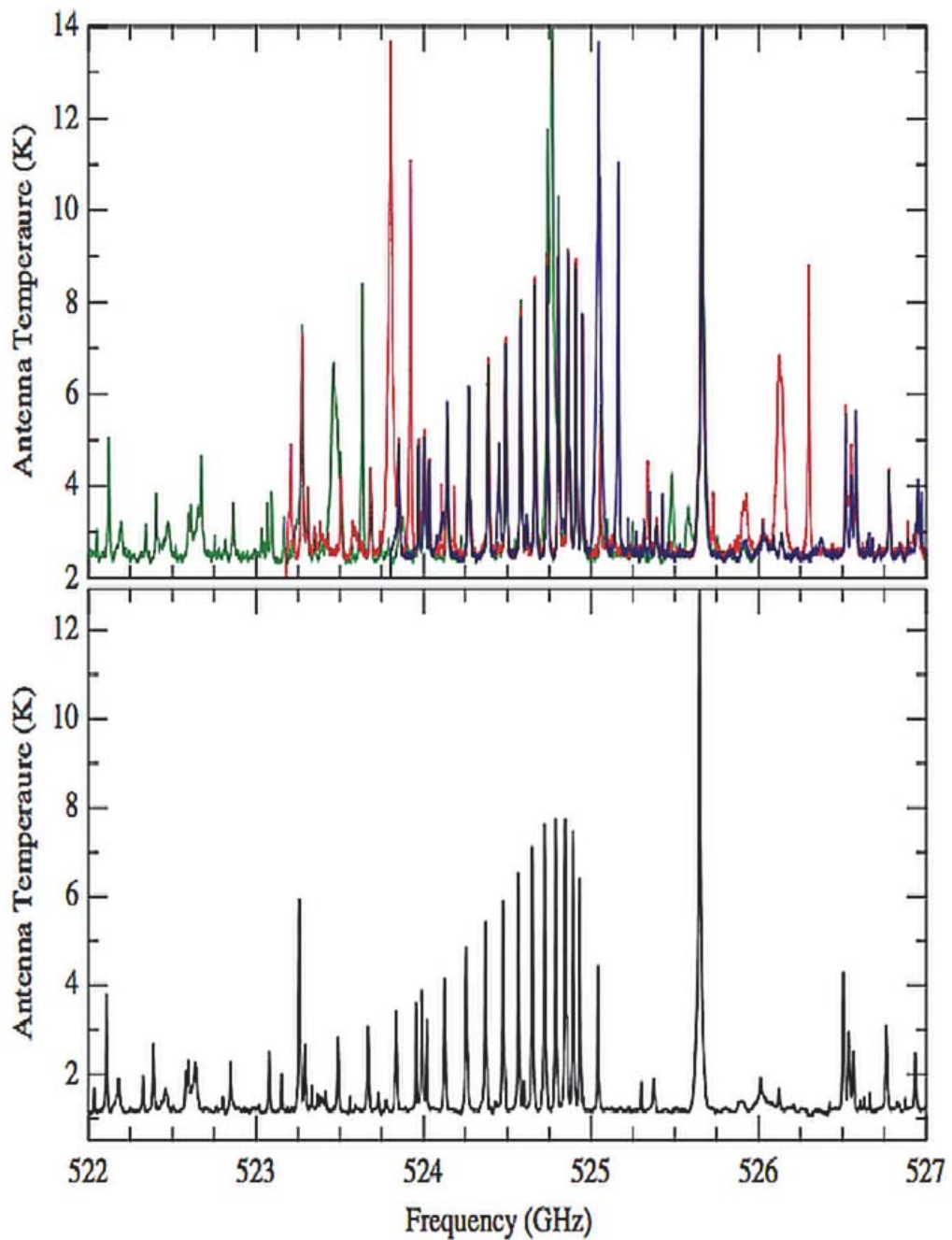


Illustration of the double sideband deconvolution within a section of a Band 1a Spectral Scan with a redundancy parameter  $R=4$  (see [Section 14.2](#)). Top: Three different DSB LO settings shown with separate colours. Note the lines from the other sideband which move at different LO settings. Bottom: Single sideband (SSB) spectrum (in this range) after the deconvolution.

**Figure 14.2. Example of three different DSB LO settings being deconvolved**

## 14.2. Basic strategy for running the deconvolution tool



### Warning

From HIPE 14 onwards, the `decon_result` product called `ssb` is now renamed `dataset`. The consequence of this change is that it will break any scripts where this naming convention is expecting `ssb`. This also includes the *Useful scripts* provided with HIPE ([Section 2.1](#)). You are thus invited to modify your scripts accordingly. **Note** that in this chapter, we liberally use the acronym 'SSB' when talking about the 'dataset' product.

What is input to the deconvolution? When a spectral survey observationContext is selected in HIPE, then one of the applicable tasks listed is `doDeconvolution`. When `doDeconvolution` is selected, a particular data product is nominally used as input to the deconvolution: the Level 2 H-polarisation USB product of the WBS, complete with its frequencies, channel fluxes, channel weights, channel flags, and metadata.

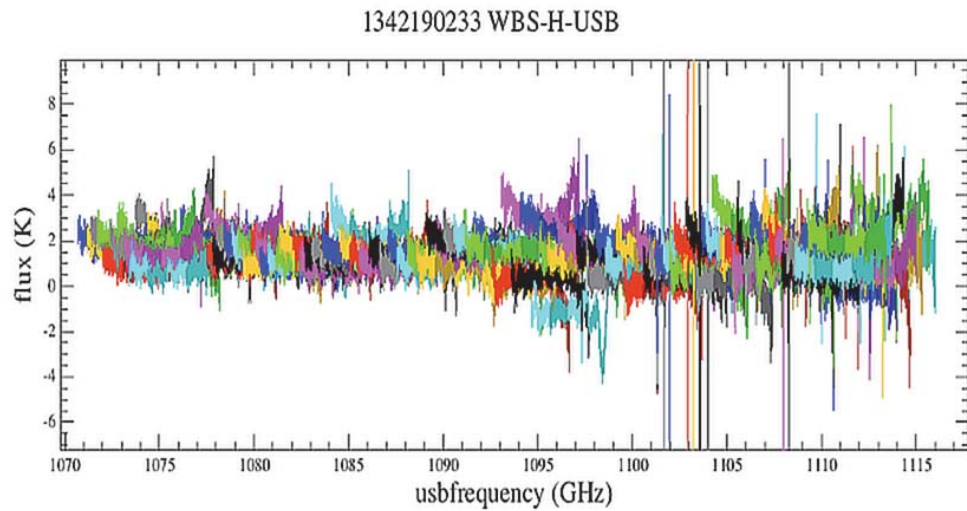
Note that pipeline processing has created this Level 2 data product by interpolating the WBS output onto a rigid grid of evenly spaced frequencies, and that during the interpolation process, the channel flags set in earlier pipeline steps are always propagated forward.

You have the option to select the WBS data from the V polarisation instead of the H polarisation, but the USB product in each case is the input product to the deconvolution.

If the DSB data have been taken with sufficient redundancy (a redundancy parameter  $R=4$  is an advised minimum), are of good signal to noise, contain detected spectral lines (lines in emission and/or in absorption and/or blended - it does not matter), and if the data are completely prepared, then the deconvolution tool typically runs successfully *right out of the box* using all default settings. The data preparation is the hard part. Correct data preparation includes complete spur-flagging, baseline-removal (see [Chapter 13](#)), and defringing (standing-wave removal) (see [Chapter 12](#)). In theory, the true continuum baseline (one unaffected by systematics) may be left present in the dataset, the data can still be deconvolved, revealing the SSB continuum. However, attempting this is not advised. At HIFI frequencies, true continuum baselines are at best linear over one IF passband, and detectable only over a wide survey where the LO spans many tens of GHz. If desired, an underlying continuum can be estimated from the DSB data, recorded, and removed before deconvolution.

When you have fully prepared the Level 2 data such that the baselines are zero and flat, the ripples are as removed as possible, the spurs flagged, and all regions of pathological noise flagged bad, then the deconvolution will immediately (in a few 10's of seconds and a dozen or so iterations), converge on a correct SSB solution. The recommended tool for this preparatory work is the `flagTool` (see [Section 11.5](#)), with which you can inspect or modify flagged channels (and rowflags), and correct baseline distortions such as offsets and sinusoidal standing waves.

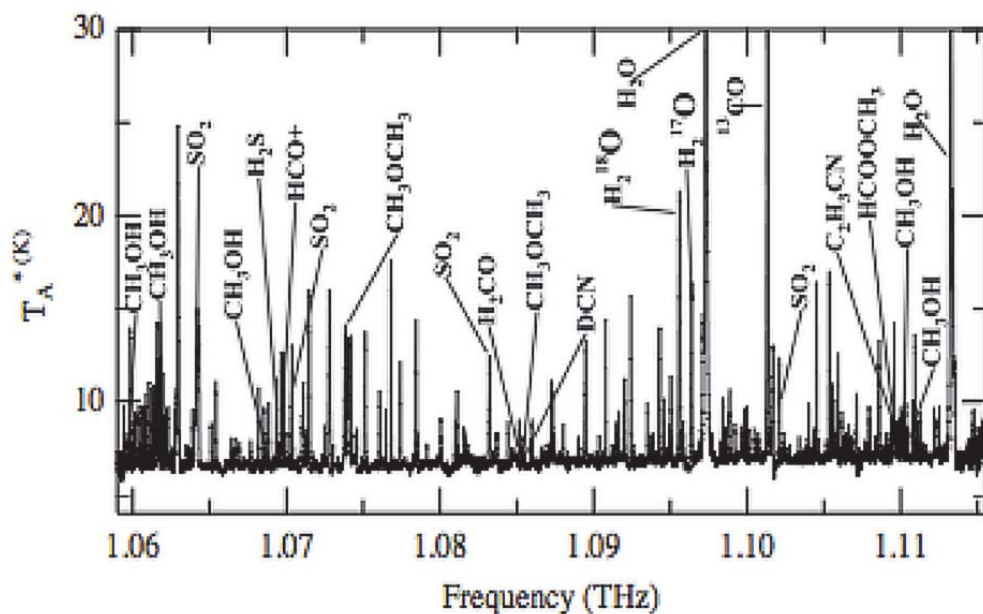
Conversely - if the deconvolution fails to converge or the SSB result looks bad, with spiky or repeating noise patches - this usually indicates that there are problems with your input DSB data. Repairing the input data problems will fix the deconvolution result. A data set of DSB scans before cleaning is shown in [Figure 14.3](#), and a good deconvolution result is shown in [Figure 14.4](#).



3 scans:

Dataset before cleanup. Each colour shows a different spectrum (with its own LO setting) of a Spectral Scan. Spurs, ripples, and bad baselines are apparent.

**Figure 14.3. Dataset before cleanup**



After cleaning the input data, here is the SSB result of a good deconvolution on Orion KL. All evident structures are real.

**Figure 14.4. SSB result after deconvolution**

When you flag channels as containing spurs, or bad data or perhaps an entire spectrum as bad (see [Chapter 10](#) and [Chapter 11](#)), the deconvolution routine rejects these data upon read-in. I.e., the ob-

served DSB spectra that the deconvolution works with internally no longer include these bad data. But the read-in routine records a tally of rejected data, scan by scan, and lists these numbers and their total in the metadata found in the SSB product header. We note that typically at least two channels are flagged as bad fluxes in each WBS spectrum since they are used for calibration. Also, the detection of a flagged strong spur may cause the deconvolution to reject a subband or an entire scan depending on your preference, with an entire scan being a safer setting. Look to the value of the *spur\_rejection* parameter in `doDeconvolution` to adjust the amount of data rejected in the presence of a spur; careful inspection of the data flags with `flagTool` before deconvolution is always recommended.

In the example below (see [Figure 14.5](#)), we show some of the SSB metadata. The first line is the last LO listed. Then, we see 5 bad scans - either due to spurs or flagged by you. Then, the individual channels (in other scans) are listed - first as a total and then, scan by scan.

Spectrum 1d			
Meta Data			
name	value	unit	description
loFreq90	628.1299325462097		LO Frequency in GHz
total_bad_scan	5		Total bad scans
bad_scan32	true		Bad subbands
bad_scan33	true		Bad subbands
bad_scan34	true		Bad subbands
bad_scan35	true		Bad subbands
bad_scan36	true		Bad subbands
total_bad_chan	272		Total number of bad channels
not_observed_chan	272		Channels not observed in 69 scans
num_bad_chan_in_scan2	6		Number of bad channels in the scan
num_bad_chan_in_scan3	2		Number of bad channels in the scan

Part of the `decon_result` SSB metadata showing individual bad scans and a count of bad channels in individual scans.

Figure 14.5. Bad scans and channels stored in the SSB metadata



#### Note

From HIPE 13 onwards, the HIFI calibration tree will contain tables with fine-tuned channel and row flags for spectral scans. These are meant to supersede the flags that were, up to now, assigned automatically by the pipeline, but could lead to significant false positive or miss important artefacts. When such tables are available, the default option *spur\_rejection* = "DO\_NOT\_REJECT\_SCANS\_WITH\_SPURS" is the one to apply. Not all spectral scans could however be covered yet in HIPE 13, so that for the ones still depending on the initial flag tables, the option *spur\_rejection* = "REJECT\_SCANS\_WITH\_SPURS" has to be enforced. The availability, or not, of such table is informed via a new metadata *erpflagged* that `doDeconvolution` will check. When it is missing, the following message will be printed in the console:

*Because HTP metadata 'erpFlagged'=False or it is not present, the spur\_rejection has been reset to 'REJECT\_SCANS\_WITH\_SPURS'!!! (If you want to use another spur\_rejection option, please set enforce\_spur\_rej = True).*

With the above behaviour, the value for *spur\_rejection* will be entirely constrained by the presence, or not, of the aforementioned metadata, and therefore the presence, or not, of the fine-tuned flag tables. This would however make `doDeconvolution` too restrictive as a user would not be able to use certain *spur\_rejection* options depending on what metadata would, or not, be in their data. In particular, until the bulk reprocessing with HIPE 13 is complete, `doDeconvolution` will consider that no observation has yet optimised flag tables and so option *spur\_rejection* = "REJECT\_SCANS\_WITH\_SPURS" will apply whatever. This is however not the best option in case a user has carefully flagged the data.

In order to allow the user to enforce a particular *spur\_rejection* option, a new option *enforce\_spur\_rej* (set to 'False' by default), is thus available to help you control the *spur\_rejection* option. For example, if you have cautiously flagged your data manually, the most optimum call to `doDeconvolution` is probably:

```
decon = doDeconvolution(obs,
    spur_rejection="DO_NOT_REJECT_SCANS_WITH_SPURS",
    enforce_spur_rej=True)
```

which will allow you to use the *spur\_rejection* option even if no optimised table would be found in the observation context.

### Basic call to the task

To run the deconvolution on a spectral scan *observation context* with the default parameters, you invoke:

```
result = doDeconvolution(obs=MyObsContext)
```



### Tip

The only parameter you really need to choose is the polarisation. The one other section of this documentation that is highly relevant to the normal use of the deconvolution is [Section 14.3](#) describing the viewing of results. For almost everyone, this parameter and [Section 14.3](#) are all that is needed in order to be ready to get a useful SSB science product.

The remainder of this Chapter describes advanced techniques - multiple dataset entry, use of rms weighting, Maximum Entropy Techniques, and a range of diagnostic tools including ghost identification. The full range of `doDeconvolution` command line input parameters and their defaults are as follows:

```
decon_result = doDeconvolution(obs=MyObsContext, spectrometer='WBS-H',
    max_iterations=200,
    enforce_spur_rej=False, tolerance=0.0010, gain='GAIN_FIT_OFF_USE_PRESET',
    channel_weighting=False, \
    spur_rejection='REJECT_SCANS_WITH_SPURS', plot_dsb='NO_PLOT', lambda1_channels=0.0,
    \
    lambda2_gains=0.0, cont_offset=0.0)
```

Parameter name	Description
obs	The primary spectral dataset.
obs2_array	An array of additional datasets, which can be an array of 1, but must be an array. Supplying optional additional datasets allows you to, for example, connect spectral surveys across HIFI subbands (e.g. 6b and 7a); or combine a deep single LO DBS AOR with a shallower spectral survey, so as to deconvolve the deep observation in that context.
spectrometer	Observations contexts store H and V polarisations. You can specify which to deconvolve with this option. Choices are: WBS-H, WBS-V. WBS-H is the default, as it is often better.
tolerance	Specifies the tolerance of the solution. When the rms of the residual of the fit changes fractionally by less than tolerance, the algorithm stops iterating. A value of 0.001 is best. Below this value, the algorithm may work too hard to make modelled lines match, and in so doing, produce poor baselines.
max_iterations	Tells <code>doDeconvolution</code> to stop after a specified number of iterations if it has not converged by then, and report the error, although the last result is returned. Healthy deconvolution should converge in less than 20 iterations.
enforce_spur_rej	If true, spur rejection selection will be enforced. Default is set to 'False'.

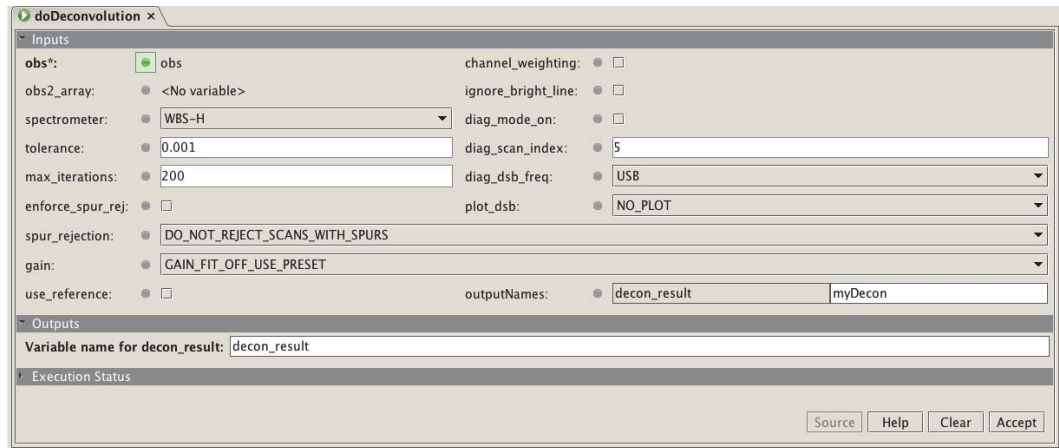


Parameter name	Description
spur_rejection	Flag to reject scans, subbands, or forego rejection of data with known cold load spurs. Choices are: REJECT_SCANS_WITH_SPURS, REJECT_ONLY_SUBBANDS_WITH_SPURS, and DO_NOT_REJECT_SCANS_WITH_SPURS (default). The first setting is the most conservative; to preserve good data in scans with weak flagged spurs, try the other options.
gain	You have the option to try to also fit the factors $2 * \text{usbGain}$ and $2 * \text{lsbGain}$ given in the formula above which the deconvolution calls <i>the Gains</i> . The options here are: GAIN_FIT_OFF_ASSUME_EQUAL and GAIN_FIT_OFF_USE_PRESET (default). The words <i>equal</i> and <i>preset</i> indicate to the routine whether to start with the assumption that the gains are equal to 1 or if they are equal to a preset value that is found in the metadata in the observation context as determined by previous calibration studies.
use-reference	If selected, <code>doDeconvolution</code> will use the reference spectra only.
channel_weighting	Toggles whether or not the deconvolution uses the weight values in the data, which will give less weight to noisier data.
ignore_bright_line	Will ignore any lines flagged as <code>HifiMask.BRIGHT_LINE</code> . See <a href="#">Section 14.5</a> for some extra information on using this parameter.
diag_mode_on	Diagnostic mode to generate interim product.
diag_scan_index	When the diagnostic mode is on, you may select which spectrum (e.g. of a spectral scan) to follow through the deconvolution (where the indices start with 1).
diag_dsb_freq	Diagnostic DSB frequency selection, USB (default), or LSB.
plot_dsb	Toggles visualisation on and off. When on, the SSB output solution can be viewed against the DSB input. SSB features lacking DSB counterparts in all DSB spectra are called <i>ghosts</i> and should not be believed. Note - the upper and lower plots in this display will pan and zoom together so as to enable you to examine the same zoomed spectral regions in tandem. Choices are: NO_PLOT (default), USB, LSB, ULSB, and PLOT_ALL

To run the deconvolution on a spectral scan using one obs and an array of obs with the default parameters you invoke:

```
my_obs2_array = [my_obs2, my_obs3]
decon_result = doDeconvolution(obs=my_obs, obs2_array=my_obs2_array)
```

The deconvolution tool can also be run from a GUI (see [Figure 14.6](#)) by clicking on the obs context in the *Variables* window, then double clicking `doDeconvolution` in the *Task* list.



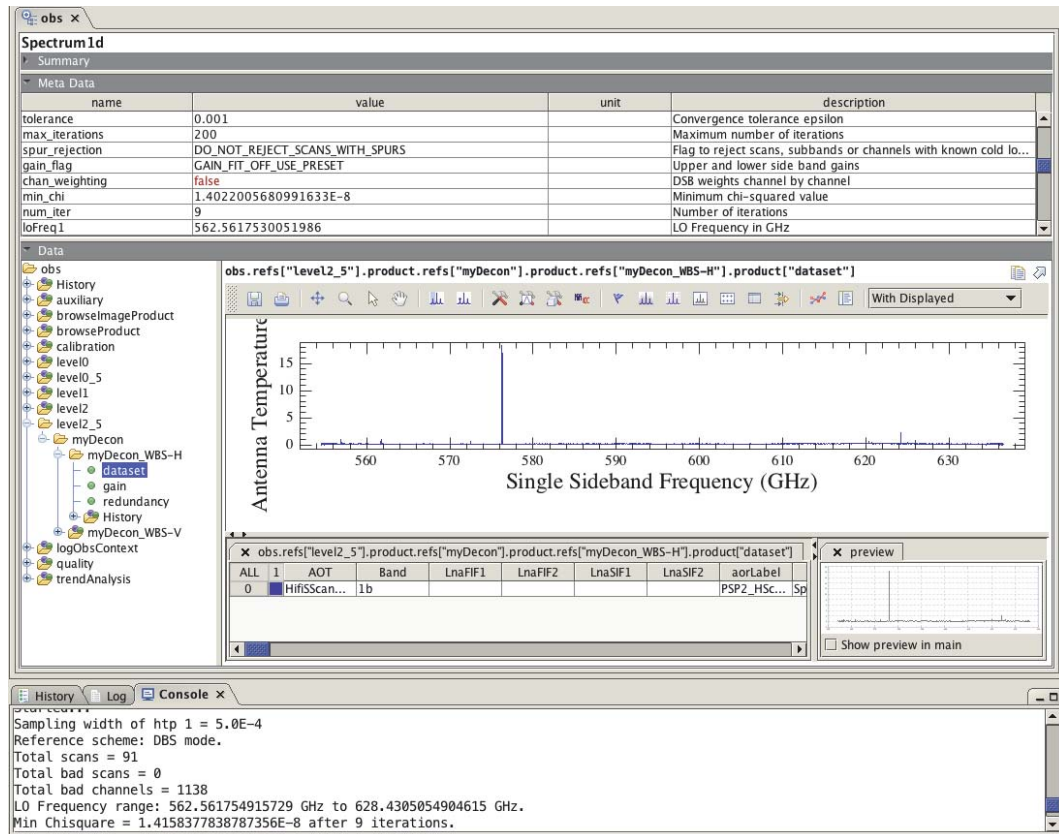
**Figure 14.6. Deconvolution GUI**

Like other GUIs in the system, once the *Accept* button is hit, the task is called and starts. The command line version is written in the console window so you know exactly how the task was called. This output can be cut and paste into scripts for repeatability.

## 14.3. Viewing deconvolution results

The single sideband (SSB) result of the deconvolution, called *dataset*, includes a header which contains, in metadata, your selected parameters of the deconvolution run, including for example, number of iterations and the tolerance. These can be seen in the HIPE screenshot (see [Figure 14.7](#)). The output product result can be viewed with the product viewer.

If you have set the console logging level to INFO or lower (see Section 12 of the HIPE Owner's Guide), this information, plus a more detailed description of data quality will also be output to the terminal from which HIPE was run. It will also appear in the 'Log' tab associated with the HIPE Console area.



The dataset (SSB) product is displayed as well as the running report of the deconvolution.

**Figure 14.7. HIPE screenshot**

The dataset (SSB) result is a dataset that can be viewed with the Spectrum Explorer. On the command line, it can be extracted from the `decon_result` produced by `doDeconvolution` (or from the product labelled as `myDecon_WBS-H/V` found under `myDecon` in the Level 2.5 product) as follow:

```
ssb = decon_result["dataset"]
```

This contains the deconvolved spectrum, and is the primary output of the tool as seen in [Figure 14.7](#).

The dataset (SSB) output also contains an estimate of the per-channel weighting. This is calculated as the inverse of the variance of the data that went into solving for the flux of that channel.

If after inspecting the dataset (SSB) you wish to remove a residual baseline or standing wave from your deconvolved data, you can save the dataset (SSB) as a `SimpleSpectrum` using the `convertSingleHifiSpectrum` task and apply `fitBaseline` or `fitHifiFringe` to that `SimpleSpectrum`.

The dataset `gain` can be viewed with dataset inspector. On the command line, it can be extracted from the product with:

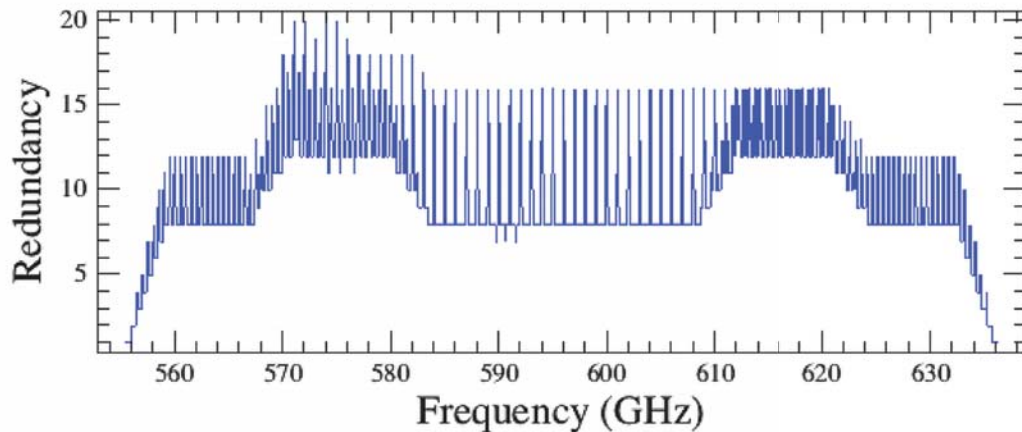
```
gains=decon_result["gain"]
```

As discussed above, when requested, the deconvolution tool will estimate the sideband gains values. These estimates are stored per LO tuning in this product. The list of gains also includes a column giving the rms value for each gain, which is computed as a measure of the dispersion of all associated gains related by DSB overlap.

The DSB input data redundancy is also available as one of the output products and can be displayed in the Spectrum Explorer (see [Figure 14.8](#)). The redundancy shown is typically more than twice the

redundancy specified in the AOR set-up for two reasons: First - you have specified a redundancy that is defined as the number of times one of the two sidebands observes a particular sky frequency - while the plot shows the total number of times *either* sideband observes one sky frequency. Second - the AOR setup inserts additional range-edge redundancy, where a range-edge can be as wide as 26 GHz on either end of the spectral scan total range. For these two reasons, the redundancy showing on the plot is substantially higher than the redundancy requested in the observation.

The redundancy plots have the following utility: when a channel has a redundancy less than three or four in the plot, the dataset (SSB) solution at this channel has questionable veracity.



The Redundancy Histogram Plot from a moderate width 90 GHz survey shows dual peaks where both sidebands reached the same sky frequencies in the added settings LO near band edges.

Figure 14.8. Redundancy Histogram Plot

## 14.4. Exporting deconvolution results to Class

To export the results of deconvolution to Class, use the `hifiDeconToClass` task.

- In the GUI, open the task on the `decon_result` produced by `doDeconvolution` or the product labelled as `myDecon_WBS-H/V` found under `myDecon` in the Level 2.5 product.
- Specify the full path name of the fits file you want to create, remembering to add the `.fits` extension.
- In the command line, this is done as follows:

```
hifiDeconToClassObj = hifiDeconToClass(decon_result=decon_result, fileName='/Users/Me/decon_result.fits')
```

## 14.5. Advanced settings and diagnostic functions

### 14.5.1. Advanced methods

**Bright Lines**

Due to the iterative nature of the algorithm and small roundoff errors during resampling, very bright lines can sometimes appear as *ghosts* in the final deconvolved spectra. In such cases, it is recommended that you mark all such lines using the `BRIGHT_LINE_FLAG` via the `flagTool` task, or command line alternatives (see [Chapter 11](#)).

Once done, we recommend running deconvolution twice. First, without the `IGNORE BRIGHT LINE` option, which will provide a result that is best used to measure the properties of the bright lines. Then, running the decon with the `IGNORE BRIGHT LINE` option turned on - this will allow you to characterise the fainter features.

*Bright lines* in this context are relative to the noise, but typically are  $> 10$  Kelvin in amplitude. To quickly assess whether the solution provided by the deconvolution algorithm generates ghosts, the task can be run with the plotting option, e.g. `Plot DSB` (see [Section 14.5.2](#) below).

### Maximum Entropy

The maximum entropy option is *turned on* by setting either of the lambda values `lambda1_channels` and `lambda2_weights` to positive real numbers on the order of magnitude of the rms reported in a standard deconvolution. Maximum entropy is used infrequently, and only as a *stop-gap* measure to help a bad situation with the input data. The bad situation can include:

- Insufficient redundancy (i.e. the number of overlaps of each sideband on itself), say a redundancy of less than  $R=4$
- Too few lines (since line strengths guide the deconvolution)
- Poor, or excessively noisy data
- Also, if the solution of the nominal deconvolution contains periodic noise patterns, or the solved gains deviate widely from 1.0.

The inclusion of the maximum entropy method adds a term to the quantity being minimised. Without this term, the quantity being minimised is the Chi-square difference between observed double sideband (DSB) spectra and the modelled DSB spectra. The minimisation is accomplished by altering the SSB model spectrum from which the modelled DSB spectra are derived. But, when the input data are of poor quality, sparse sampling, or contain few lines, repetitive noise structures may appear in the solution and/or the fitted gain values may begin to diverge and become non-physical. Since the entropy of these artifacts is low, we compute the solution's channel and gain entropy, and subtract it from the Chi-square value at each iteration. In this way the deconvolution must still match the observations but has the additional task of keeping the entropy of its solution high, yielding a non-highly patterned result. Turning-on the entropy terms helps the deconvolution *behave*.

The two lambda factors are the two relative weights of the entropy terms for the channel solution (dataset spectrum, here called liberally 'SSB' for the remainder of this section) and the gain values. To use the maximum entropy lambda terms strategically, set the weight low, e.g. channel weight to  $10^{-5}$  and gain weight to 0. Slowly increase either of these weights ( $10^{-4}$ ,  $10^{-3}$ , ...) and look for an improvement. The weights should not exceed  $10^{-1}$ .



#### Warning

There is currently no evidence that *Maximum Entropy* helps. If used incorrectly, it can lead to incorrect results. Therefore, if you would like to try *Maximum Entropy*, it is recommended that you consult with an expert via the helpdesk.

### Channel Weighting

The channel weighting option can be turned-on so as to utilise a relative weight that has been attached to estimate the veracity or correctness of the flux in each DSB channel. In this way, spectral information from low S/N regions of the spectra will carry less weight in contributing to the reduced chi-square between the modelled and observed spectra. The weighting value is equal to  $t_{\text{int}}/T_{\text{sys}}^2$ .

It has been seen that turning this on has improved the chi-square of deconvolution for some examples, but the actual effectiveness of using this weight term is still unclear - partially because we have been comparing weighted and un-weighted chi-square values, neither of which are normalised chi-square values.

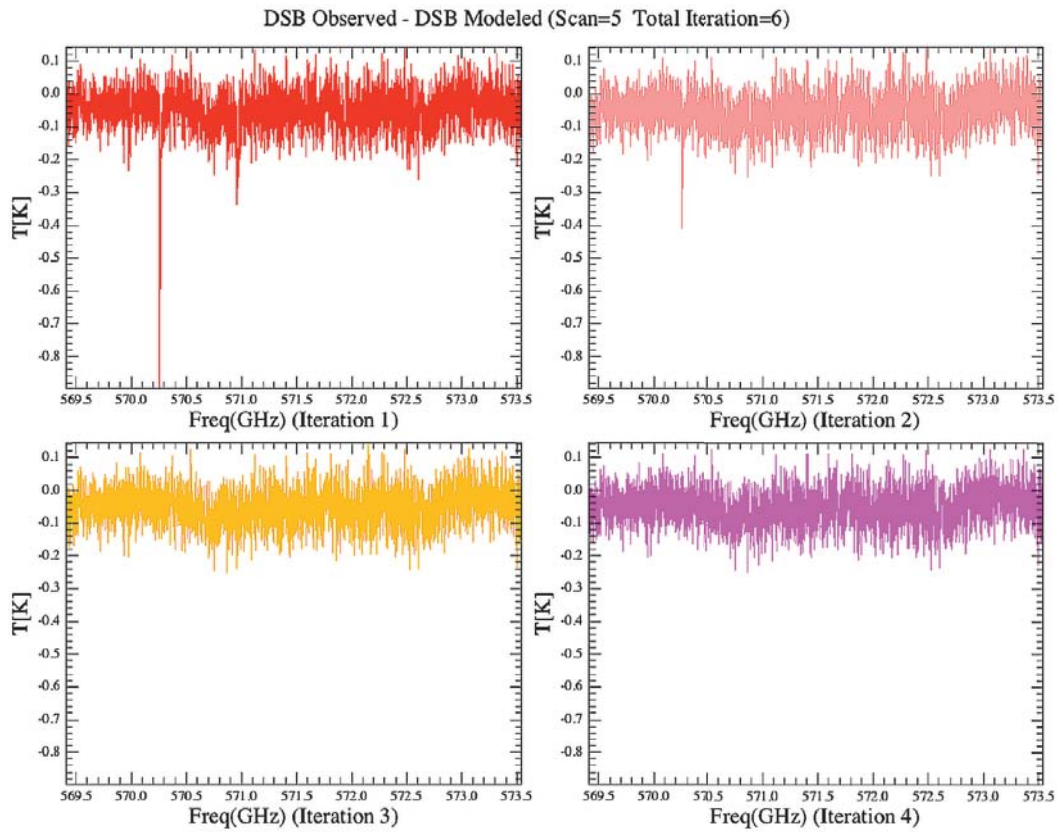
## 14.5.2. Diagnostic functions

### Plot DSB (default="no\_plot")

This is one of the most useful diagnostic plots as it can help you check for ghosts. Ghosts are attempts of the deconvolution to compensate for gain errors of line measurements by creating fictitious line (sometimes negative) in the other sideband and thus at an offset frequency. A ghost is identified as a ghost when it fails the following test: A true SSB line must appear in all DSB input spectra covering that line. So this diagnostic plot window gives you an upper (SSB) and lower (all the input DSBs) plot where for the lower plot you can choose the frequencies to be the USB or LSB frequency interpretation with the *USB* and *LSB* options. Additionally, you can plot both USB and LSB together with the *ULSB* option, and you can create all plots with *plot\_all*. The selected interpretation does not really matter. What matters is that for the line in the upper SSB plot to be real, it must be seen in all the spectral scans in the lower box. (There is one caveat - it is possible, but very unlikely, that a nastily placed absorption line in the other sideband could overlap on the line and cancel it out, but that is a highly unlikely occurrence).

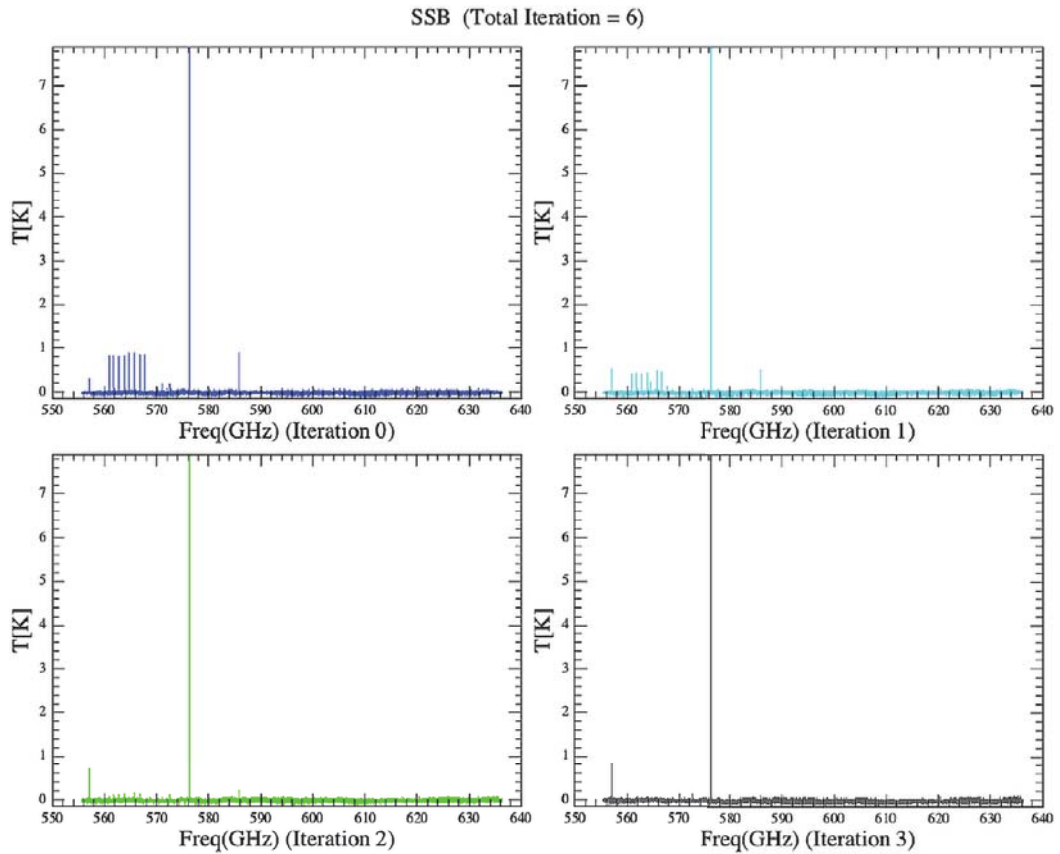
### Diagnostic Mode (true or false = on or off) and Ancillary Products

When this mode is turned on, we are able to watch the inner action of the deconvolution as it models a single DSB spectrum. We have presented the Deconvolution with a Spectral Scan containing  $N$  DSB input spectra. The task for the deconvolution is to arrive at the best sky flux spectrum that will replicate all  $N$  scans. If we wish to follow one of these - the  $n$ th out of  $N$ , we specify this one via *diag\_scan\_index* and we will get plots at each iteration showing both the SSB being iterated and refined into its final form, and also the DSB of choice (the  $n$ th) being modelled more and more correctly: we plot the  $n$ th (DSB\_obs - DSB\_modelled). Note - this is a memory-intensive process - you should have at least a 1 GByte RAM and, on install, have told HIPE you wish to be using it, in order to run this diagnostic mode. In [Figure 14.9](#) and [Figure 14.10](#) we show examples.



We show the difference between the 4th DSB Observed and its modelled form on the first four iterations where the spikes go away and the noise goes down a bit.

Figure 14.9. Observed vs. Modelled

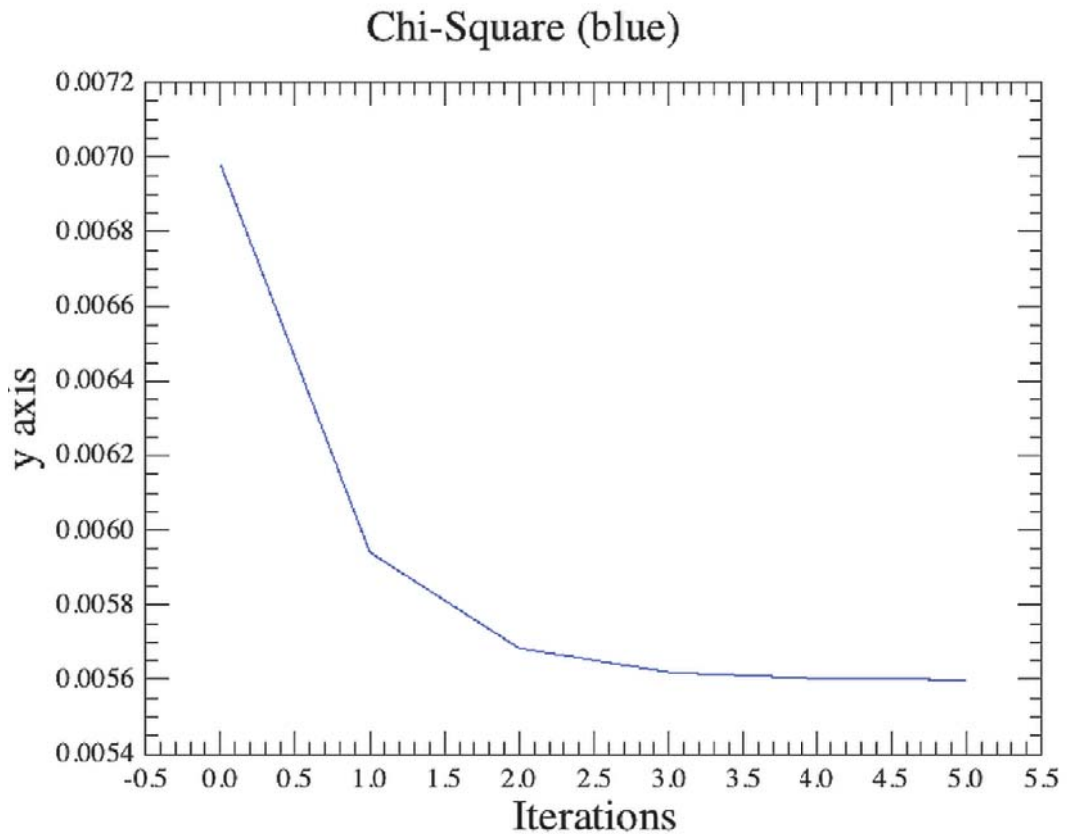


We show the entire SSB progressing through the first 4 iterations where the algorithm obliterates a series of ghosts.

**Figure 14.10. Iterations progression**

Finally, one strong associated product of the diagnostic mode is the plot of Chi-square vs. iteration. See [Figure 14.11](#). Had we turned on the two entropy terms, we would have seen four plots on this axis: (1) the channel Chi-square, (2)  $\lambda_1$  \* the channel's entropy, (3)  $\lambda_2$  \* the gain's entropy, and then (4): the sum  $((1) - (2) - (3))$ , that is minimised.





Chi-square vs. Iteration. This should show a steep (exponential) decline vs. iteration number leading to an asymptotic tail as the algorithm converges.

**Figure 14.11. Chi-square vs. Iteration**

#### Deconvolution Interim Products

Please note that in these products, we are tracking a single DSB spectrum  $n$  of  $N$ .

#### **dsb\_input:**

These data are in vertical columns. All the  $N$  observed DSB input spectra for deconvolution, with all their fluxes, and two frequency scales corresponding to the USB and LSB are given, one after the other. The last column, *dsbPointer*, indexes the spectra - from 0 to  $N-1$  with about 8096 points in each spectrum.

#### **dsb\_model:**

This is a complex structure. The first column is the index number of the iteration: 0 to something like 5, 10, or 15. Then there are four more columns: *dsbModelInt*, *dsbModelImage*, *dsbModelSignal*, and *dsbModelPointer*.

- *dsbModelInt* are the double sideband model intensities for the modelling of the  $n$ th DSB input
- *dsbModelImage* and *dsbModelSignal* are the Upper and Lower sideband frequencies (in GHz) for this  $n$ th spectrum
- *dsbModelPointer* - a copy of *dsbPointer* described above where negated products are listed as  $\lambda_1 * \text{entSb}$  and  $\lambda_2 * \text{entGain}$ . These row arrays are populated when either of the lambda weights are non-zero. Note - all four terms (including the sum) are also plotted on the same *chi\_sq* vs. iteration plot for you.

#### **interim\_issb:**

A row for each iteration of the entire ssb solution for that iteration of the deconvolution.

**interim\_lgain:**

A row for each iteration of the entire gain array (first N USB values and then N LSB values - one value for each N scans - so 2 N values). All will be equal to 1.0 when gain is off and assumed equal.

**interim\_chiSquare:**

chiSquare value - at each iteration we record the chisquare value. With both lambdas 0 it is from the channel fitting, and if asked for, gain fitting alone.

But when either of the two lambdas are non-zero, the total chisquare is more complex. In this case:

$$\text{chiSqSum} = \text{chiSquareValue (from channels, gains)} - \text{lambda1} * \text{entropy(SSB)} - \text{lambda2} * \text{entropy(Gain)}$$

where the negated products are listed as *lambda1\_entSsb* and *lambda2\_entGain*. These row arrays are populated when either of the lambda weights are non-zero. Note - all four terms (including the sum) are also plotted on the same chi\_sq vs. iteration plot for you.

# Chapter 15. How to make a spectral cube

Last updated: 31 August, 2015

## 15.1. Introduction to doGridding

Spectral cubes from all HIFI mapping observations are produced as part of the SPG pipeline and are found in the `cube` context of the Level 2.5 product. Please note that baseline and standing wave corrections are not done automatically in the pipeline because of the risk of harming the scientific content of the data. The cubes produced automatically by the pipeline must therefore be regarded as browse-quality products. Consequently, it should not be assumed that scientific analysis can immediately be carried out on them.

*It is strongly recommended that you inspect your Level 2 HTPs for baseline drifts and residual standing waves and decide if some, or all, datasets need to be cleaned up before re-gridding the Level 2 data using the `doGridding` task.*

The use of the `doGridding` task is described in this chapter, while corrections of baselines and standing waves are discussed in [Section 13.3](#) and [Chapter 12](#).

The `doGridding` task is provided to allow you to create cubes from the Level 2 data. Your science objectives may require you to create cubes using different parameterisations (pixel size, beam size etc...) for the gridding than the defaults used by the pipeline.



### Note

The cubes produced automatically by the pipeline (Level 2.5), and consequently by `hi-fiPipeline` if you choose to run it again, will stitch the WBS subbands and create one final cube (with the index `'_1'`). You will not see individual cubes per subbands except for the HRS. Therefore, in this chapter, we show you how to create cubes per subband using the task `doGridding`, applicable to both WBS and HRS. Details of the HIFI pipeline are discussed in [Chapter 4](#) and [Chapter 5](#).

## 15.2. doGridding Summary

The default operation (by the pipeline) of the `doGridding` task is to create a cube from the Level 2 HTP for each given spectrometer. The cubes generated are stored in the Level 2.5 product called `cubesContext`, where you will find: 1) one cube per spectrometer per subband for HRS, and 2) one 'subbands stitched' cube per spectrometer for WBS. If the map was carried out with a non-zero position angle, you will find two sets of cubes at Level 2.5, one non-rotated and the other with the rotation angle applied. In addition to the cubes, the Level 2.5 is also populated with subband stitched Level 2 HTPs.

Each slice of the cube is produced by computing a two dimensional grid covering the area of the sky observed in a mapping mode. For each pixel in the grid, the task computes an equally weighted convolution of those spectra falling in the convolution kernel around that pixel. The convolution ignores any data that is flagged by the pipeline with `SPUR_CANDIDATE`, `SATURATION`, `BAD_PIXEL`, `NOT_CALIBRATED`, `GLITCHED`, or `DARK_PIXEL` with the consequence that any channel that is always flagged in such a way will result in a cube slice consisting of NaNs. The weights in the cube reflect the fact that a lower integration time has been used when flagged data is ignored.

As part of the automated (SPG) pipeline, `doGridding` handles Observation Contexts but if you are making a cube yourself then you should use a Level 2 `HifiTimelineProduct` (HTP). The reason for this is that `doGridding` assumes that the spectra have a linear frequency axis, and this may not be the case for Level 0 or Level 1 HTP, where there can still be overlap of subbands. Resampling to a linear frequency axis is carried out in the [doFreqGrid](#) step of the Level 2 pipeline.

The `doGridding` Task can be found in the *Applicable* folder of the Tasks view when an HTP is selected in the variable view; double-click on it to open the dialogue in the Editor View. You can also find the task under the *Task View* in *By Category* → *HIFI*.

**Note** that using `doGridding` outside of the pipeline will not produce a 'subbands stitched' final cube for WBS. Instead, you will find one cube per spectrometer, per subband just like for the HRS.

**Note** also that using `doGridding` outside of the pipeline will produce a rotated cube if `flyAngle` is non-zero. The task will automatically read the `flyAngle` value from the metadata and set the parameter. If your observation was taken with a non-zero `flyAngle`, upon calling the `doGridding` task for the first time on an htp, you will see the value already set. Therefore, if you wish to recreate the native orientation on the sky, you must explicitly set the parameter to `flyAngle=0.0` in the GUI, or add it in the command line option.

To create cubes using the default parameter values, simply load an HTP into the GUI and press 'accept'. In addition to the usual echo to the Console, the parameters used by `doGridding` as well as some details of the output cube are written in the Log window. The GUI will produce three variables called `cubesContext`, `cubes`, and `cube`.

- The variable `cubesContext` is a `MapContext` which contains an array of cubes, one for each subband, and allows you to easily browse the cubes from the *Context Viewer* (right click on the `cubesContext` and select *Open With... → Context Viewer*) without need to extract them from the `cubesContext`. If you wish to extract a cube from the `cubesContext`, drag the cube name into the *Variables View*. In the command line this is done as follows:

```
# Extract cube for subband 1
cube_v1 = cubesContext.refs["cube_WBS_V_USB_1"].product
#
# Extract cube for subband 3
cube_v3 = cubesContext.refs["cube_WBS_V_USB_3"].product
```

- The variable `cubes` is an array (a `PyArray`), and it contains the same data cubes (one per subband) as in the variable `cubesContext` but without the History. You cannot view the cubes in the same way as with the `cubesContext` but you can extract the cubes in a more simple way, as follows:

```
# Extract cube for subband 1, subband 2, subband 3, and subband 4
cube_v1 = cubes[0]
cube_v2 = cubes[1]
cube_v3 = cubes[2]
cube_v4 = cubes[3]
```

- Finally, when running the task GUI, the **last cube** in the array is also created as a variable called `cube`; this is helpful if you are only creating a cube of one subband.

To obtain the same variables `cubesContext` and `cubes` from the command line:

```
cubesContext, cubes, cube, xPoints, yPoints, convolutionTable, grid =
doGridding(htp=htp)
```

Or equivalently, we can create an `ArrayList` of length=7 (named *result* here), and extract each product:

```
result=doGridding(htp=htp)
cubesContext=result[0]
cubes=result[1]
cube=result[2]
xPoints=result[3]
yPoints=result[4]
convolutionTable=result[5] # by default, detail=False, so this table will be empty
grid=result[6]
#
# to obtain a table with details of the convolution performed at each pixel
```

```
result=doGridding(htp=htp, detail=True)
convolutionTable=result [5]
```

The GUI looks like this:

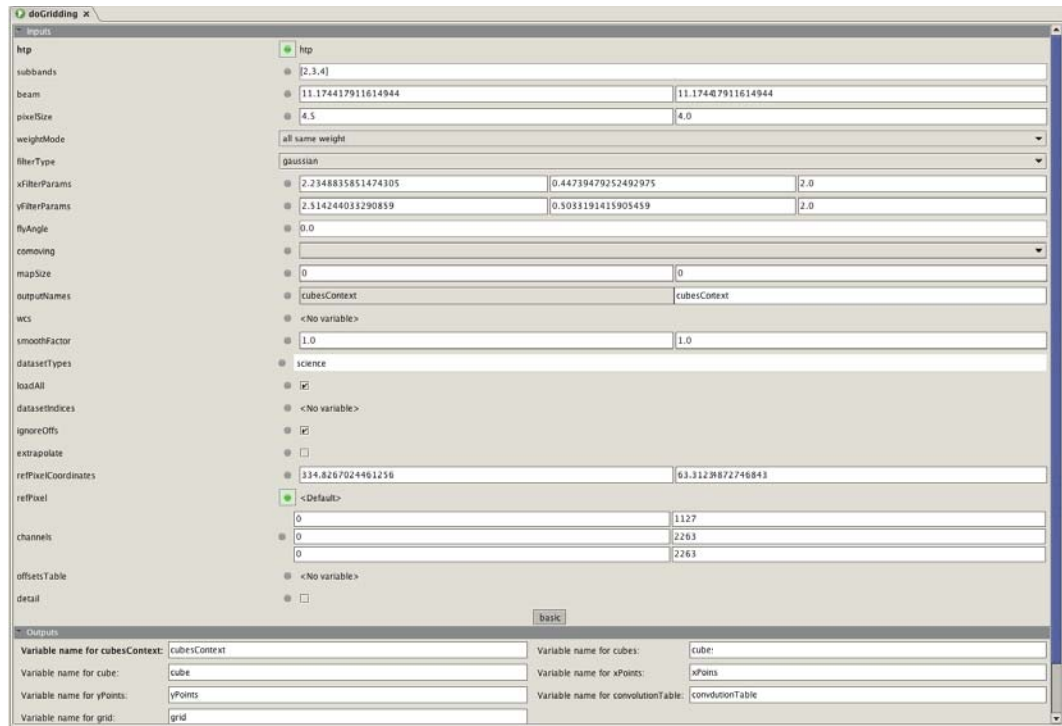


Figure 15.1. The doGridding GUI

The SimpleCube product can be viewed and analyzed in the [SpectrumExplorer](#).

There are many options you can change in the doGridding task. They are listed, with a brief description, in the table below and explained more fully in the next section. Note that you do not need to specify an option if you wish to keep the default behaviour. By hovering the mouse over the parameter names in the GUI, you can find more information about the parameter and some tips on usage. Additional information of these parameters is also available in the HIFI User's Reference Manual under [doGriddingTask](#) in *HIFI User's Reference Manual*.

Parameter name	Description	Command line example
<i>htp</i>	HTP containing data to be gridded. Should be a Level 2 HTP.	
<i>subbands</i>	Indices of the subbands to be gridded.	<b>subbands=Int1d([2, 3])</b>
<i>beam</i>	Beam size in arcsec. An appropriate value for the data is passed to the task but you may wish to modify this to compare with other data.	<b>beam=Double1d([40.0])</b>
<i>pixelSize</i>	Pixel size in arcsec. Appropriate values for the data are passed to the task but you may wish to change them.	<b>pixelSize=Double1d([15,25])</b>
<i>weightMode</i>	Weighting used in the convolution. Default is to use equal weighting but you can also	<b>weightMode="equal" or "selection"</b>

Parameter name	Description	Command line example
	choose to use the weighting in the dataset.	
<i>filterType</i>	The filter used in the convolution. Default is to use a Gaussian filter but a box filter may be more appropriate for a raster map.	<b>filterType="box" or "gaussian"</b>
<i>xFilterParams</i>	Numerical parameters for the filter function in the x-axis (longitude, or RA direction). Unit is pixels. The task auto-computes the best values for the beam size, pixel size and smoothing selected, set to zero to allow autocomputation.	<b>xFilterParams=Double1d([0.5])</b>
<i>yFilterParams</i>	Numerical parameters for the filter function in the y-axis (latitude, or dec direction). Unit is pixels.	<b>yFilterParams=Double1d([1.5])</b>
<i>detail</i>	If turned on, will generate a table of information about the convolution process. Increases processing time significantly.	<b>detail=True</b>
<i>loadAll</i>	Loads all datasets into memory to speed up execution of the task. This is done by default. If you find that you run into memory consumption issues it may help to turn this off.	<b>loadAll=False</b>
<i>datasetIndices</i>	Indices of the datasets to be used to create a cube. The values set here will override any input from the datasetTypes parameter.	<b>datasetIndices=( [3,4,5] )</b>
<i>comoving</i>	Set to None by default, which will create maps with centre that tracks the coordinates of a moving (Solar System Object - SSO) target or with absolute positions for non-SSO targets. You can turn it on to force a map to be made in comoving coordinates and off to force a map created in absolute coordinates.	<b>comoving=None</b>
<i>flyAngle</i>	Angle of the map in respect to the RA/dec axes. If the position angle (pattAngle) is non-zero then flyAngle is 180deg - pattAngle. Note this is not the same as the identically	<b>flyAngle=165</b>

Parameter name	Description	Command line example
	named AOT parameter; for more about the difference between flyAngle, pattAngle, and posAngle, see <a href="#">Section 15.3.5</a>	
<i>channels</i>	Specify the start and end range (in channel numbers) of the cube to be produced. By default all channels are used.	<b>channels = [[1000,2000], [1000,2000], [1000,2000], [1000,2000]]</b>
<i>refPixelCoordinates</i>	Specifies the coordinates of the reference point (the map centre). The task computes this when creating the cube or you can specify the coordinates in the same units as the data (usually decimal degrees).	<b>refPixelCoordinates = Double1d([83.805, -5.368])</b>
<i>offsetsTable</i>	A table you provide to give the positions of each pixel. This can be used in the case that the longitude and latitude cannot be found in the data.	<b>offsetsTable=myoffsetsTable</b>
<i>datasetTypes</i>	Supply the type of data to be gridded into a cube. Look in the HTP summary table to find the types available. The default is to use the science data. Only science data is available at Level 2 so this is unlikely to be used.	<b>datasetType="science"</b>
<i>refPixel</i>	Position of the reference pixel (map centre) in pixel coordinates. Note that the (0,0) pixel is the bottom-most, left-most pixel in the map.	<b>refPixel = Double1d([3.5, 4.0])</b>
<i>smoothFactor</i>	A smoothing factor to expand the kernel size. The default is set to 1.0.	<b>smoothFactor=1.0</b>
<i>ignoreOffs</i>	Ignore datasets from the OFF position. The default value is set to 'True' (or 1). To process the reference spectra, this should be set to 'False' in the command line (the ignoreOffs button should be deselected in the GUI).	<b>ignoreOffs=False (or =0) if you want to process the OFF spectra</b>
<i>extrapolate</i>	Used to switch on / off the extrapolation. Decide if the pixels far away from real observed position should be filled with a value (extrapolate = True), or masked (extrapolate = False). With (ex-	<b>extrapolate=False (default)</b>

Parameter name	Description	Command line example
	trapolate = False) the pixels with a weight below the value defined in the property: "hcss.hifi.dp.off.filter.threshold" (default=0.01); will be masked, filling them with the value NaN.	
<i>mapSize</i>	Size of the map, in pixels. The default operation of the task is to calculate a size based on the input data; with this parameter you can set the x (longitude) and y (latitude) sizes of the map in pixels.	<b>mapSize=Int1d([10,20])</b>
<i>outputNames</i>	Provides a name to the output that should be included in the ObservationContext in the level 2.5 Pipeline. The output should return a Product or a Dataset. The output (its name is the key in the HashMap) will be included in the ObservationContext with a mapContext that will have, as name, the correspondent value provided in the HashMap.	<b>outputNames=cubesContext</b>
<i>wcs</i>	Provide a WCS to perform the projection.	<b>wcs=Mywcs</b>

## 15.3. Using doGridding...

### 15.3.1. ...to change beam, pixel, and map size

- Mapping observations from *OD 835* onwards, and processed with HIPE 8 onwards, are gridded using the scan and readout spacing used during the observation, whilst earlier observations are gridded using a better approach than in HIPE 7. Note that the data must be processed including the `doUplink` step from the Level 0 pipeline so you must either retrieve data from the HSA processed already with HIPE 8 onwards, or you should run the script given in the introduction to this chapter, where more information is given on this topic, [Section 15.1](#).

Despite the need to grid data according to your original specifications, you may wish to regrid data with a new pixel or beam size in order to compare with other observations. The beam width in arcseconds is automatically entered into the GUI when you pass the HTP to the task. It can be found in the *beam* box and can be edited there. If you modify the beam size then the filter parameters will also be automatically updated unless you specify parameter values yourself, see [Section 15.3.6](#) below.

For observations from OD 835 onwards, the beam width is calculated from the calibration tree using the frequency requested in HSpot. Note that this results in slightly different beam sizes for the WBS and the HRS as a consequence of how the spectrometers are set up. For older observations, the beam size is calculated from  $HPBW["] = 75.44726 * \text{wavelength}[\text{mm}]$  where wavelength is the LO Frequency.

The beam width can be set in the command line with:



```
# The extended version which can be adapted to the rest of this section
cubesContext, cubes, cube, xPoints, yPoints, convolutionTable, grid =
  doGridding(htp=htp, beam=[15.4])

# or the shorter version, where, here, we are only interested in obtaining the
  cubesContext
cubesContext = doGridding(htp=htp, beam=[15.4])[0]
```

The HIFI beam is symmetric but you can also specify an elliptical beam by specifying the dimensions of the x and y axes:

```
# specify an elliptical beam. In this case the beam is wider along the vertical
  axis.
cubesContext = doGridding(htp=htp, beam=[20.0, 40.0])[0]
```

- For observations from OD 835 onwards, the `doGridding` task will compute the pixel size to be appropriate according to the spacing used in the observation - metadata values of *mapReadoutSep* and *mapLineStep* - these reflect the angular separation of the scan lines as carried on the sky at the sampling requested with the observation. For earlier maps, i.e. when *mapReadoutSep* and *mapLineStep* are not present in the product metadata, different estimates are used depending on what information can be found in the metadata, such as the structure and number of datasets within the HTP, and the metadata parameters *supersampling*, *n\_cycles* for OTF maps, and *crossStep* for DBS raster/cross maps. In the most sparse information case, the task will assume the beam size is given by  $75.44726 * \text{wavelength}[\text{mm}]$  (where wavelength is the LO Frequency) and take (beam width/2) for half-beam spacing and (beam width/2.4) for Nyquist sampled maps.

You can specify the pixel size by editing the *pixelSize* boxes in the GUI, units are arcseconds. Note that if you do this then the filter parameters will also be updated to reflect the new pixel size. However, editing the filter parameters does not affect the *pixelSize* parameter. Editing the beam width parameter does not affect the *pixelSize* parameter but entering a non-zero *flyAngle* (see [Section 15.3.5](#)) will in order that flux is conserved when rotating the map.

The *pixelSize* can be modified in the command line as:

```
# For a square pixel
cubesContext = doGridding(htp=htp, pixelSize=[16.36])[0]
# For a rectangular pixel
cubesContext = doGridding(htp=htp, pixelSize=[10.0, 20.0])[0]
```



### Warning

#### Flux Conservation in Spectral Cubes from Mapping Observations:

The `doGridding` task in the HIFI pipeline is responsible for convolving the spectral datasets acquired in an OTF or DBS Raster mapping observation into a spectral cube with a specified pixel scale, which by default should match how the scan lines and readout points within each line were spaced during the observation. The scheme of signal filtering and interpolation to put the data on the specified grid may affect the overall flux conservation, at a level which is low but you should be aware of. For example, the total signal summed from a spectral cube produced using a Gaussian filter over the datasets of a Nyquist-sampled OTF map is generally < 1% lower than the sum of the signal taken directly from the input datasets (the Level 2 HTP datasets) before convolution. A part, or all of this slight mismatch may be on the assumed versus actual beam shape at the observed frequency. If you wish to put the map on a coarser grid, effectively reducing the spatial resolution to a wider beam in order to match another observation, then the flux losses become more noticeable. Doubling the pixels sizes from their default (native map point spacing) can reduce the total flux by as much as 10%, accompanied by an increase in baseline RMS noise. The effect is more prevalent in OTF maps than DBS Raster, and in addition to deviations from ideal beam shape characteristics becoming more important, the filtering and interpolation method, and

parameter values can be influential. No changes to the `doGridding` algorithm are planned, and this issue applies to all HIPE versions.

For convenience, the beam widths per HIFI band and beam spacing for half-beam and Nyquist sampled maps are given in the table below.

Band	Beam Width (")	Nyquist spacing (")	Half-beam spacing (")
1a	43.5	18.4	22.0
1b	37.7	15.9	19.0
2a	33.3	13.9	17.0
2b	29.8	12.5	15.0
3a	27.3	11.4	14.0
3b	24.9	10.4	13.0
4a	22.5	9.4	11.0
4b	20.8	8.7	10.0
5a	19.6	8.0	9.0
5b	18.6	7.8	9.0
6a	15.2	6.3	8.0
6b	14.0	5.8	7.0
7a	12.8	5.3	7.0
7b	12.2	5.2	6.0

- The pixel size is multiplied by a smoothing factor, which is 1 by default. You can modify the smoothing factor by editing the `smoothFactor` box in the GUI or in the command line by:

```
cubesContext = doGridding(htp=htp, smoothFactor=[2.0, 2.0]) [0]
```

The convolution filter parameters are dependent on the smoothing factor and modifications to it will cause the filter parameters to be automatically updated, see [Section 15.3.6](#) below.

- By default, the `mapSize` parameter is set to 0 in order to let the algorithm choose the optimal map dimension. The map size is automatically calculated by `doGridding` based on the spatial information in the input spectra. The map size can be specified in units of pixels by entering values in the `mapSize` boxes on the GUI. If you wish to fix the map size, you must set `mapSize` to your choice (say [8.0, 7.0]), and then you must set `pixelSize`=(0.0, 0.0) and `(x, y) FilterParams`=(0.0, 0.0).

In the command line, this can be done as:

```
cubesContext = doGridding(htp=htp, mapSize=[8.0, 7.0]) [0]
```

Note that if the map dimension in pixels are specified, then unless it is also specified, the pixel size will be given by the area observed divided by the number of pixels specified in the map size parameter.

Additional information about map size is available in the HIFI User's Reference Manual under [do-GriddingTask](#) in *HIFI User's Reference Manual*.

## 15.3.2. ...to make cubes of combined H- and V- polarisation

At the moment, the `doGridding` task only works with one HTP at a time. However, you may wish to create maps of combined H and V polarisation in order to increase signal to noise or to be able to compare with HSpot noise predictions, which assume that both polarisations are combined.

This can be done using the `mergeHtps` task, as follows:

```
# Get the HTPs of H and V data, here we use Level 2 WBS data from two polarisations
  of an observation (obs)
htpv=obs.refs["level2"].product.refs["WBS-V-USB"].product.copy()
htph=obs.refs["level2"].product.refs["WBS-H-USB"].product.copy()
#
# Merge the HTPS
htps = [htpv, htph]
mergedHtp = mergeHtps(htps=htps)
#
# Create the cube of merged data
cubesContext = doGridding(htp=mergedHtp) [0]
```

The resulting cube is labelled with the name of the first HTP passed to `mergeHtps` so in the example above, the resulting cube has the name `cube_WBS_V_USB_1`. Generally, the `mergeHtps` task attempts to merge metadata in a sensible fashion, and the details can be found in the [mergeHtps](#) in *HIFI User's Reference Manual* URM entry.

Note that differences may be seen in H and V profiles, as a consequence of the beam separation between the H and V polarisations, and of structural and/or velocity variations in your source. If you are particularly interested in the spatial structure of your source you may prefer not to average the H and V polarisations together. More detailed information is provided in the HIFI AOT Observing Mode Release and Performance Notes v3.0 (24 September 2011), available from the [HIFI instrument and calibration web pages](#).

The task `mkRms` allows you to assess the statistics of your new combined cube compared to the original cubes and HSpot noise predictions.

The `mergeHtps` task can be used to combine any number of HTPs and so can be used to create combined maps of data from different observations at the same frequency.

## 15.3.3. ...to make cubes for Solar System Objects

From HIPE 10 onwards, cubes of Solar System Objects (SSOs) produced by the standard pipeline are created with co-moving coordinates, i.e., the map centre follows the moving target. Cubes for non-SSO targets are automatically centred on the centre of the area observed. You can change these defaults using the `comoving` parameter, which has three possible options:

- True: creates the map in comoving coordinates
- False: creates the map centred on the centre of the area observed (appropriate for non-SSO objects)
- None (default): for SSO, creates the map in comoving coordinates - for non-SSO, creates the map centred on the centre of the area observed

To, for example, force creation of a map centred on the centre of the area observed (this was previously the default approach) select `comoving = False` in the `doGridding` GUI. To do the same in the command line, use:

```
cubesContext = doGridding(htp=htp, comoving=False) [0]
```

You may wish to view the positions in your maps as offsets (or relative positions) rather than in the absolute coordinates that are offered in the Spectrum Explorer. This can be done using the `doOffset` task, see [Chapter 7](#).

## 15.3.4. ...to make cubes more efficiently (limiting data input)

- By default, `doGridding` will create cubes for each subband. However, if you know (from inspection of Level 2 HTPs) that you are only interested in a subset of subbands, you can specify these in the comma separated list in the `subbands` box of the GUI. This will reduce processing time and memory usage.

A command line example to create cubes only for subbands 1 and 4:

```
cubesContext = doGridding(htp=htp, subbands=Int1d([1, 4]))[0]
#
# Extract the cube for subband 1
cube1 = cubesContext.refs["cube_WBS_V_USB_1"].product
# Extract the cube for subband 4
cube4 = cubesContext.refs["cube_WBS_V_USB_4"].product
#
```

The metadata of each cube will include a "subband" parameter stating the subband of the spectra which was used to compute the cube. This can be checked with:

```
print cube1.meta['subband']
```

- You can limit the spectral range of the cubes produced to cover only the region of interest by providing a range, per subband, for which cubes should be generated. The channel numbers must be input, and these can be read off the text to the bottom left of the plot in Spectrum Explorer while hovering the cursor over the spectrum.

The channel ranges can be entered in to the channels boxes in the GUI, by default the full channel range is already entered. Channel ranges can only be entered for those subbands selected in the subbands box at the top of the GUI, the non-editable boxes will turn red to indicate this.

The example below shows how to create a cube for the first and fourth subbands of a given spectrometer, reading just the channels 200 to 1200 in the first subband, and the channels 400 to 700 in the fourth:

```
cubesContext = doGridding(htp=htp, subbands = Int1d([1,4]), channels=[[200,1200],
[400,700]])[0]
```

- Finally, you can also create cubes for only a selection of datasets in the HTP. You do this by specifying the index of the dataset. Here we select subbands 2 and 4, and datasets 3, 4, and 5 from the HTP.

```
cubesContext = doGridding(htp=htp, subbands=Int1d([2,4]),
datasetIndices=Int1d([3,4,5]))[0]
cube_subband_2 = cubesContext.refs["cube_WBS_V_USB_2"].product
cube_subband_4 = cubesContext.refs["cube_WBS_V_USB_4"].product
```

To use the GUI, create the variable `datasetIndices` using the format in the example above and drag it from the Variables View into the `datasetIndices` bullet.

## 15.3.5. ...to make a rotated map or use a different WCS

- In the HTP metadata, you will find the parameter `pattAngle` (for pattern angle) which corresponds to the *Position Angle* was provided in HSpot. If the `pattAngle` is non-zero, then `doGridding` calculates a parameter called `flyAngle = 180 - pattAngle` (units of degrees).

You can apply a rotation angle to the map yourself using the `flyAngle` parameter. For example,

```
cubesContext = doGridding(htp=htp, flyAngle=50.0) [0]
```

Note that there is also a `HifiUplink` parameter called `flyAngle` and this is not the same quantity as the `flyAngle` created by `doGridding`. It is, unfortunately, possible to find the parameter `flyAngle` with several values in your data. For example, in the case of the observation 1342201689, which is an OTF map carried out with a rotation angle of 65 degrees, you find the following:

- **HifiUplinkProduct:** `flyAngle = 65 deg`

`pattAngle` takes the same value as `flyAngle` in the `HifiUplinkProduct`

- **cube headers in cubesContext:** `flyAngle = 0 deg`

Recall that in the case a map was carried out with a non-zero rotation angle, then two sets of cubes are generated at Level 2.5 by the SPG; one set is not rotated and this is contained in `cubesContext`, while the rotated cubes can be found in `cubesContextRotated`. But if you use the `doGridding` task on an `htp`, you will only have one sets of cubes generated i.e. with the `flyAngle` applied. Therefore, if you wish to recreate the native orientation on the sky, you need to set `flyAngle=0.0`.

- **cube headers in cubesContextRotated:** `flyAngle = 115 deg`

`doGridding` calculates `flyAngle = (180 - 65) degrees`.

- By default, `doGridding` applies a generalised least squared (GLS) projection of the input spectral coordinates into a flat map. You can also supply your own WCS to perform this projection. To create a WCS in HIPE, see [Defining and using the World Coordinates System \(WCS\)](#) in the Herschel Data Analysis Guide. Your WCS, for example called `my_wcs`, is passed to `doGridding` by:

```
cubesContext = doGridding(htp=htp, wcs=my_wcs) [0]
```

To use the GUI, you should pass the `my_wcs` variable to the `wcs` bullet.

## 15.3.6. ...with a different convolution

- By default the convolution is performed with a Gaussian filter function for OTF maps and Nyquist or smaller sampled raster maps, while a box filter is used for greater than Nyquist sampled raster maps. A box filter is the most appropriate choice for a raster map. However, in the case of small (less than 4x4) Nyquist sampled raster maps, a box filter which has length `BeamSize/PixelSize` or 2.4 times the pixel size, will result in a map with pixels at all the same values - the average of all the pixels in the map. For larger Nyquist sampled maps you may like to experiment with a box filter.

The filter type can be selected from the drop-down menu in the GUI or in the command line as:

```
# box filter
cubesContext = doGridding(htp=htp, filterType="box") [0]
# or Gaussian filter
cubesContext = doGridding(htp=htp, filterType="gaussian") [0]
```

- The parameters characterising the convolution filter are automatically computed by the task, and depend on the beam size, the pixel size, and the smoothing factor. If you change any of these parameters, the optimum filter parameters are recalculated by the task, and you can see the update occurring in the GUI. Note, however, that the reverse is not true if the filter parameters are modified. In case of doubt, you can set the filter parameters to zero to force the task to auto-compute the optimal values.

While we note that it is more intuitive to change the smoothing parameters to alter the convolution than the filter parameters, there may be cases, for example to compare a HIFI map to other data with an unusual sampling, that you would wish to do so. The filter parameters are given in pixel lengths and can be set using `xFilterParams` and `yFilterParams`. The parameters can be entered directly into the GUI, or set in the command line as demonstrated in the examples below.

The box filter is defined by lengths (in pixels) in the x (RA) and y (Dec) directions. The Gaussian filter is defined by the length (or influence area), which is a +/- value rather than a total range, the sigma of the Gaussian function multiplied by the square root of 2, and the order of the exponent. The default value for the length of the Gaussian filter is given by 3 times the kernel beam size divided by the pixel size, i.e. 0.9 times the telescope beam size divided by the pixel size ([Section 15.4.1](#)).

Example 1: a box filter with lengths x=0.5 pixels in the x (RA) and y=1.5 pixels in the y (Dec) directions:

```
cubesContext = doGridding(htp=htp, filterType="box", xFilterParams=[0.5],
  yFilterParams=[1.5]) [0]
#
# Or you can wrap the x and y parameters into one
parameters = [Double1d([0.5]), Double1d([1.5])]
cubesContext = doGridding(htp=htp, filterType="box", filterParams=parameters) [0]
```

Example 2: specify the parameters length and sigma of the Gaussian filter function:

```
# the "influence area" is the area surrounding a grid point
# where the algorithm must pick up all the available data points.
influence_area = 1.95 # length in pixels
# sigma of the gaussian function times SQRT(2)
sigma_sqrt2 = 0.3 # in pixels
xFilterParameters = Double1d([influence_area, sigma_sqrt2])
yFilterParameters = Double1d([influence_area, sigma_sqrt2])
# default case: influence_area = 1.8; sigma_sqrt2 = 0.36

cubesContext = doGridding(htp=htp, filterType="gaussian", xFilterParams=
  xFilterParameters, \
  yFilterParams = yFilterParameters) [0]
```

- The `doGridding` task assumes an equal weighting during the convolution, as the integration time per point is equal. In the `weightMode` drop-down menu in the GUI this option is labelled as 'all same weight'. In the command line, this option is selected by:

```
cubesContext = doGridding(htp=htp, weightMode="equal") [0]
```

You can also choose to use weights already present in the data, which may have been set by the pipeline or by you, by choosing 'read weights column from dataset' from the drop-down `weightMode` menu in the GUI. To do the same in the command line, use:

```
cubesContext = doGridding(htp=htp, weightMode="selection") [0]
```

- If you wish to learn more about the details of the convolution, you can check the `detail` box in the GUI, or set

```
cubesContext = doGridding(htp=htp, detail=True) [0]
```

This creates an output called *convolutionTable*, which can be viewed with the *Dataset Viewer*. The metadata of the table gives information about the map, beam, and pixel sizes (but note that the x values are negative because the x-axis values increment from right to left, while the y-axis values increment from bottom to top), and also information about the filter used in the convolution and the position of the reference pixel.

The *convolutionTable* itself contains the following information for each spectrum that contribute to a given pixel:

- The pixel number, as `ypixel` and `xpixel`. The number of spectra contributing to a given pixel depends on the gridding parameters used and the spacing of the readout points on the sky.
- The `latitude` and `longitude` of the pixel centre in decimal degrees.
- `vm`, `v`, `vp`, are the offsets of the bottom, centre, and top of the y position of the pixel from the map centre, while `um`, `u`, `up` are the same for the x position of the pixel.
- `PointSpectrum_id` gives a reference to the location of the spectrum in the HTP. For example, "row 0: box\_1: container 13" refers to the first (labelled 0.0) `PointSpectrum` of `SpectrumDataset 0014` in box 1 of the HTP. Unfortunately, that was not a typo: the `ConvolutionTable` labels `SpectrumDatasets` start from 0, while in the `ContextViewer` `SpectrumDatasets` are labelled from 1.
- `dv` and `du` are the distance of the spectrum (in pixels) from the pixel centre in the y and x directions, while the Pythagorean distance is given by `distance` (in pixels) and `dist_beams` (in beam size).
- If a spectrum is not included in the convolution as a consequence of a very low weighting then the `blanked` column will show "true".
- `du_beams` and `dv_beams` give the distance of the spectrum in the x and y directions from the pixel centre in antenna beam sizes.
- `filter_du` (`filter_dv`) is the value computed by the x-axis (y-axis) filter for the distance of the spectrum to the pixel centre, while `filter_value` is the value computed by the filter for this element, combining the filter functions along both axes.
- `spectrum_latitude` and `spectrum_longitude` give the position of the spectrum in decimal degrees.
- The `spectrum_flux` column contains the flux of the contributing spectrum.
- The weights of each channel of the spectrum are given in the `spectrum_weights` column.
- The spectra within the HTP are sorted according to their coordinates in horizontal (x) and vertical (y) directions prior to gridding. The final three columns in the `ConvolutionTable` refer to the indices after sorting of the first and last spectra in the current strip as well as the index of the spectrum.

### 15.3.7. ...to specify the map centre

The `doGridding` task computes the map centre as the centre of the coordinates of the input spectra. The pixel that the map centre falls in is called the *reference pixel*. It is possible to specify the reference pixel, in either pixel coordinates or in celestial coordinates, to `doGridding`. This may be useful in the case that you know the coordinates of the map centre to higher accuracy than the Herschel-HIFI Absolute Pointing Error (2").

- When specifying the reference pixel in pixel coordinates it is important to bear in mind that the (0,0) pixel is the bottom-most, left-most pixel in the cube. In contrast, the header information of the cube follows normal FITS convention in which the bottom-most, left-most pixel is (1,1).

For example, if you want to force the map centre to be the pixel (3.5, 4.0), then the `refPixel` input will be `Double1d([3.5, 4.0])`. This means that in the cube header the values of CRPIX1 and CRPIX2 will be (4.5, 5.0) and the centre of the coordinates of the input spectra will be located at the pixel (3.5, 4.0).

```
cubesContext = doGridding(htp=htp,refPixel = Double1d([3.5, 4.0])) [0]
```

To use the GUI, you will need to create a variable `refPixel` as above and drag it to the `refPixel` bullet.

- To specify the celestial coordinates of the map centre, you can enter the map position in decimal degrees directly into the `refPixelCoordinates` box in the GUI. Here is a command line example:

```
longitude = 307.9
latitude = 40.36
cubesContext = doGridding(htp=htp,refPixel=Double1d([0,0]),
  refPixelCoordinates=Double1d([longitude, latitude])) [0]
# or more compactly,
refPixel = Double1d([0,0])
refPixelCoordinates = Double1d([longitude, latitude])
cubesContext = doGridding(htp=htp,refPixel=refPixel,
  refPixelCoordinates=refPixelCoordinates) [0]
```

You can check that the cubes generated have the same `crval1` (longitude) and `crval2` (latitude) as the map centre you specified in the following way:

```
# Value returned: True or False
print cubesContext.refs["cube_WBS_V_USB_2"].product.wcs.crval1 == longitude #
True
print cubesContext.refs["cube_WBS_V_USB_2"].product.wcs.crval2 == latitude #
True
```

Of course, it is possible to set the map centre such that the defined grid falls partially, or completely, outside of the observed region. In that case, `doGridding` will not fail but will fill the pixels in the unobserved regions with NaNs.

## 15.3.8. ...with selected data types

The default action of `doGridding` is to take the science datasets, i.e. those datasets that are on source, and with the LO setting required to observe the frequency of interest. You can also select data to be gridded according to any data type you can see in the summary table of the HTP. This is particularly important if you choose to process the reference spectra (OFF positions) found in the product *Calibration* -> *pipeline-out* to allow you, e.g., to correct for any emission contamination in the OFF positions.

For example to make a grid of the hot/cold load observations:

```
cubesContext = doGridding(htp=htp, datasetType="hc") [0]
```

To include the OFF positions (found in *Calibration* -> *pipeline-out* -> *ReferenceSpectra*), which are ignored by default, in the gridding with:

```
cubesContext = doGridding(htp=htp, ignoreOffs=False) [0]
# In the GUI, you just deselect the ignoreOffs button
```



In practice, the Level 2 HTP contain only science data and these options are not expected to be used by astronomers unless the `doCleanUp` step of the Level 2 pipeline is omitted. The option to select `dataType="other"` may be of use for calibration scientists studying engineering mode observations.

### 15.3.9. ...to deal with NaNs

On occasion, you may see 'Not a Numbers' (NaNs) at the edge of the cube. This is not a problem with computation but a pixel that is flagged as containing spectra that have a very low weighting. Typically, this is found in maps displaying a strong zig-zag effect, and it is because the data arise from too far away from the map area. You can avoid this to some degree by using the *extrapolate* option:

```
cubesContext = doGridding(htp=htp, extrapolate=1) [0]
```

Furthermore, you can adjust the threshold at which pixels are flagged at, the default is  $1e-5$  but higher values may help remove the NaNs.

```
Configuration.setProperty("hcss.hifi.dp.otf.filter.threshold", "1e-1")
cubesContext = doGridding(htp=htp) [0]
cubes = cubesContext.refs["cube_WBS_V_USB_1"].product
```

## 15.4. doGridding in Detail

### 15.4.1. Particulars of Convolution

The gridding and convolution technique used is very similar to the one used in CLASS. The convolution is performed with a Gaussian filter function (recommended for OTF maps), where the Gaussian function is defined as:

$$\exp -(x^2/2\sigma^2)$$

and where the FWHM of the antenna beam B is:

$$2\sqrt{(2\ln 2)} \sigma_B$$

The final beam, Bf, is equal to

$$\sqrt{(\text{Kernel}^2 + B^2)}$$

Using a kernel of  $0.3\#B$  then  $B_f = CB$ , where  $C = 1.0440306508$  i.e. the final beam is 4.4% larger than the normal beam. We loose a bit of the resolution in order to have a better reconstruction of the map.

If the smooth factor (Sm) is  $> C$  then the kernel becomes equal to:

$$B \sqrt{(\text{Sm}^2 - 1)}$$

and the filter is Gaussian where

$$\sigma = \text{Kernel} / (2\sqrt{2\ln 2} \times \text{PixelSize})$$

[Figure 15.2](#) shows an example where we have three beam positions, 1, 2 and 3 (red circles) and where we find, in the influence area (blue square), beams 1 and 3. We define: 1] the influence area (also called the length) of the filter as  $(3\#\text{Kernel} / \text{PixelSize})$ . It is the area surrounding a grid point where the algorithm must pick up all the available data point, 2] Dx and Dy as the distances from the centre of the beam to the centre of the influence area, measured along their respective x and y axis, and 3] the filter parameter for each of the x and y:

$$V_x = \exp(-(D_x/\sigma)^2) \text{ and } V_y = \exp(-(D_y/\sigma)^2)$$

The filter final value, V, is  $V_x\#V_y$ .

The `doGridding` task offers two options to determine the weights: *all same weight* and *read weights column from dataset*. The final weight value  $W$  at the centre of the influence area is  $(W1V1+W3V3)$ . For option *all same weight*,  $W1=W3=1$  and the Cube Weight array will contain only the weights computed when building the map. For option *read weights column from dataset*,  $W1$  and  $W3$  are the values of the weight of the related spectra at that specific frequency. Thus, the Cube Weight array will contain a mix of the weights found in the htp and the weights computed when building a map. Finally, the final flux  $F$  at the centre of the influence area is  $(F1V1W1+F3V3W3) / (V1W1+V3W3)$ .

The box filter is also based around a kernel and is defined in pixels in the  $x$  and  $y$  directions where  $x$  and  $y$  can be either of equal length to create a square box, or of different lengths to create a rectangular box. The box filter (recommended for raster maps) may be more appropriated for your data (see [Section 15.3.6](#)). For the box filter, the values  $V$  are a constant function. The effect is equivalent to averaging if the observed points are *close* e.g. the value in  $\text{pixel}_1 = (V1*1+V3*1)/(1+1)$  and the value in  $\text{pixel}_3 = (V3*1+V1*1)/(1+1)$ . Because  $V$  is a constant and because it does not depend on the distance between the 2 pixels,  $\text{pixel}_1 = \text{pixel}_3$  if they are in the same influence area. Similarly to the Gaussian filter, the weight value  $W (=W1V1+W3V3)$  will be computed using one of the two options: *all same weight* with  $W1=W3=1$  and *read weights column from dataset* where  $W1$  and  $W3$  are the values of the weight of the related spectra at that specific frequency. The flux in the influence area is also calculated as  $(F1V1W1+F3V3W3) / (V1W1+V3W3)$ .

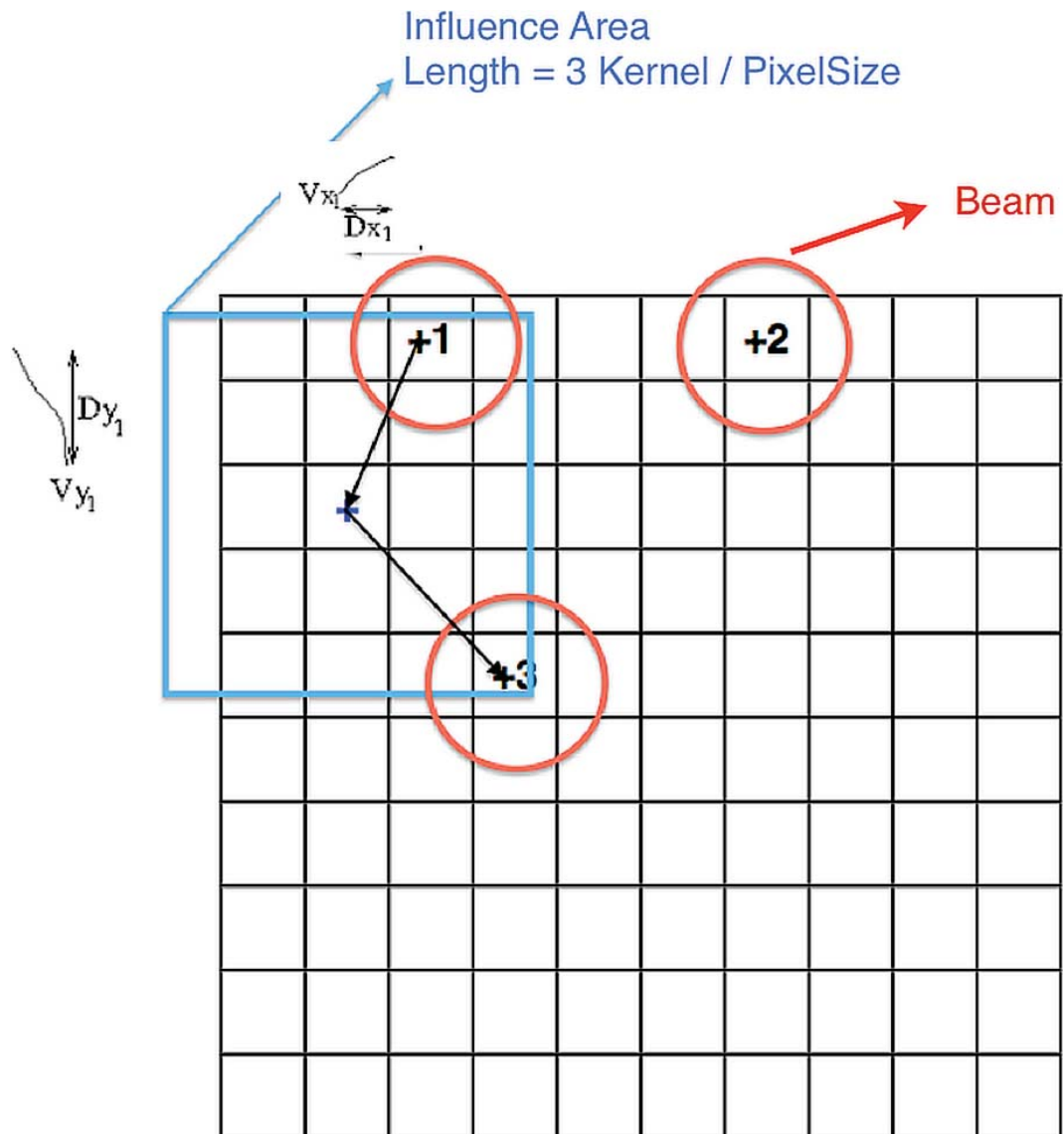


Figure 15.2. Gaussian Filter

## 15.4.2. Using the Gridding task with the Spectrum Toolbox

Another task, called `Gridding`, is available to make cubes of images. Like `doGridding`, it creates a cube by performing a spatial regridding of the input spectra onto a regular grid. However, unlike `doGridding`, it works with any dataset or product that implement the `SpectrumContainer` or interface. This means that the `Gridding` task is not limited to working with HTP and can be used instead for more general (to all three Herschel instruments) `Spectrum1d` and `Spectrum2d`. Using the `Spectrum Toolbox`, you may also create your own collection of datasets to pass to the `Gridding` task.

### The Gridding task and the Spectrum Toolbox.

You can make use of the spectrum selection tools of the spectrum toolbox to perform any selection of spectra followed by the usage of the `Gridding` task to create a cube for each segment of the spectra in these selections.

The following example shows how to combine `SelectSpectrum` with the `Gridding` task:

```
# first, create an instance of the SelectSpectrum task
selector = herschel.hifi.pipeline.util.tools.SelectSpectrum()
# use SelectSpectrum to get a single HifiSpectrumDataset with the spectra that
# fulfill certain criteria
# e.g. here one selects those spectra where its containing dataset has bdtype equal
# to 6022.
selected = selector(htp=htp, selection_lookup={'bdtype':[6022]},
  return_single_ds=Boolean.TRUE )
# one might have a glance at the spectra in the "selection" dataset e.g. in the
# TablePlotter
cube = gridding(selected)
cubes = gridding.cubes
```

### Making a SpectralSimpleCube with the Gridding task.

```
#-----
# make a dataset with all the spectra from all the science datasets
# (isLine == true => bdtype == 6022)
#-----
selector = herschel.hifi.pipeline.util.tools.SelectSpectrum()
selected = selector(htp=htp, selection_lookup={'bdtype':[6022]},
  return_single_ds=Boolean.TRUE )

scienceOnIndices = htp.summary['isLine'].data.where(\
  htp.summary['isLine'].data == Boolean.TRUE)
bbid = htp.summary['Bbid'].data[scienceOnIndices]

#another way of selecting...

selected = selector(htp=htp, \
  selection_lookup={'bdtype':bbid[0]}, \
  return_single_ds=Boolean.TRUE )

#-----
# the Gridding task that can work with any SpectrumContainer or collection of
# SpectrumContainers, like the selected above
#-----

cube = gridding(container=selected)

#get a point spectrum from this selection,
ds_spectrum = selected.getPointSpectrum(1)
ds_segment = ds_spectrum.getSegment(3) # read its third subband : get
SpectralSegment.
plotSegment = PlotXY(ds_segment.wave,ds_segment.flux,xtitle='Frequency
(MHz)',ytitle='Intensity')
```

```

#-----
# now let's play with the result cubes
#
#
# each cube is a SpectralSimpleCube which in its turn is an SpectrumContainer
# hence we profit from all the spectrum toolboxes: arithmetics, statistics, etc.
# and we can e.g. directly obtain a point spectrum as for any other
# SpectrumContainer
#-----

row = 0; column = 10;
spectrum = cube.getPointSpectrum(row,column)

print spectrum.getLongitude()
print spectrum.getLatitude()

print spectrum.segmentIndices
# you can check the cube, hence its spectra has a single "segment" or subband

segment = spectrum.getSegment(0)
# or...
segment = spectrum.getSegment(spectrum.segmentIndices[0])

plotSpectrum = PlotXY(segment.getWave(),segment.getFlux(),xtitle='Frequency
(MHz)',ytitle='Intensity')

# There are several ways to visualise a cube such as
# the CubeSpectrumAnalysisToolbox:

cat = CubeSpectrumAnalysisToolbox(cube)

# you can also visualise it with the SpectrumExplorer,
# since the cube is an SpectrumContainer

# or simply display it as a cube of images:
display = Display(cube)

```

### Optional inputs for the Gridding task.

Most of the optional inputs of the `doGridding` task are also applicable to the `Gridding` task namely: `weightMode`, `filterType`, `mapSize`, `refPixel`, `refPixelCoordinates`, `pixelSize`, `smoothFactor`, `filterType`, `filterParams`, `detail`, and the input `Wcs`.

In addition, there are other optional inputs which are specific to this task, namely `container` and `containerBox`.

# Chapter 16. Undoing the application of sideband gains

Last updated: 19 May, 2012

## 16.1. Introduction

HIFI is a double sideband (DSB) instrument. In consequence, application of sideband gains by the `doSidebandGain` step of the Level 2 pipeline mean that the detected lines are calibrated to single sideband (SSB) scale, while the continuum contains contribution from both sidebands and remains at the DSB scale. If the sideband gains are unity then the DSB scale continuum is twice the SSB level.

However, at the low end of band 2a and in bands 5a and 5b, non-unity sideband gains are applied, resulting in correct line calibrations but an incorrect continuum calibration that manifests as jumps in the continuum level. This can be problematic for absorption studies. Therefore, a task called `undoSidebandGain` is provided to allow you to undo the sideband gain correction, leaving you with a DSB continuum.

## 16.2. Using `undoSidebandGain`

To run the task you must first extract a Level 2 HTP from the Observation Context, here we extract the WBS-H-USB product, then the task is run on the HTP:

```
htp = obs.refs["level2"].product.refs["WBS-H-USB"].product
undoSidebandGain(htp=htp)
```

The `undoSidebandGain` is intended to be run on a Level 2 HTP, and will not run on data that has not had the Level 2 pipeline tasks, `doSidebandGain` and `convertFrequency`, applied to it. From HIPE 8 onwards, the sideband gain applied to the data has been written in the metadata fields `usbGain` and `lsbGain`. The `undoSidebandGain` task looks for these metadata and uses them to return the sideband gains to 0.5.

In the event that you have data that has been processed with a version of HIPE prior to HIPE 8.0, but using calibration with version `HIFI_CAL_6_0` (when non-unity sideband gains were introduced in band 2a) or more recent, then `undoSidebandGain` will find the side band ratio used in the `sidebandGainsIF` table in the calibration context, if the calibration context is passed to the task. These values will be used to restore the sideband gains to 0.5.

```
undoSidebandGain(htp=htp, calibration=obs.calibration)
```

If non-unity sideband gains have never been applied to your data then the task will do nothing and exit.

The true (SSB) continuum level can then be determined by halving the DSB solution.

# Chapter 17. Mathematical Operations on Spectra

Last updated: Nov 12, 2014

## 17.1. Introduction

Mathematical operations on spectra are carried out using the tasks in the *Spectrum Toolbox*. While viewing a spectrum in the Spectrum Explorer, you can open up the Spectrum Toolbox by clicking on the crossed hammer and spanner icon in the Spectrum Explorer button bar. You can then select the task you want from the drop-down menu in the panel that appears to the right of the spectrum. The task GUI and the Spectrum Explorer interact so that you can select points and ranges, or even several spectra for use in the task. Alternatively, the Spectrum Toolbox tasks can be run in the command line.

Most of the tasks in the Spectrum Toolbox are available for use on data from all Herschel instruments so the Data Analysis Guide contains the most complete information for spectrum arithmetics (the spectrum toolbox). Please see the [Spectrum Toolbox](#) section.

Here we give some HIFI specific examples of the Spectrum Toolbox and we describe some tasks that are only for use on HIFI data. The tasks in the Spectrum Toolbox can be used on subclasses of spectrum datasets (`Spectrum1d`, `Spectrum2d`) or on spectral cubes (`SimpleCube`) but not on products containing such data structures (e.g., `HifiTimelineProduct` (HTP)).

Operations on spectra include *Selection* and *Arithmetic Operations*.

- *Selection*: Provide means of selecting those spectra that can be combined. For instance cold-load spectra, ON spectra, etc. Selection can be applied to datasets, such as rows of a `Spectrum2d`, or to tables within a product, such as datasets included in a `HifiTimelineProduct`.
- *Arithmetic Operations*: Provide means of combining the selected spectra. This includes:
  - Basic arithmetic operations such as addition, subtraction, multiplication, or applications of scalar functions
  - Statistical operations such as mean, median, variance, standard deviation or percentiles for samples / selections of spectra
  - Data transformations such as smoothing or frequency re-sampling

Tasks that are available for use only with HIFI data are:

- `selectHifi`: a task for selecting datasets and/or rows of an HTP
- `DTFProduct`: a task to compute the discrete Fourier transform of a HIFI `SpectrumDataset`.
- `fold (DoFold)`: a task for folding frequency switched spectra (HTP)
- `stitch`: a generally available task but is particularly useful for stitching HIFI subbands, `doStitch` is used for HTP

## 17.2. Spectrum Toolbox HIFI Primer

We present the power of the toolbox with a few code examples.

### Stitching and Folding

Let's begin by loading a Point mode Frequency Switch observation into HIPE:

```
obs = getObservation(1342249401, useHsa=True, save=True)
```

After you have run the above line, you can access the data from your locally stored MyHSA pool (created automatically). This avoids the need to connect to the HSA and the observation can then be accessed in the following way:

```
obs = getObservation(1342249401, poolName="MyHSA")
```

Frequency Switch observations are stitched and folded in the Level 2.5 pipeline but you may wish to modify the default parameters used, or modify the data prior to stitching and folding. You can use the interactive pipeline to do these things but you can also run the tasks stand-alone on HTP, or on spectra (*SpectrumDatasets*). The examples below illustrate how to use the tasks using only the default parameters. To run the tasks on HTPs, use the `DoStitch` and `DoFold` tasks.

First extract some Level 2 HTPs.

```
# WBS-H-USB
htp_wbs_h = obs.refs["level2"].product.refs["WBS-H-USB"].product
# HRS-H-USB
htp_hrs_h = obs.refs["level2"].product.refs["HRS-H-USB"].product
```

Now stitch the subbands together:

```
# Stitch a copy of the HTP so you can compare the results with the original
htp_wbs_h_stitched=doStitch(htp=htp_wbs_h.copy())
```

HRS data can contain subbands which do not overlap so `doStitch` must be run with an additional parameter, `fillGaps`, to account for this. In the case of this particular observation, several of the HRS subbands lie at the same frequency so the `variant` parameter should also be set to stitch all subbands.

```
htp_hrs_h_stitched=doStitch(htp=htp_hrs_h.copy(), fillGaps=True, variant='stitchAll')
```

Now fold the stitched data, again the example shows how to work on a copy of the data.

```
htp_wbs_h_folded=doFold(htp=htp_wbs_h_stitched.copy())
htp_hrs_h_folded=doFold(htp=htp_hrs_h_stitched.copy())
```

To perform the same operations working on *SpectrumDatasets*, use the `stitch` and `fold` tasks.

```
# Extract SpectrumDatasets
ds_wbs_h = htp_wbs_h.refs["box_001"].product["0001"]
ds_hrs_h = htp_hrs_h.refs["box_001"].product["0001"]
#
# Stitch
ds_wbs_h_stitched=stitch(ds=ds_wbs_h.copy())
ds_hrs_h_stitched=stitch(ds=ds_hrs_h.copy(), fillGaps=True, variant='stitchAll')
#
# Fold
ds_wbs_h_folded=fold(ds=ds_wbs_h_stitched.copy())
ds_hrs_h_folded=fold(ds=ds_hrs_h_stitched.copy())
```

### HIFI specific selection task: `selectHifi`

The `selectHifi` task allows you to make selections from both HTPs and *SpectrumDatasets*, and for HTPs allows to select based on metadata. This provides a broader functionality than the `select` task in the *Spectrum Toolbox*, which in common with all the *Spectrum Toolbox* tasks, does not

work on HTPs. Furthermore, the `selectHifi` task can return a single `SpectrumDataset`. This allows you to operate the *Spectrum Toolbox* tasks on the contents of an HTP.

The full capabilities of the `selectHifi` task are described in the [selectHifi](#) in *HIFI User's Reference Manual* URM entry. Here, we give an example in which all the science data sets are extracted from a Level 1 HTP from the Frequency Switch observation used above, and a single `SpectrumDataset` is returned.

```
# Extract the WBS-H Level 1 HTP
htp = obs.refs["level1"].product.refs["WBS-H"].product
# Select the 'science' observations
science_11 = selectHifi(htp=htp, selection_meta={"sds_type":["science"]},
    return_single_ds=True)
```

If you look at `science_11` in the *Spectrum Explorer*, you will see that it contains a set of spectra containing a line, and a set of spectra that look like background. The second set of spectra are the observations taken on target but at the frequency switched LO setting. In order to select only the on-source, on-frequency data, you need to make a tighter selection based on the `Bbtype` (or `Bbid`) of that observation; these can be found by looking in the *Summary Table* in the HTP. We want to select datasets that are science datasets identified as `IsLine=True`, and these have `Bbid=6038`.

```
science_11 = selectHifi(htp=htp, selection={"bbtype":[6038]}, return_single_ds=True)
```

#### Fourier transform task: `DFTProduct`

The `DFTProduct` class can be used to calculate the Discrete Fourier Transform of a `HIFI SpectrumDataset` subband. This can then potentially be used to remove standing waves before using the class to perform the inverse Fourier Transform.

The `DFTProduct` class is described in more detail in the [DFTProduct](#) in *HIFI User's Reference Manual* URM entry. Here, we give a brief example of its usage using the WBS dataset extracted above; usage is identical for the HRS.

```
# Perform the Fourier Transform for the 4th subband of the WBS dataset
wbs_ft_4 = DFTProduct(ds_wbs_h, 4)
# Plot the phase and amplitude of the Fourier Transform
wbs_ft_4.plot()
```

#### Selection and other standard *Spectrum Toolbox* tasks

When digging into data, perhaps the most important thing to do is to be able to select out exactly the spectra you want to work on. This is possible with the `select` task, which allows you to select by:

- subband, or *segment*,
- index (the number of the row in the dataset) or by pixel coordinates (for a cube),
- or by a selection model that allows you to specify certain attributes, such as LO frequency or Chopper throw, or a range of values of an attribute.

Here, we give a few examples of selection, followed by some examples using other standard *Spectrum Toolbox* tasks. Unlike `selectHifi`, the `select` task will not work on an HTP so we first extract a `SpectrumDataset` from the Level 1 HTP used above to work on. We take the dataset with index 4, which is the first science dataset taken at the target LO frequency.

```
ds = htp.refs["box_001"].product["0004"]
```

Later, we will average together the rows in the selected datasets so first, resample `ds` so that all the spectra have the same frequency grid.



```
ds_resampled = resample(ds=ds, density=True, resolution=1.0, unit="MHz")
```

Using the `density` parameter set to 'True' as above forces the task to treat the flux as a flux density.

Now, let's try a few selections:

```
# Select only the 2nd subband
ds_2 = select(ds=ds_resampled, segments=[2])
# Select only rows with a Chopper column value of -5.536, to within a tolerance of
0.0001
ds_chop = select(ds=ds_resampled, selection={"Chopper":([-5.536], 0.0001)})
```

Now, average the spectra in `ds_2`:

```
ds_2_av = avg(ds=ds_2)
```

The selection and averaging can be done in one step:

```
ds_2_av = avg(ds=ds_resampled, segments=[2])
```

The simple arithmetic tasks have been overwritten with the usual key commands so you can use:

- '+' instead of add,
- '-' instead of subtract,
- '\*' instead of multiply,
- and '/' instead of divide

For example,

```
ds_plus = add(ds=ds_2_av, param=2.0)
```

and

```
ds_plus = ds_2_av + 2
```

are equivalent.

For more information about the *Spectrum Toolbox*, please see the [Spectrum Toolbox](#) section in the Herschel Data Analysis Guide.

# Chapter 18. Unit conversions

Last updated: 6 February, 2015

## 18.1. Converting to velocity and other frequency scales or frames

### 18.1.1. Changing frequency scale to USB, LSB, IF or velocity

The "x-axis" of spectra can be converted into other units (frequency, velocity, wavelength) using the general task, `convertWavescale`. The `convertWavescale` task uses the RADIO convention in converting to or from velocity.

There is also a task specifically for HIFI data, the `ConvertFrequencyTask`, which works on a `HifiSpectrumDataset` and on a `HifiTimelineProduct`. This can be used to convert between the USB, LSB, and IF scale and also to convert to velocity.

#### Converting Frequency Scales.

Assuming `spectrum` is the variable name for a `HifiSpectrumDataset` with the frequency scale of the data expressed as IF frequencies, you can convert to the lower sideband frequency scale as follows:

```
cft=ConvertFrequencyTask()
cft(sds=spectrum, to="lsbfrequency")
```

Of course, it is also possible to convert to the upper sideband. To achieve this, the keyword is "usbfrequency".

```
cft(sds=spectrum, to='usbfrequency')
```

To convert back to the IF, use:

```
cft(sds=spectrum, to='frequency')
```

The `ConvertFrequencyTask` works equally well on the `HifiTimelineProduct` itself. In this case, all the internal `HifiSpectrumDatasets` are converted. Note that this is not something you should do in the early stages (i.e. before Level 0.5) of the HIFI pipeline. For example, on a Level 1 `HifiTimelineProduct`:

```
cft=ConvertFrequencyTask()
cft(htp=hifitimelineproduct, to='frequency')
```



#### Note

Direct application of the `ConvertFrequencyTask` changes the data listed in the spectrum. Conversion back to the original IF scale is possible, just use the `to='frequency'` option.

#### Conversion to Velocity.

The `ConvertFrequencyTask` also works to convert the frequency scale to a velocity scale once given the reference frequency. By default, the RADIO velocity convention is used, but the OPTICAL and RELATIVISTIC conventions are available too.

```
cft=ConvertFrequencyTask()
cft(sds=spectrum,to='velocity',reference=576.268,inupper=False)
```

In the above example, you need to specify the reference frequency in GHz (the rest frequency of the line, for example). You also need to specify whether this reference frequency is for the upper (*inupper*=True) or lower (*inupper*=False) sideband. The 'inupper' parameter is always used, even for taking a spectrum already in USB or LSB frequencies to velocity; the default value is *inupper*=True. So if your line is in LSB and you go to velocity, be sure to explicitly set *inupper*=False.

### Summary.

to=	Description	Other keywords necessary
frequency	Converts to the Intermediate Frequency scale in MHz.	None
usbfrequency	Converts to the Upper side band Frequency scale in GHz.	None
lsbfrequency	Converts to the lower side band Frequency scale in GHz.	None
velocity	Converts to radial velocity (km/s) relative to reference frequency (RADIO convention)	reference=reference frequency (GHz), inupper=(True or False)
radio-vel	Converts to radial velocity (km/s) relative to reference frequency (RADIO convention)	reference=reference frequency (GHz), inupper=(True or False)
optical-vel	Converts to radial velocity (km/s) relative to reference frequency (OPTICAL convention)	reference=reference frequency (GHz), inupper=(True or False)
relativistic-vel	Converts to radial velocity (km/s) relative to reference frequency (RELATIVISTIC convention)	reference=reference frequency (GHz), inupper=(True or False)

## 18.1.2. Use of the Local Oscillator (LO) Frequency

Going from the IF scale to LSB or USB requires an LO frequency. To be explicit,

### Bands 1-5:

USB:  $SKY = LO + IF$

LSB:  $SKY = LO - IF$

### Bands 6-7:

USB:  $SKY = LO + F_{conv} - IF$

LSB:  $SKY = LO - F_{conv} + IF$

where  $F_{conv}$  is 10404.7 MHz (H-pol) or 10403.2 MHz (Vpol).

The `convertFrequency` Task demonstrated above makes use of the 'LoFrequency' data column in the spectrum. During the Level 1 pipeline, the `doVelocityCorrection` Task multiplies both the LoFrequency and `frequency_X` (where X is the subband number) columns by a Doppler factor to take them into the LSRk frame of reference (or the SOURCE frame for Solar System targets). If you add these columns together, you will have SKY frequencies already in the final frame of reference. So the meaning of the column 'LoFrequency' depends on the level in which it is found:

- **Levels 0 and 0.5**

Column 'LoFrequency' is the actual, instrumental LO setting.

- **Levels 1 and 2**

- **'science' datasets**

Column 'LoFrequency\_measured' is the actual, instrumental LO setting.

Column 'LoFrequency' is the LO Frequency multiplied by a Doppler factor to put it in the reference frame specified in metadatum freqFrame (SPECSYS)

- **non-'science' datasets**

Column 'LoFrequency' is the actual, instrumental LO setting.

- **Level 2.5**

For point modes, same as Level 2.

In mapping modes, the cubes don't have such columns, though there is a metadatum 'loFrequency', description 'The LO frequency of the source phase', which is the actual instrumental LO frequency. In addition, there are copies of the Level 2 products alongside the cubes, in which you will find the columns described for Level 2

In spectral scans, there are no such columns. However, the values from the Level 2 Doppler-corrected 'LoFrequency' column are copied into metadata with names such as 'loFreq1', 'loFreq2', etc.

The instrumental IF frequencies are not preserved after Level 0.5. They can be recreated most easily by repipeling with `doVelocityCorrection` turned off, or by using `doVelocityCorrection` and `convertFrequency`.

There are metadata, which are observing mode dependent, appearing throughout the observation context derived from these LoFrequency data columns.

Metadatum	Description	Comment
LoFreqAvg	Average LO frequency Doppler-corrected to freqFrame (SPECSYS)	Average of the LoFrequency column after doVelocityCorrection
loFrequency	Actual local oscillator frequency	For observations with a fixed instrumental LO
loFreqMax	Max LO frequency of the spectral scan Doppler-corrected to freqFrame (SPECSYS)	Doppler-corrected max LO frequency for Spectral Scan observations
loFreqMin	Min LO frequency of the spectral scan Doppler-corrected to freqFrame (SPECSYS)	Doppler-corrected min LO frequency for Spectral Scan observations
loFrequencyEnd	Actual end local oscillator frequency	Instrumental LO in Spectral Scan observations
loFrequencyStart	Actual start local oscillator frequency	Instrumental LO in Spectral Scan observations
obsFreqLsbMax	Observed max frequency for LSB in freqFrame (SPECSYS)	Max observed Doppler-corrected sky frequency for the LSB for Point and Mapping observations
obsFreqLsbMin	Observed min frequency for LSB in freqFrame (SPECSYS)	Min observed Doppler-corrected sky frequency for the LSB

Metadatum	Description	Comment
		for Point and Mapping observations
obsFreqUsbMax	Observed max frequency for USB in freqFrame (SPECSYS)	Max observed Doppler-corrected sky frequency for the USB for Point and Mapping observations
obsFreqUsbMin	Observed min frequency for USB in freqFrame (SPECSYS)	Min observed Doppler-corrected sky frequency for the USB for Point and Mapping observations
obsFreqMax	Observed max frequency in freqFrame (SPECSYS)	Max observed Doppler-corrected sky frequency for Spectral Scan observations
obsFreqMin	Observed min frequency in freqFrame (SPECSYS)	Min observed Doppler-corrected sky frequency for Spectral Scan observations

### 18.1.3. Changing frequency rest frame with doVelocityCorrection

By default, the Level 2 spectra are in the LSRk rest frame (or the source frame for solar system targets). The [doVelocityCorrection](#) Task can be used to change the rest frame to one of 'topocentric' (spacecraft), 'geocentric', 'barycentric' (close to heliocentric), 'lsrk', and 'source'. [Chapter 25](#) describes how these transformations are done.

For example, to transform all Level 2 products to the spacecraft rest frame:

```
level2 = obs.getLevel2()
refs = level2.refs
for ref in refs:
    http=level2.refs[ref].product
    doVelocityCorrection(http=http, targetFrame="topocentric")
```

Because of the way referencing works inside observation contexts, the original data are now changed. Along with the frequency axes, the 'LoFrequency' column is transformed to the spacecraft frame and now matches 'LoFrequency\_measured':

```
print http.meta['freqFrame']
{description="Standard of rest for spectral axis", string="topocentric"}
print http.refs['box_001'].product['0001']['LoFrequency_measured'].data -
http.refs['box_001'].product['0001']\
['LoFrequency'].data[0.0]
```

### 18.1.4. The meaning of velocities found in data and metadata

After application of `doVelocityCorrection`, three data columns appear in spectra: 'velocity\_hso\_1', 'velocity\_hso\_2', and 'velocity\_hso\_3'. These are the three components of the spacecraft velocity in the barycentric rest frame computed for the midpoint of the integration. To preserve HIFI frequency calibration accuracy, this velocity is necessarily better than 1 m/s accurate.

There are metadata read and written by `doVelocityCorrection`, some of which come from the original observing proposal.

Metadatum	Description	Comment
vlsr	"Velocity in the frame of reference"	From the proposal: source redshift
redshiftFrame	"Reference frame of redshift"	From the proposal: reference frame for source redshift
redshiftType	"Type of redshift: optical, radio, redshift"	From the proposal: definition of redshift ('vlsr') parameter
freqFrame (SPECSYS)	"Standard of rest for spectral axis"	dovelocityCorrection read/write: ref frame of associated dataset

There are some other metadata *not* used by HIFI, here listed for completeness:

Metadatum	Description	Comment
radialVelocity (VFRAME)	"Spacecraft velocity along the l-of-s of the telescope wrt the LSR"	This is computed by the <code>RadialVelocityTask</code> using the spacecraft orbital ephemeris for the mid-time of the observation (metadata (startDate + endDate)/2). Note the sign, it is positive if the spacecraft is moving toward the source. This value is used in the SPIRE and PACS pipelines to convert frequencies from topocentric to the LSR rest frame. Both use the RADIO convention: $\text{freq} = \text{freqRest} * (1 - \text{Vrad}/C)$
velocityDefinition (VELDEF)	"The velocity definition and frame"	Current value is 'RADI-LSR'. It reflects the way radialVelocity is used to compute LSR frequencies from topocentric.
frame	"Frame of reference for Vlsr"	Ignore it. It is a leftover from previous versions of the HSPOT proposal software.

## 18.2. Flux conversions

HIFI data can be converted from temperature scale to Jy using the `convertK2Jy` task. The task works for spectra in `ObservationContexts` (`obs`), `HifiTimelineProducts` (`htp`), `HifiSpectrumDatasets` (`ds`), and in `SimpleSpectrum` format (`spectrum`). The conversion can be done assuming a disk-like (top-hat), a Gaussian source morphology, or an ad hoc source model morphology. In the first two cases, a source diameter needs to be entered (`<size>`). The `convertK2Jy` task also allows you to convert data back from Jy to whatever temperature scale the data was originally on.

In the GUI, the `convertK2Jy` task is found under *Applicable* in the *Tasks* view panel when you have selected an `ObservationContext` in the *Variables* view panel. To open the `convertK2Jy` task GUI, double click on its name in the *Task* panel, the GUI will open with the `ObservationContext` you had selected in the *Variables* panel loaded into the `obs` parameter in the GUI. If you have selected an `HTP`, `SpectrumDataset` or `SimpleSpectrum` in the *Variables* panel, you can find the `convertK2Jy` task under *By Category* → *Hifi* or under *All* in the *Tasks* panel. You will need to drag the name of the variable containing the data you want to convert into the appropriate field in the GUI.

The simplest call to the task requires only the input data and the diameter of the source in arcsec (`size`):

```
MyConvertedObs = convertK2Jy(obs=obs, size=10.0)
```

By default, the output name from `convertK2Jy` is *result*. You can change this in the GUI by typing a different name into the *Variable name for result* field. In the command line, you can set the result name as you call the task, as was done in all the example above and other examples below.

All other parameters have default values, as described below. Depending on your science goals and your data, it is expected that you will to modify the defaults parameter values.

- **size**

As described above, for the `size` parameter, you should fill in the size of your source, in arcsec. The meaning of `size` depends on what you assume for your source geometry (see next bullet). For **shape=disk**, the `size` is the diameter of your source, while for **shape=gaussian**, the `size` is the FWHM of your source.

This is a mandatory parameter, so for a point source you must specify **size=0.0**.

Note that you must include a decimal point, so **size=10.** or **size=10.0** rather than **size=10**, in order for the task to work.

- **shape**

The conversion can be made assuming a disk (top-hat), or Gaussian morphology for the source. The latter is generally more appropriate for the semi-extended sources typically observed with HIFI. The default is to assume a disk shape.

To change the parameter in the GUI, type **gaussian** or **disk**. In the command line, it is used like this:

```
MyConvertedObs = convertK2Jy(obs=obs, size=10.0, shape='gaussian')
```

- **inputShape**

A user-provided source brightness distribution model can be provided by the user using this option. This option will only work if the option `hifiBeam` is selected, i.e. when the detailed 2-D HIFI beam is used. When such an input is provided, the source coupling factor (see below) is will be the same for each position contained in the input products, and is computed at the averaged position of the products passed in, see [Figure 18.1](#). For a mapping observation, this means that the same correction will apply to all positions, and this correction is computed from the coupling to the source model at the centre of the mapped area. This behaviour is similar to what is done when e.g. a simple Gaussian or Disk source model is used.

- **reverse**

The `convertK2Jy` task adds a new matadatum at the `SpectrumDataset` level called *temperatureScaleOrigin*, with the original temperature scale (T\_A\*, T\_A', T\_MB) as a value. This metadata item can then be used by the task in order to reverse the calculation back from Jy to the original temperature scale.

To use the `reverse` option, check the `reverse` box in the GUI. In the command line use:

```
MyObs = convertK2Jy(obs=MyConvertedObs, size=10.0, shape='gaussian',
reverse=True)
```

Note that you need to provide the same `size` and `shape` values as used in the conversion to Jy in order to get correct results.

- **overwrite**

The data in an `observationContext` will always be overwritten, in order to conserve memory. In the cases of `HTP`, `SpectrumDataset`, or `SimpleSpectrum`, you can choose not to overwrite the input data. In the GUI, do this by unchecking the `overwrite` box. In the command line, set `overwrite=False`.

- **cal**

If you pass an `observationContext` to `convertK2Jy`, it will automatically use the beam widths and beam efficiencies found in the Calibration Tree. If you want to use the beam parameters from the calibration tree to convert fluxes in an `HTP`, `SpectrumDataset`, or `SimpleSpectrum`, you need to pass the calibration tree directly to the task.

```
# Extract a calibration context from an observation context named obs
cal = obs.getCalibration()
# Pass the calibration to convertK2Jy
convertedHtp = convertK2Jy(htp=htp, size=10.0, shape='gaussian', cal=cal)
```

In the GUI, you can drag the calibration context from an `observationContext` viewed in the *Context Viewer* into the *Variables* panel. From there, drag the variable created to the `cal` bullet.

If you pass an `HTP`, `SpectrumDataset`, or `SimpleSpectrum` to `convertK2Jy` and do not specify a calibration context to use, the task will use hard coded values to calculate  $\eta_A$  and  $\eta_B$  for the conversion. See below for more details about the calculations carried out by `convertK2Jy`.

- **useInterpolation**

You can force `convertK2Jy` to use the hard-coded values to calculate  $\eta_A$  and  $\eta_B$  by checking the `useInterpolation` box in the GUI. In the command line, use `useInterpolation=True`.

The forward and beam efficiencies are calculated in the following way:

$$\eta_A = \eta_{A0} \times \exp[-(4 \times \pi \times \sigma / \lambda)^2]$$

$$\eta_B = \eta_{B0} \times \exp[-(4 \times \pi \times \sigma / \lambda)^2].$$

where,  $\sigma = 3.8$  micron,  $\eta_{A0} = 0.68$  and  $\eta_{B0} = 0.76$ , in bands 1-4, 6 and 7 and  $\eta_{A0} = 0.58$  and  $\eta_{B0} = 0.66$ , in band 5

The frequency used to determine  $\lambda$  in the expressions above is printed out to the console, along with the values found for  $\eta_A$  and  $\eta_B$ .

- **useLo**

You can choose whether the calculations are made using the LO frequency (or frequencies) in the observation, or the frequency of each spectrum with the `useLo` parameter. Using the LO frequency (`useLo=True`) is most efficient, and the recommended approach for all types of observations, except Spectral Scans. In the case of Spectral Scans, where the LO frequency changes for each spectrum at Level 2, it is recommended to use `useLo=False`. In the GUI, this option is toggled on and off via a check box.

- **tol**

You can limit (or increase) the number of calculations `convertK2Jy` does with the `tol` parameter. This is the delta frequency for the calculations of the forward and beam efficiencies so, for example, a `tol=0` will result in a calculation for every LO or spectrum frequency in the dataset. The default value is 0.1 and it takes the units the data are in. However, the default value is chosen assuming that Level 2 data is passed to the task.

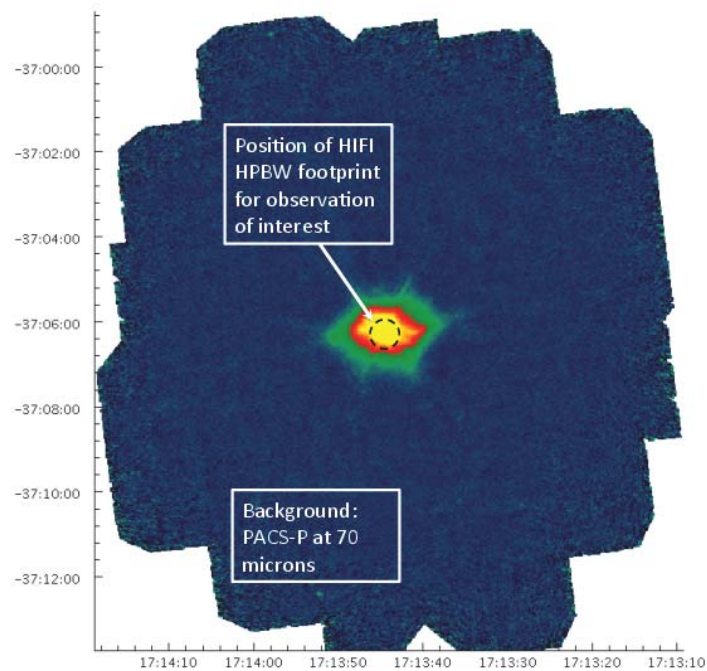
To change this parameter in the GUI, type a new value in the `tol` parameter field. In the command line, use `tol=your value`.



- **hifiBeam**

By default, `convertK2Jy` uses the values of the HIFI beam model introduced to the calibration tree in HIPE 13, and the parameter `hifiBeam` is set to `True`. If you prefer to use the old (Gaussian) model and efficiencies, you can set `hifiBeam=False`. In the GUI, do this by unchecking the tick box.

It is strongly recommended to use the new beam model, see the [release note](#) for more details. The possibility to use the Gaussian beam assumption is provided for backwards compatibility, and to allow the task to be used with data from another instrument (in which case the appropriate beam width and efficiencies must be passed manually to the task).



**Figure 18.1. Illustration of the beam coupling computation in case of a user-provided source brightness distribution model. In this case, the model is passed as a PACS continuum image (background image) and the corresponding position of the HIFI observations where it is computed is shown as a circular footprint of the size of the HPBW.**

The temperature values in the spectrum are converted by `convertK2Jy` from  $T_A^*$  or  $T_{MB}$  to  $T_A'$  using the forward and beam efficiencies available from the calibration tree or hard-coded values, as described above. The conversion to Jy is dependent on the size of your source, and is computed differently for point sources and extended sources.

- **Point source (size=0.0)**

$$\frac{S}{T_A'} \left[ \frac{Jy}{K} \right] = \frac{2k}{A_{geom} \times \eta_A}$$

where  $S$  is the Energy flux expressed in Jansky,  $T_A'$  is the Antenna Temperature measured in Kelvin,  $k$  is the Boltzmann's constant, and  $\eta_A$  is the aperture efficiency.  $A_{geom}$  is the effective area of the telescope, which we take to be 3.283 m, so  $A_{geom} = 8.465091 \text{ m}^2$ .

- **Extended source (size >0.0).**

For extended sources, assumptions about the HIFI beam and the source geometry become important and a flux dilution factor,  $K$ , is introduced into the expression above:

$$\frac{S}{T'_A} \left[ \frac{Jy}{K} \right] = \frac{2k}{(A_{geom} \times \eta_A)/K}$$

The flux dilution comes from the ratio of the beam and source solid angles. The following expressions are extracted from the HIFI beam model [release note](#), to which you are directed for a detailed discussion.

The source solid angle is given by:

$$\Omega_s(\lambda) := \int \int_{\Omega} \psi(\phi, \theta, \lambda) d\Omega$$

the beam-weighted solid source angle by:

$$\Omega_{\Sigma}(\phi_0, \theta_0, \lambda) := \int \int_{\Omega} P(\phi - \phi_0, \theta - \theta_0) \psi(\phi, \theta, \lambda) d\Omega$$

and the total integrated source flux by:

$$S^{tot}(\lambda) = \frac{2k}{\eta_A A_{geom}} \frac{\Omega_s}{\Omega_{\Sigma}(\phi_0, \theta_0)} T'_A$$

The computation of  $K$  depends on the assumption made about the HIFI beam profile.

- **HIFI beam model**

If the (non-Gaussian) beam model is employed (`hifiBeam=True`), the integrals above are performed numerically, using the source shape as specified by the `shape` and `size` parameters, and using the beam model extracted from the calibration tree. Note that due to the azimuthal symmetry of the source models (and the fact that central pointing is assumed), the azimuthal integral does not depend on the source. Equivalently, for our assumptions on the source, the 1D beam model can be used without loss of generality (see [Chapter 8](#) for more information about the 1D and 2D beam models). This is done by `convertK2Jy`.

- **Gaussian beam assumption**

If `hifiBeam` is set to `False` then `convertK2Jy` uses a less accurate Gaussian beam assumption. In this case, the integrals defining  $K$  can be performed analytically for the assumption of a disk or a Gaussian source geometry as follows:

- **shape=disk**

$$K = \frac{(1 - e^{-x^2})}{x^2}$$

- **shape=gaussian**

$$K = \frac{1}{\left(1 + \frac{x^2}{\ln(2)}\right)}$$

where,

$$x = \sqrt{\ln(2)} \times \frac{\text{source diameter}}{FWHM_{mainbeam}}$$

and

$$FWHM_{mainbeam} = \sqrt{4 \times \frac{\ln(2)}{\pi} \times \eta_B \times \frac{\lambda^2}{(\eta_A \times A_{geom})}}$$

with  $S$ ,  $T_A'$ ,  $\eta_A$ , and  $A_{\text{geom}}$  as above, and  $\eta_B$  is the main beam efficiency.

- **shape=user-provided**

$$K = \Omega_{\Sigma} / \Omega_S$$

# Chapter 19. Combining H- and V-polarisation Spectra

Last updated: 21 October, 2015

## 19.1. Introduction

You may wish to average H- and V-polarisation data together in order to improve the signal-to-noise. You *should* average H- and V-polarisation data if you wish to compare the signal-to-noise with the noise estimates provided by HSpot, which are calculated assuming that the Level 2 H and V spectra are averaged. H- and V-polarisation HIFI data do not have exactly the same frequency scale (there is a small offset) so a convolution should be made prior to averaging in order to calculate an accurate channel by channel average. The `polarPair` task does this for you.

Note, however, that differences may be seen in H and V profiles as a consequence of the beam separation between the H and V polarisations, and of structural and/or velocity variations in your source. If you are particularly interested in the spatial structure of your source, you may prefer not to average the H and V polarisations together, at the cost of signal-to-noise. More detailed information is provided in the HIFI AOT Observing Mode Release and Performance Notes v3.0 (24 September 2011), available from the [HIFI instrument and calibration web pages](#).

## 19.2. Using the polarPairs task



### Warning

The `polarPairs` task has been modified in HIPE 12 to enable it to be used as part of the pipeline and to bring its syntax in line with other Spectrum Toolbox tasks. The syntax used prior to HIPE 12, e.g.

```
pp_wbs = PolarPair(wbs_h, wbs_v)
wbs_av = pp_wbs.avg()
```

will not fail but can now produce incorrect results. The reason for this is still being investigated.

Be assured that `polarPairs` worked correctly in previous HIPE versions but it is recommended to use the new syntax from HIPE 12 onwards.

The `polarPairs` task can be used for all types of HIFI spectra and spectral cubes, and also for HIFI TimelineProducts (HTP). The task associates a pair of H- and V-polarisation spectra in order to compute their average or their difference. The algorithm used convolves one spectrum on the other, insuring a perfect channel-by-channel association without any spectroscopic resolution loss. The average and difference takes also into account channel weights.

The `polarPairs` task checks that each pair of H- and V-polarisation spectra have consistent LO frequency and pointing positions. This ensures, for example, that all the data in a spectral cube is not collapsed or that data at different LO settings are not averaged together in the case of Spectral Scans.

In the command line, the `polarPairs` task is used to calculate the average(s) in the following ways for *Spectrum Containers* (Spectrum Datasets and Spectral Cubes) and HTP, respectively:

```
average_spectrum = polarPair(ds1=spectrum_h, ds2=spectrum_v)
average_htp = polarPair(htp=htp_h, htp2=htp_v)
```

You can find the difference in the pairs of spectra in the following way:

```
average_spectrum = polarPair(ds1=spectrum_h, ds2=spectrum_v, difference=True)
average_ftp = polarPair(ftp=ftp_h, ftp2=ftp_v, difference=True)
```

The task can also be used via a GUI interface, which can be found in the *Tasks* panel under *Applicable* if you have an HTP selected in the *Variables* panel. In the cases of Spectrum Datasets and Spectral Cubes, you have to look under the *All* menu in the *Tasks* panel.

In the command line, it is also possible to compute the average (or difference) for a given segment (subband):

```
average_spectrum = polarPair(ds1=spectrum_h, ds2=spectrum_v)
# Get the average for just the third subband
av_subband3 = avg(ds=average_spectrum, segments=[3])
```

Note the following:

- In the case of HTPs, the first HTP passed to the task - the H polarisation in the example above - is overwritten with the average. If you want to retain the original HTP, you should work on copies.
- The result contains only changed fluxes. Header information (metadata) are not adapted. This remark is particularly relevant for the integration time given in the averaged spectrum.

### Task details

The order of the datasets or HTP determines the frequency grid of the result. In the examples above, the V-polarisation spectra will be convolved over the H-polarisation spectra. New V-polarisation spectra (called V\* hereafter) is calculated with exactly the same frequency grid as the H-polarisation spectra. V\* and H are then averaged or differenced.

Given a middle frequency  $v_0$  of a channel  $i$  in the H dataset, the corresponding V\* channel is given by:

$$V^* [i] = \sum_{j=0..N} V[j] * \exp(-(v[j]-v_0[i])^2 / 2 * \sigma[i]^2) / \sum_{j=0..N} \exp(-(v[j]-v_0[i])^2 / 2 * \sigma[i]^2)$$

where  $N$  is the number of channels in  $V$  and  $\sigma$  is set so that the FWHM of the Gaussian corresponds to the channel width in  $H$ .

Computing  $V$  over  $H$  takes  $N*N$  operations, which can be a lengthy procedure. As only channels near  $v_0$  significantly contribute to the V\* channel value, this sum can be reduced to a limited number of channels around  $v_0$  by setting a tolerance.

If  $H$  and  $V$  are shifted in frequency, a minimal tolerance is required for a correct convolution which is equal to the number of channels corresponding to the shift. However, if the tolerance is too small, the final averaged dataset may contain NaNs only. The default value of the tolerance is set to 100 but if you wish to modify this, you can do so with the `tolerance` argument. For example, below we change to tolerance to 140 channels.

```
average = polarPair(ds1=spectrum_h, ds2=spectrum_v, tolerance=140)
```



#### Note

If you wish to combine maps, you should use the task `mergeHtp` as it was specifically developed for combining maps (see [Section 15.3.2](#) for further details).

## Chapter 20. Fitting Spectra

Spectral fitting is described in the [Spectral fitting](#) chapter of the Herschel Data Analysis Guide.

When working interactively with the SpectrumFitterGUI, it is important to remember that the Spectrum Fitter will work only with a single SpectralSegment. HIFI data contain one SpectralSegment per subband; four for the WBS, and up to sixteen, depending on the setting used, for the HRS. Graphically, SpectralSegements can be plotted individually by clicking on only one box in the SpectrumExplorer selector panel.

You may find it easiest to [Stitch](#) together the HIFI subbands, and/or [Extract](#) the range of interest before using the SpectrumFitterGUI.

# Chapter 21. The HIFI line identification tool

Last updated: 24 November, 2015

## 21.1. Introduction

The `identifyLines` task allows you to identify lines in your spectrum and then run a comparison of known lines with a *linelist*.

Prior to using the task, you must have cleaned your spectrum (no fringes, no spurs, etc...). Please visit [Section 1.1.4](#) to learn more about the data reduction steps.

This chapter is divided in four sections:

- **Basic usage:** This is where you will find the minimum instructions you need to identify the lines in a spectrum.
- **Advanced usage:** If you want to get the maximum out of the line identification task.
- **A guided tour:** The line identification task comes with handy functionalities for handling, filtering, and visualising your lines. This section will teach you how to do that.
- **The exportLines task:** Once the `identifyLines` task terminates, you can then export the results using the `exportLines` task into a *SpectralLineList*.



### Note

In this chapter, we will use, as an example, a spectrum from Orion, more specifically, Orion South in band 1a between 485 and 490 GHz. We have made accessible, from the [HIFI Instrument and Calibration](#) Web page, the [spectrum](#) (in FITS format), the [CASSIS linelist](#) (for band 1a, in ascii format - therefore reproducible with other lines of your choice), and a [script](#) (please remove the extension '.txt' prior to using the script) to help you practice. You can thus eventually create your own spectrum and linelist.

### Initialisation

You will need to set the correct path to `line_identification_guide/data` using these command lines:

```
import os
from herchel.hifi.dp.tools.linelist import Linelist

#
# The path to the directory (here called 'line_identification_guide/data') where the
# Spectrum1d fits
# and the CASSIS linelist (of the known lines) files are stored.
#
INPUT_DATA_DIR = "yourpathdirectory/line_identification_guide/data"
```

## 21.2. Basic Usage

The `identifyLines` task is applied on a *Spectrum1d* and once selected, you will find `identifyLines` in the *Applicable Tasks* menu where you can open the GUI from there (see [Figure 21.1](#)). You can also open the GUI from the *HIFI* list of tasks under *By Category*. A table with the list and definition of each parameters can be found at the end of this section.

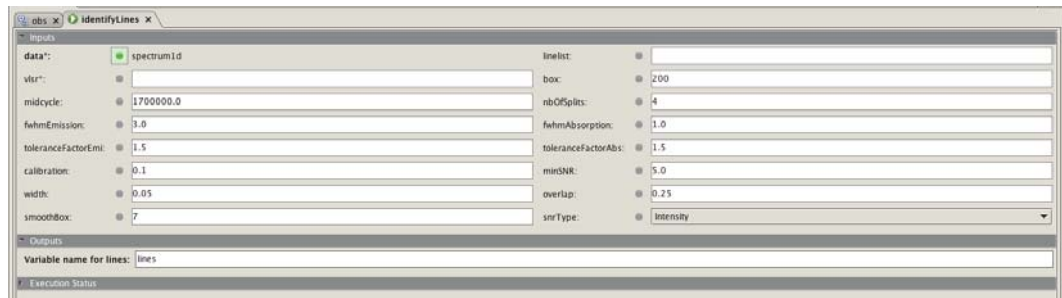


Figure 21.1. IdentifyLines task GUI

The `identifyLines` task can also be used with a series of command lines. The rest of the chapter will make use of the command lines.

In order to identify the lines in a spectrum you need at least 4 components:

- a **spectrum**: the *SpectrumId* for which we want to identify the lines (see [Figure 21.2](#))
- a **linelist**: the path to the [CASSIS](#) linelist file used as the known lines. Please note that HIPE has a linelist in the build and can be used by default (leave the parameter *linelist* unset).
- a **vlsr**: the VLSR of the source in km/s
- **identifyLines**: the task for the line identification

The first three components are the data, and are called as follow:

```
## Instructions on how to obtain the spectrumId, the lineList, and the script used
## as an example
## in this chapter are found in the Section 21.1.

# The SpectrumId fits file
spectrum = fitsReader(os.path.join(INPUT_DATA_DIR,
    "Orion_S_baselined_485_490.fits"))

# The CASSIS linelist of known lines
linelistFile = os.path.join(INPUT_DATA_DIR, "16293_SIS.txt")

# The VLSR (in km/s), here, specifically for Orion
vlsr = 6.7
```

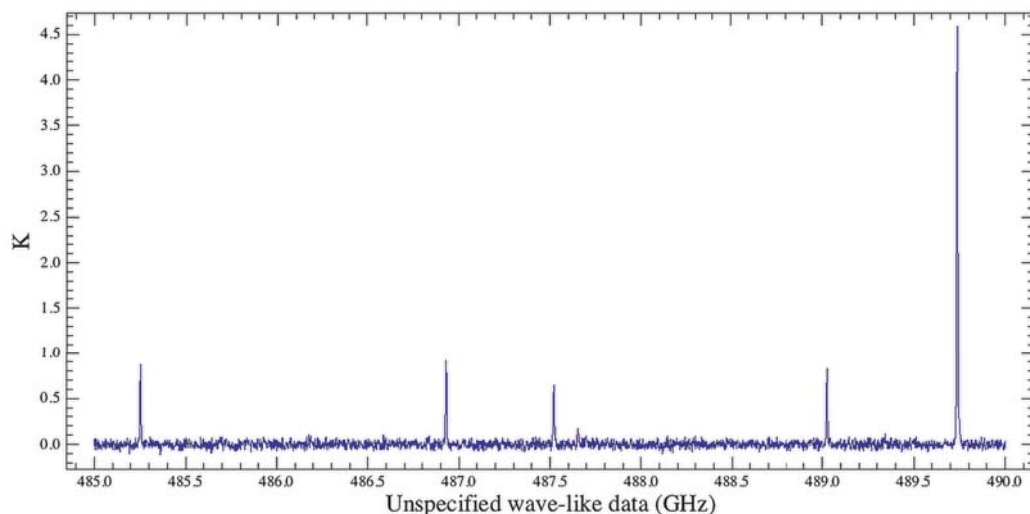


Figure 21.2. The spectrum1d of Orion South in band 1a



The line identification is performed with the task `identifyLines`. You pass the data defined above to the task:

```
# In our example, vlsr is already set to vlsr = 6.7 (see previous script)
lines = identifyLines(data=spectrum, linelist=linelistFile, vlsr=vlsr)
```

You then get the identified lines, unidentified lines (only if there is at least one), and the baselined spectrum:

```
identified = lines["identified"]
unidentified = lines["unidentified"]
baselinedSpectrum = lines["baseline"]
```

Whilst the command:

```
lines = identifyLines(data=spectrum, linelist=linelistFile, vlsr=vlsr)
```

may be enough to perform the line identification, it makes many assumptions and sets default values to the optional parameters. You may want to have more control on what is actually happening. This is what we describe further down in [Section 21.3](#). If you want to learn how to work with the resulting lines, please consult [Section 21.4](#).

Parameter	Comment	default value
box	The input smooth box used by <code>smoothBaseline</code> in the detection step.	200
calibration	The calibration of the instrument used in the computation of the noise flux. Used in the filtering step.	0.1
data	The <i>SpectrumId</i> that contains the lines to identify.	
fwhmAbsorption	The FWHM values (in km/s) for the absorption lines. This value is used for the computation of the noise flux in the criterion for filtering real lines. Used in the filtering step.	1.0
fwhmEmission	The FWHM values (in km/s) for the emission lines. This value is used for the computation of the noise flux in the criterion for filtering real lines. Used in the filtering step.	3.0
lines	The detected lines output file.	
linelist	The path to the linelist file. Default is to leave unset, the task will then use the linelist already present in the build.	
midcycle	The middle of the frequency range in the fringe search (per inverse wavenumber in micron). Used by <code>smoothBaseline</code> in the detection step.	1700000.0

Parameter	Comment	default value
minSNR	The minimum signal-to-noise ratio for which the filtered line is accepted. Used in the filtering step.	5.0
nbOfSplits	The number of shorter spectra of the same size to create from the initial wide spectrum. Splitting the initial spectrum has 2 benefits: 1) the fit of the baseline is better, which slightly improves the detection, 2) the number of channels to be processed by <code>smoothBaseline</code> is reduced, which greatly improves its speed. If you do not want the data to be split, set <code>nbOfSplits = 1</code> .	4
overlap	The overlap (in GHz) between the split spectra. Overlapping the spectra prevent the loss of lines between 2 consecutive spectra.	0.25
smoothBox	The number of points on which the smoothing is done during the separation.	7
toleranceFactorEmi	The tolerance factor for the emission lines. The tolerance factor allows to determine how the detected frequency should be compared to the known frequency. In particular, it defines a frequency range in which the comparison is done. The known line within this frequency is then associated to the detected line.	1.5
toleranceFactorAbs	The tolerance factor for the absorption lines. The tolerance factor allows to determine how the detected frequency should be compared to the known frequency. In particular, it defines a frequency range in which the comparison is done. The known line within this frequency is then associated to the detected line.	1.5
vlsr	The vlsr of the source in km/s.	
width	The width of the region (in GHz) on which the RMS is computed around the line. Used in the filtering step.	0.05

Parameter	Comment	default value
snrType	The signal to noise ratio calculation algorithm (Intensity or Flux).	Intensity

To obtain details about all the parameters, you can also type:

```
print identifyLines.__doc__
```

or read the corresponding entry in the HIFI User Reference Manual.

## 21.3. Advanced Usage

`identifyLines` allows you to determine how the different steps of the identification must be performed through several optional parameters. Before introducing them, it is necessary to understand what `identifyLines` does.

The line identification is a 2-step process:

- the lines are detected:
  - In order to optimise the quality and the speed of the detection, the spectrum is split into N smaller spectra (*nbOfSplits*) with a given 'overlap'.
  - The detection is performed by `smoothBaseline` which needs a *box* value for the smoothing and a *midcycle* value (see [smoothBaseline](#)).
  - The detected regions of lines are then separated using a boxcar smoothing (*smoothBox*) and a slope detection algorithm.
  - All the lines that are detected may not be real lines but noise, so a filter using a minimal signal-to-noise ratio (*minSNR*) is necessary to keep only relevant lines. The signal-to-noise ratio is computed with an estimation of the FWHM of the lines (*fwhm*), the *calibration* value of the instrument, and the *width* of the region around the line used to compute a local RMS.
- these detected lines are then compared with the known lines (*linelist*):
  - Basically, each detected line is compared with each known lines, and if the frequency between the detected line and a known line is inferior to a given *tolerance*, the detected line is identified otherwise it is unidentified.

Using the parameters table found in the "basic" section (see [Section 21.2](#)), we can re-write the basic command line:

```
lines = identifyLines(data=spectrum, linelist=linelistFile, vlsr=vlsr)
```

into an extended form (here, using the default values):

```
lines = identifyLines(data=spectrum, linelist=linelistFile, vlsr=vlsr, \
    box=200, midcycle=1.7E6, nbOfSplits=4, fwhmEmission=3.0, \
    fwhmAbsorption=1.0, toleranceFactorEmi=1.5, toleranceFactorAbs=1.5, \
    calibration=0.1, minSNR=5.0, width=0.05, overlap=0.25, \
    smoothBox=7, snrType="Intensity")

# If you do not set 'linelist', the task will use the default linelist present in
the build
```

followed by:

```
identified = lines["identified"]
unidentified = lines["unidentified"]
baselinedSpectrum = lines["baseline"]
```

to obtain the identified lines, the unidentified lines, and the baselined spectrum.

The identification is then finished. Identifying your lines is one thing, analysing what was identified is another story. Fortunately, the line identification tool comes with a few functions that may be helpful. The next [Section 21.4](#) will teach you about those functions.

## 21.4. A guided tour

This section will teach you about:

- **Getting to know your lines**
- **Saving and plotting**
- **Looking closer**
- **Adding and removing lines**
- **Loading your lines**

From here, you should read the comments and run the code line by line to make sure you understand everything.

### Getting to know your lines

Let's look at the identified lines. The variable *identified* is a Linelist (a data structure containing Line objects):

```
print identified
```

You should see the number and the species of the lines. If you want to print more details for each line, use the *inspect* method:

```
identified.inspect()
```



#### Note

*inspect* does not return anything, it just prints the result.

Sometimes, a table is more handy:

```
table = identified.toTable()
```

If you open *table* in the Dataset viewer you will see the same result as with *inspect* but this time, each line is in a row, and each attribute of a line is in a column. If you want to get, for example, the sky frequency of all the lines, you can use:

```
print table["skyFrequency"].data
```

But you can also use:

```
print identified.get("skyFrequency")
```

Working with the Linelist has other advantages. You can select the lines with a specific attribute. For example, this is how you can select all the A-CH<sub>3</sub>OH lines:

```
print identified.select(species="A-CH3OH")
```

This command returns a Linelist, which means that you can apply all the methods you know so far, including *select*:

```
ach3oh = identified.select(species="A-CH3OH").select(maxAmplitude=[0.7, 0.9])
print ach3oh
```

You have just selected all the A-CH<sub>3</sub>OH lines, whose maximum amplitude is between 0.7 and 0.9 K. You can check it is true with:

```
print ach3oh.get("maxAmplitude")
```

If you want more details about the command *select*, type *print identified.select.\_\_doc\_\_*.

### Saving and plotting

If you are happy with the lines you have (we will assume you are), you can save them in a FITS file so that you can use them later.

```
ach3oh.saveLinesTo("ach3oh_lines.fits")
identified.saveLinesTo("identified_lines.fits")
```

*saveLinesTo* saves all the lines in a TableDataset into a FITS file. For a CSV file, use *'csv'* instead of *'fits'* if you prefer CSV.

So far you just used numbers and tables. You can have a spectrum of the lines by using the *spectrum* method:

```
ach3ohSpectrum = ach3oh.spectrum(data=baselinedSpectrum)
```

or just plot them with *plot*:

```
ach3oh.plot(data=baselinedSpectrum)
```

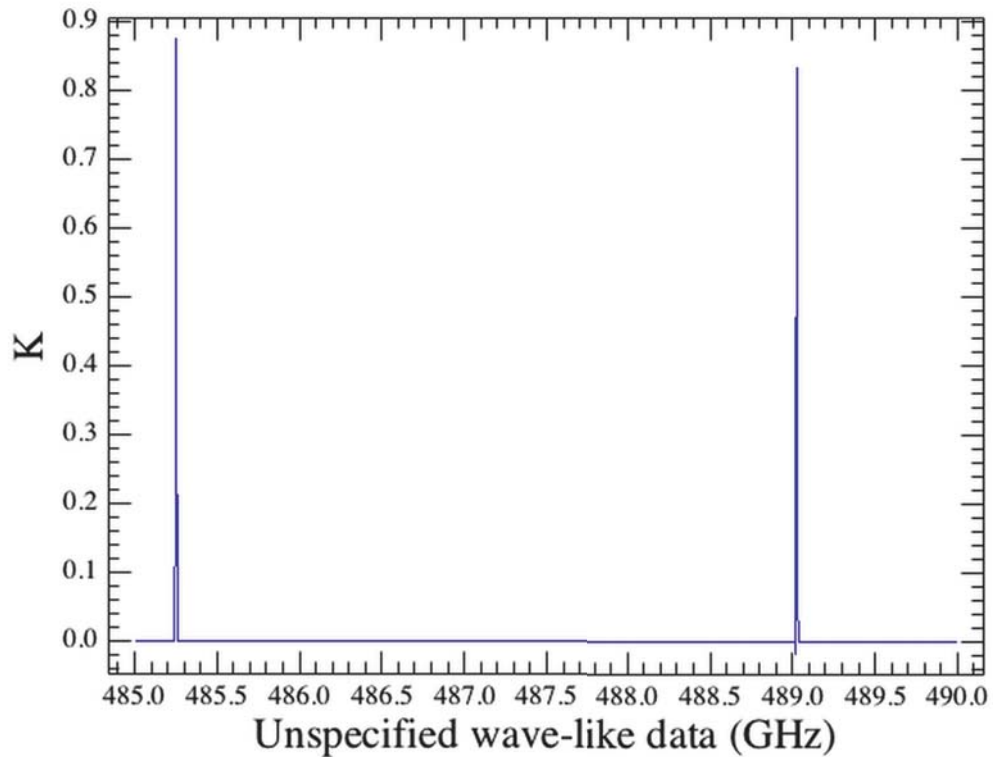


Figure 21.3. Plot example for `identifyLines`



#### Note

You need the *baselined* spectrum (or at least a spectrum) for 'spectrum' and 'plot'. We will explain why later as it involves some details on how the Line was implemented.

The same way you used `saveLinesTo` to save your lines to a file, you can save your spectrum to a file with `saveSpectrumTo`:

```
ach3oh.saveSpectrumTo("ach3oh_spectrum.fits", spectrum=baselinedSpectrum)
```

You probably noticed that the spectrum created is just a continuum at 0 K except where the lines were detected. Let us choose the first line and see what we can learn.

#### Looking closer

It was mentioned earlier that a `Linelist` contains `Line` objects. A `Line` just stores information about the line detected in your spectrum. In order to get a line, specify its index (starting from 0):

```
print ach3oh[0]
```

In fact, you can treat the `Linelist` like any other iterable you are used to. For example, you can iterate on it with a 'for' loop:

```
for line in ach3oh:
    print line
```

As for a `Linelist`, if you want to know in details the attributes of this line, just use `inspect`:

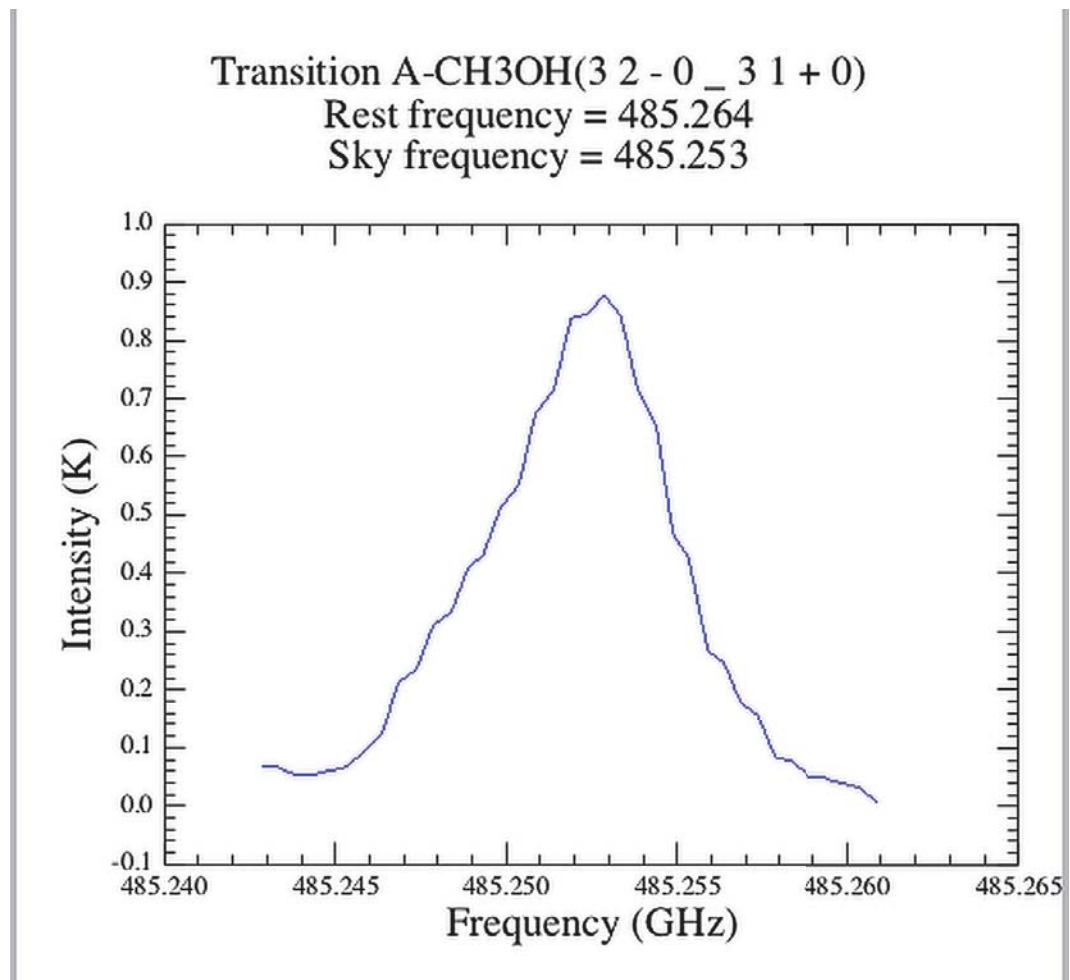
```
print ach3oh[0].inspect()
```

All these attributes are easily accessible. For example, you want to get the signal-to-noise ratio (snr) of this line, you just specify this attribute:

```
print ach3oh[0].snr
```

Linelist and Line share some methods like *plot* and *spectrum*. So you can plot this line with the *plot* method:

```
ach3oh[0].plot(data=baselinedSpectrum)
```



**Figure 21.4.** Plot example for `identifyLines` with specifying an index number

and obtain the corresponding spectrum with the *spectrum* method:

```
ach3ohSpectrum = ach3oh[0].spectrum(data=baselinedSpectrum)
```

One remark about these 2 methods and the philosophy behind the Line object. As you saw earlier, a Line object just has information about the line in the spectrum, not the spectrum itself. The attribute allowing to bind the line and the spectrum is the frequency range. This is why the methods *plot* and

*spectrum* need the spectrum as argument in order to select only the part of the spectrum within that frequency range.

### Adding and removing lines

Any convenient data structure allows you to add or remove elements, so does the *Linelist*. First, let's create an empty *Linelist*:

```
from herschel.hifi.dp.tools.linelist import Linelist
myLinelist = Linelist()
```

You can add a *Line* or a *Linelist* into this *Linelist* with *add*. Here, we add all the A-CH<sub>3</sub>OH lines and the H<sub>2</sub>CS line:

```
ach3ohAll = identified.select(species="A-CH3OH")
h2cs = identified.select(species="H2CS")
myLinelist.add(ach3ohAll)
myLinelist.add(h2cs)
```

If we now look at the *Linelist*, there should be 5 lines in it:

```
print myLinelist
print len(myLinelist)
```

You could have had the same result by removing the CS, v=0-4 line from our initial *Linelist* with the method *remove*:

```
cs_v0_4 = identified.select(species="CS, v=0-4")
identified.remove(cs_v0_4)
```



#### Note

You could also have removed it with its index (i.e. 5):

```
identified.remove(5)
```

You must be careful with *remove* because it removes permanently the lines from the *Linelist*. If you want to make a copy of your *Linelist*, you can do it by creating a new *Linelist* with the lines you want instead of leaving it empty:

```
identifiedCopy = Linelist(identified)
print identifiedCopy
```

To remove all the A-CH<sub>3</sub>OH lines:

```
identified.remove(ach3ohAll)
```

To print the result:

```
print "The original linelist has %d lines." % len(identified)
print "The copied linelist has %d lines." % len(identifiedCopy)
```

You can see that *identified* does not have A-CH<sub>3</sub>OH lines anymore, and *identifiedCopy* is not modified.



### Loading your lines

It is generally a good idea to save your lines so you can reload them if you want to share them, or if something goes wrong. Fortunately, this is what you did in the **Saving and Plotting** section above. You can load all the lines you had at the beginning with `loadLinesFrom`.

First you create an empty `Linelist` that will be populated by the lines that you will load from the file `identified_lines.fits`:

```
originalLines = Linelist()
originalLines.loadLinesFrom("identified_lines.fits")
```

## 21.5. The exportLines Task

The `exportLines` task was designed to export the results from `identifyLines` into a `SpectralLineList`.

In order to use it, you need to set some parameters:

Parameter	Comment	default value
lines	The results from <code>identifyLines</code> .	None
obs	The observation context which contains the <code>SpectrumId</code> used in <code>identifyLines</code> .	None
spectrum	The spectrum used as data in <code>identifyLines</code> .	None
author	The author name.	None
calibration	The calibration of the instrument used in the computation of the noise flux.	0.1

```
# Get the observationContext
obs = getObservation(obsid=1342203743, useHsa=True)

# or from your localStore
# obs = getObservation(obsid=1342203743, poolName="1342203743")

hsls = exportLines(lines=lines, obs=obs, spectrum=spectrum, author="AUTHOR_NAME",
  calibration=0.1)

# After that, you can get the results with the same Linelist elements from lines:
hslsIdentified = hsls["identified"]
hslsDetected = hsls["detected"]
```

## 21.6. Line assignment: the identifyLinesCatalog task

A new task has been introduced, that allows to perform automatic line assignment based on an input catalogue of detected spectral features: [identifyLinesCatalog](#) in *HIFI User's Reference Manual*. This task relies on a [default catalogue of spectral lines](#), that is provided together with the HIPE software. Alternatively, a [smaller line list](#) can also be used. This file can be adopted to the users need, and then passed as input to the task. Since the line species catalogue provides frequencies in the local standard of rest, a velocity needs to be provided. This task can e.g. work on spectral feature catalogues from SPIRE, PACS, or any other facility. It can take as input tables with line positions in frequency

(GHz), wavelengths (microns) or wavenumber (cm<sup>-1</sup>). See the [task reference manual](#) in *HIFI User's Reference Manual* for further details.

# Chapter 22. Making Publication quality plots

Last updated: Feb 18, 2011

Within HIPE you have the possibility to make publication quality plots graphically with Spectrum Explorer or via scripts. The ways to do this are described within several chapters in the HIFI Data Reduction Guide and the Herschel Data Analysis Guide. This chapter pulls this information together in a way that is relevant for HIFI data.

Probably the most efficient and customisable way to make plots is to use the PlotXY package, which is described in full detail in the [Plotting](#) chapter of the Herschel Data Analysis Guide. Below is a script which shows how to make this figure.

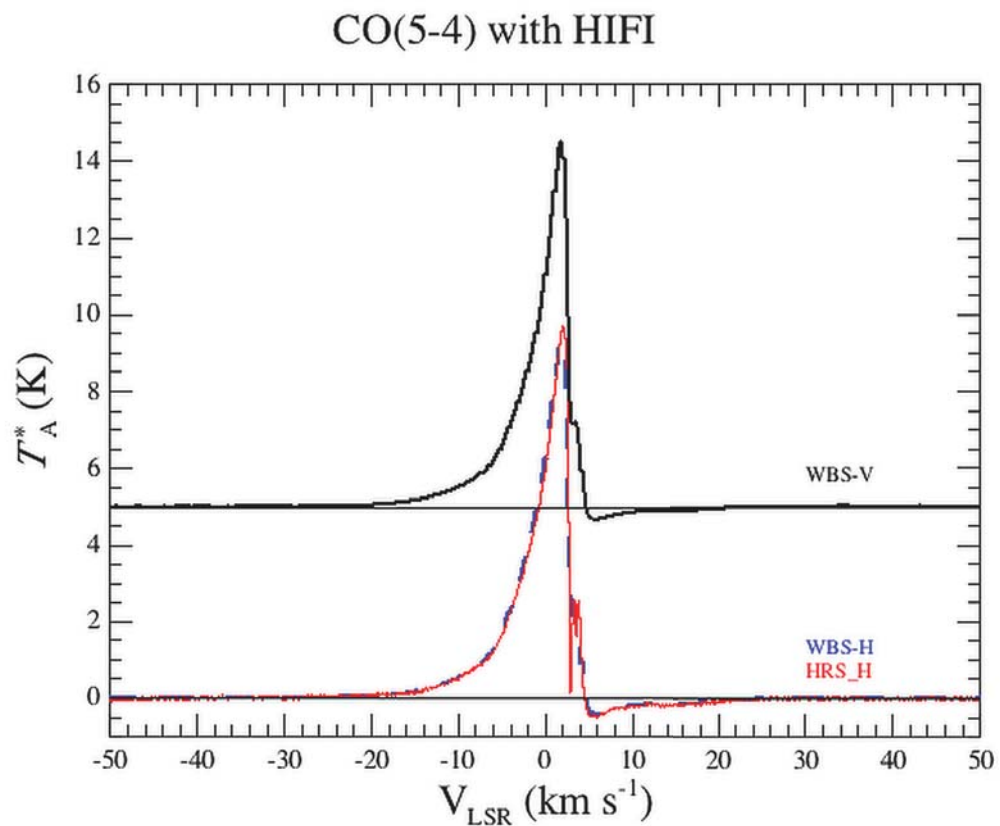


Figure 22.1. Plot example produced by the provided script

```
# Script to create publication quality plots

# Get data from HSA and extract Level 2 WBS-H, WBS-V, and HRS_H USB spectra
obs = getObservation("1342190183", useHsa=True)
L2_wbs_h_u = obs.refs["level2"].product.refs["WBS-H-USB"].product.refs \
["box_001"].product["0001"]
L2_wbs_v_u = obs.refs["level2"].product.refs["WBS-V-USB"].product.refs \
["box_001"].product["0001"]
L2_hrs_h_u = obs.refs["level2"].product.refs["HRS-H-USB"].product.refs \
["box_001"].product["0001"]

# First make a copy of the original datasets so that they are not overwritten
wbs_h_u = L2_wbs_h_u.copy()
wbs_v_u = L2_wbs_v_u.copy()
```

```

hrs_h_u = L2_hrs_h_u.copy()

# Convert to velocity, the strong line is CO(5-4) (576.268 GHz)
convertWavescale(ds=wbs_h_u,to="km/s",reference=576.268,referenceUnit="GHz")
convertWavescale(ds=wbs_v_u,to="km/s",reference=576.268,referenceUnit="GHz")
convertWavescale(ds=hrs_h_u,to="km/s",reference=576.268,referenceUnit="GHz")

# Add 5 to the WBS-V spectrum to create an offset
wbs_v_5 = add(ds=wbs_v_u,param=5.0)

# PlotXY works with Numeric1d (e.g. Double1d) data so you need to extract
# the numbers from the SpectrumDataset and you need to extract the x (wave)
# and y (flux) axes separately.
# The CO line appears in subband (segment) 2 of the WBS, the HRS was set up
# in the observation to have only one subband
wbs_h_wave = wbs_h_u.getPointSpectrum(0).getSegment(2).getWave()
wbs_h_flux = wbs_h_u.getPointSpectrum(0).getSegment(2).getFlux()
hrs_h_wave = hrs_h_u.getPointSpectrum(0).getSegment(1).getWave()
hrs_h_flux = hrs_h_u.getPointSpectrum(0).getSegment(1).getFlux()
wbs_v_wave = wbs_v_5.getPointSpectrum(0).getSegment(2).getWave()
wbs_v_flux = wbs_v_5.getPointSpectrum(0).getSegment(2).getFlux()

# Alternatively, stitch subbands so that can plot all the subbands, this gives
# you one segment to extract, e.g.:
# wbs_v_st = stitch(ds=wbs_v_5,edgeTolerance=0.01,stepsize=0.0,unit="km/s")
# wbs_v_wave = wbs_v_u.getPointSpectrum(0).getSegment(1).getWave()
# wbs_v_flux = wbs_v_u.getPointSpectrum(0).getSegment(1).getFlux()

# Create plot. This is done in batch mode so that the plot is not updated
# everytime you add a layer (p.batch=1 starts batch mode, p.batch=0 stops it
# and creates the plot)
p=PlotXY()
ll=[]
# Make a blue dashed line (line=3) line, with thickness (stroke) 1.5
l = LayerXY(wbs_h_wave, wbs_h_flux, color=java.awt.Color.BLUE, line=3, \
stroke=1.5, chartType=Style.HISTOGRAM)
ll.append(l)
# Make a red solid (line=1), thickness 0.75
l = LayerXY(hrs_h_wave, hrs_h_flux, color=java.awt.Color.RED,line=1, \
stroke=0.75, chartType=Style.HISTOGRAM)
ll.append(l)
# Make a black solid line, thickness 1
l = LayerXY(wbs_v_wave, wbs_v_flux, color=java.awt.Color.BLACK,line=1, \
stroke=1, chartType=Style.HISTOGRAM)
ll.append(l)
p.layers=ll

# Now set the axes limits and titles
p.xaxis.range = [-50.0, 50.0]
p.yaxis.range = [-1, 16]
# Latex options are available but an extra backslash is needed
from java.awt import Font
p.xtitle = "V$_{\text{LSR}}$ (km s$^{-1}$)"
p.ytitle = "T$_{\text{A}}^*$ (K)"

# You might want to draw a zero level line for each plot.
# Create a "zero spectrum" by subtracting one from itself
zero = wbs_h_u - wbs_h_u
# You can add five to it to create a line at y=5
five = zero + 5.0
# Now we have to extract the numbers again, make sure to
# use the same segment as used to plot data from
z_wave = zero.getPointSpectrum(0).getSegment(2).getWave()
z_flux = zero.getPointSpectrum(0).getSegment(2).getFlux()
f_wave = five.getPointSpectrum(0).getSegment(2).getWave()
f_flux = five.getPointSpectrum(0).getSegment(2).getFlux()

# Now add these to the plot we already made
l = LayerXY(z_wave, z_flux, color=java.awt.Color.BLACK,line=1, \

```

```

stroke=0.5, chartType=Style.HISTOGRAM)
ll.append(l)
p.layers=ll
l = LayerXY(f_wave, f_flux, color=java.awt.Color.BLACK,line=1, \
stroke=0.5, chartType=Style.HISTOGRAM)
ll.append(l)
p.layers=ll

# Annotate plot
# Clear any old annotations first (helpful if you need to move annotations)
p.clearAnnotations()
# Annotations are placed using plot location (x,y)
p.addAnnotation(Annotation(30,5.5,"WBS-V",color=java.awt.Color.BLACK))
p.addAnnotation(Annotation(30,1.0,"WBS-H",color=java.awt.Color.BLUE))
p.addAnnotation(Annotation(30,0.4,"HRS_H",color=java.awt.Color.RED))

# Add a title
p.setTitleText("CO(5-4) with HIFI")

# Now save
p.saveAsEPS("Fig1.eps") # Encapsulated PS

# You can also saveAsPNG, JPG or PDF, and set a path
p.saveAsPNG("Fig1.png") # Encapsulated PNG
p.saveAsPDF("Fig1.pdf") # Encapsulated PDF

```

There is also another scripting package for plotting called SpectrumPlot (or splot), which is described in [Section 6.4](#). Splot is simpler to use but is still limited in some capabilities.

You can use SpectrumExplorer to:

- Plot multiple spectra by dragging spectra from the Variables view into the plot.
- Modify the line (colour, style and thickness) and symbol (shape, size and colour) in the properties dialogue. You can also provide a name for the line that will appear in the legend below the plot.

To do this, hover your mouse over the spectrum until you see a dot the same colour as the line you want to modify next to `spectrum [name...]` beneath the plot, and select properties from the right click menu.

To change the line colour is not so obvious: click on the coloured square and then on the grey box that appears to the right. Then select your preferred colour from the grid.

Property changes may require you to hit return before taking effect.

If you have multiple spectra plotted you can hover over a different spectrum and 'shift-click' to switch the dialogue to the properties for that spectrum.

- Similarly, you can modify the properties of the axes (labels and tick marks), and of the plot itself by right clicking over those regions.
- You can add an auxiliary axis and change the units displayed. To do this, right click on the axis (top or bottom) and select 'Axis->add aux axis'. A new axis with the same units as the main axis will be displayed.

To change the units right click on the axis again and select 'Axis->set unit'. To convert between frequency and velocity you will need to supply a reference value. Note that this is a display function only, the data itself is not changed.

You may wish to right click on the axis title and change it to a more appropriate name via the properties menu.

- You can print or save the plot by right clicking on the plot and selecting the file menu. You can save as eps, png, pdf or jpeg.

The displayed plot is printed/saved; a zoom on the plot is retained.

An example of what can be done using SpectrumExplorer is shown below. The same data as used in the script above is plotted.

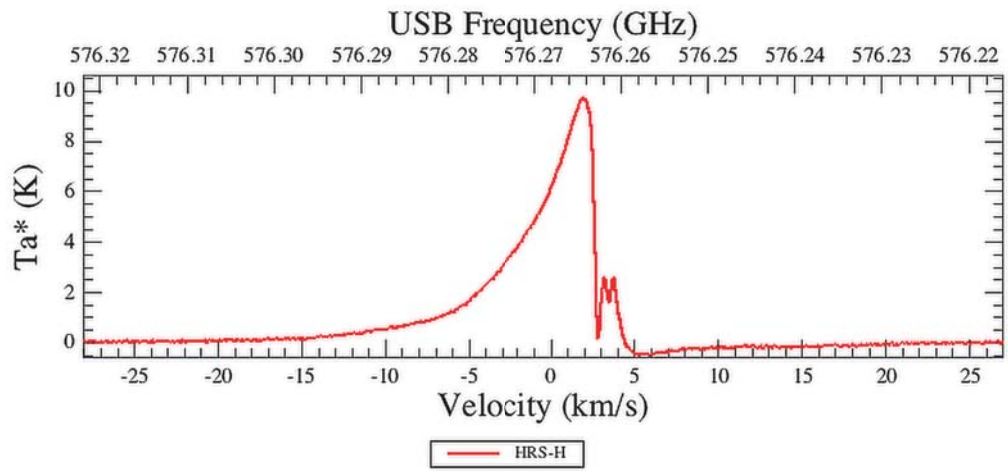


Figure 22.2. Same result as above but if plotting with Spectrum Explorer

# Chapter 23. Exporting HIFI data to CLASS

Last updated: 22 November, 2016

## 23.1. Processing version from HIPE 12 onwards: direct FITS reading

If you are using a Gildas version later than oct15, it is possible to read HIFI FITS files directly from CLASS. This will, however, only be true for products generated from HIPE 12 onwards. For older products, you should use the `hiClass` task (see next section). The direct conversion will apply both to level 2 and level 2.5 products from the Archive, as well as any equivalent products generated from HIPE using e.g. `FitsArchive()`. Also, don't forget that the FITS in Level 2 and 2.5 observations downloaded from the HSA are gzipped. To test if the FITS are properly un-zipped before reading to CLASS, the user could try a utility like 'fv' (FITS Viewer) particularly good for spectra, or 'ds9' for spectral cubes

Because the CLASS header expects a pre-defined set of meta-data, all meta-data need to be filled when the products is imported. If some of those are missing, they will be set to default values in CLASS. This can happen for some meta-data that were not introduced in the 12 and 13 version of the HCSS products. An example is shown below where such missing meta-data are reported in a set of warning messages.

```
## Example of direct reading from a HIFI FITS file
LAS> file out spectra.hifi mul
LAS> fits read HIFI_SScan_Spec1d.fits
I-FITS, Found the Basic HDU.

I-FITS, Found a Binary Table in extension #1
I-FITS, Importing data from the 'frequency' and 'flux' columns

I-FITS, Found a Binary Table in extension #2
W-FITS, No relevant data in this HIFI extension, skipping

W-FITS, Skipping remaining extensions

W-FITS, --- Warning summary (all extensions) ---
W-FITS, Column flag not found, channel flags defaults to 0
W-FITS, Metacard rowflag not found, row flags defaults to 0
W-FITS, Metacard bdtype not found, R%HEAD%GEN%SCAN defaults to 0
W-FITS, Metacard bnumber not found, R%HEAD%GEN%SUBSCAN defaults to 0
W-FITS, Metacard tsys_median not found, R%HEAD%GEN%TSYS defaults to .0
W-FITS, Metacard LoFrequency or LoFrequency_measured not found, LO frequency
defaults to .0
W-FITS, Metacard Gain not found, R%HEAD%CAL%GAINI defaults to .0
W-FITS, Card ETAMB not found, R%HEAD%HER%ETAMB defaults to .0
W-FITS, Card ETAL not found, R%HEAD%HER%ETAL defaults to .0
W-FITS, Card ETAA not found, R%HEAD%HER%ETAA defaults to .0
W-FITS, Card HPBW not found, R%HEAD%HER%HPBW defaults to .0
W-FITS, Card REDSHFT and/or metacard 'vlsr' not found, R%HEAD%HER%VINFO and R%HEAD
%HER%ZINFO default to 0
W-FITS, Card LODOPPAV not found, R%HEAD%HER%LODOPAVE defaults to LO frequency
W-FITS, Metacard Gain_0 not found, R%HEAD%HER%GIM0 defaults to .0
W-FITS, Metacard Gain_1 not found, R%HEAD%HER%GIM1 defaults to .0
W-FITS, Metacard Gain_2 not found, R%HEAD%HER%GIM2 defaults to .0
W-FITS, Metacard Gain_3 not found, R%HEAD%HER%GIM3 defaults to .0
W-FITS, Metacard MJC_Hor not found, R%HEAD%HER%MIXERCURH defaults to .0
W-FITS, Metacard MJC_Ver not found, R%HEAD%HER%MIXERCURV defaults to .0
W-FITS, ==> One or more meta-data missing, default values used in CLASS
```

The users should keep in mind that those warnings do not prevent the data from being properly read. All details about this HIFI FITS reader in CLASS can be found in the following [documentation](#). You should also be aware that the flags present in the HIFI FITS data can now be imported into the CLASS data and made use of. This is described in this [document](#).



#### Warning

Please be aware that the HSC cannot to guarantee that CLASS will read those FITS files in future versions of CLASS. In case of issues, please get in touch with the Gildas helpdesk: [gildas@iram.fr](mailto:gildas@iram.fr).

## 23.2. Processing version earlier than HIPE 12: the hiClass task

### 23.2.1. Introduction to hiClass

The `hiClass` task can be used for ObservationContexts, Level 0, 0.5, 1, 2 products, HTP of Level 0, 0.5, 1, 2 data, and SpectrumDatasets. Note that if you pass on the ObservationContexts to the task, only the Level 2 data is exported.

Note that it is possible to export the Level 2.5 products but only those which are in the form of HTPs - *SpectrumId* such as the level 2.5 products for point or spectral scans cannot be exported by `hiClass`. For spectral scans, a dedicated task `hifiDeconToClass` can be used.

The following information is exported to CLASS:

- The fluxes
- ObsId, BbType, BbId
- The name of the observed source, which can be non-astronomical for spectra of level lower than 2 (integrations on the Hot Black Body for example).
- The Rest Frequency, Image Frequency, Channel References, Frequency Step. `hiClass` always chooses the centre of the spectrum as the reference.
- Dates of observation, and name of the instrument (HIFI plus spectrometer and polarisation)
- Pointing information
- Tsys
- Forward and, if present, beam efficiency

The `hiClass` task is a wrapper around the HiClass object defined in `herschel/hifi/dp/tools/hi-class_tools.py`. Only the usage of the `hiClass` task is described here. If you want to work directly with the HiClass object, you can read further documentation about how the HiClass object works, including examples, by typing in the console:

```
print herchel.hifi.dp.tools.hiclass_tools.__doc__
```

#### Metadata propagation to Class FITS files

`hiClass` propagates new metadata in the Class FITS files per default, and exports the data to a Product that contains *TableDatasets* (each *TableDataset* represents a *HifiSpectrumDataset*).

These default metadata can be grouped in 2 categories:



- the metadata coming from a *HifiSpectrumDataset*:

Metadata name	Proposed FITS name	Description
usbGain_0 or lsbGain_0	GAINCOE0	Sideband gain correction coefficient 0
usbGain_1 or lsbGain_1	GAINCOE1	Sideband gain correction coefficient 1
usbGain_2 or lsbGain_2	GAINCOE2	Sideband gain correction coefficient 2
usbGain_3 or lsbGain_2	GAINCOE3	Sideband gain correction coefficient 3
posAngle	POSANGLE	[deg] Observation position angle
naifId	NAIFID	source NAIF ID (0 if fixed target)
raoff	RAOFF	[deg] Commanded RA (J2000) of reference position
decoff	DECOFF	[deg] Commanded DEC (J2000) of reference position

```
## These metadata are propagated in the rows of the Tabledataset
## Example

print h.Prod["0"].meta["POSANGLE"]
{description="Observation position angle", double=98.46283357392767, unit=deg [1 deg
= 0.017453292519943295 rad]}
```

- the other metadata:

Metadata name	Proposed FITS name	Description
calVersion	CALVERS	caltree version
creator	HIPEVERS	HIPE version
cusMode	OBSMODE	Observing mode
pmRA	PMRA	[arcsec/yr] Target proper motion in RA
pmDEC	PMDEC	[arcsec/yr] Target proper motion in DEC
apertureEfficiency	ETAA	Aperture efficiency
hpbw	HPBW	[arcsec] Half-Power Beam Width
forwardEfficiency	ETALCAL	Take that from the HCSS header
mainBeamEfficiency	ETAMBCAL	Take that from the HCSS header

```
## These metadata are propagated in the Product
## Example

print h.Prod.meta["PMRA"]
{description="Target proper motion in RA", double=0.0, unit=arcsec/a [1 arcsec/a =
1.5362818500441602E-13 rad/s]}
```

## 23.2.2. hiClass examples

The `hiClass` task appears under the *Applicable Tasks* menu where you can open the GUI from there. The selected product (ObservationContext, HTP, SpectrumDataset...) will then be loaded into the *data* bullet. You can also open the GUI from the *HIFI* list of tasks under *By Category*. Similarly, an input product should be dragged to the *data* bullet. It is also possible to invoke the `hiClass` task by simply using the right-click on a variable and select *Sent to* and then *CLASS FITS file*. This method is a shortcut to the *Applicable Tasks* menu to open the `hiClass` task GUI.

The GUI is shown in [Figure 23.1](#). An output filename is also required (*fileName*). The parameter *engineeringMode*, if selected, will force the access to the calibration tree root.

### Inputs accepted by HiClass

- HifiSpectrumDataset
- HrsSpectrumDataset
- WbsSpectrumDataset
- SingleHifiSpectrum
- HifiVectorDataset
- Product
- ProductRef
- HashSet
- HfiTimelineProduct
- ObservationContext
- CalFluxHotCold

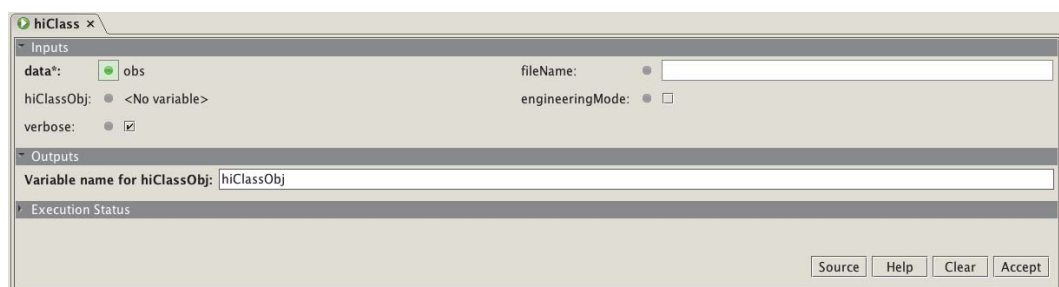


Figure 23.1. HiClass task GUI

### Command line examples

The `hiClass` task can also be used in the command line. Some examples are shown in this section. They cover a range from basic usage (like in the GUI), to more advanced scripting.

1. Export the Level 2 spectra (by default) to a FITS file, by supplying an ObservationContext (obs):

```
hiClassObj = hiClass(data = obs, fileName = 'obs.fits')
```

Example 6 will teach you how to export other levels.

Note that if you do not specify a path in the *fileName*, the FITS file will be saved in the directory in which you launched HIPE. See example 4 to learn how to specify a path to *fileName*.

2. Export one HIFI timeline product, including calibration observations, to a FITS file:

```
# First extract an HTP from the ObservationContext.
# Here the Level 1 HTP for the HRS-V is extracted from an ObservationContext
  named obs.

htp = obs.refs["level1"].product.refs["HRS-V"].product

#
# Now create the FITS file:

hiClassObj = hiClass(data = htp, fileName = 'myhtp.fits')
```

3. Export one dataset (one spectrum) to a FITS file:

```
# First, extract a dataset from the ObservationContext.
# Here the first spectrum in the Level 2 product for the upper side band of the
  WBS-H
# is extracted from an ObservationContext named obs.

spectrum = obs.refs["level2"].product.refs["WBS-H-
USB"].product.refs["box_001"].product["0001"]

#
# Now create the FITS file

hiClassObj = hiClass(data = spectrum, fileName = 'myspectra.fits')
```

4. The *.fits* file is written in the active directory. You can specify a specific path to write to a different location:

```
hiClassObj = hiClass(data = obs, fileName = '/mypath/mydir/obs.fits')
```



#### Note

In a more general way, to specify a path in a platform independent way, use `os.path.join()`.

```
import os # See http://www.jython.org/docs/library/os.path.html
file_name = 'myfits.fits'
file_path = os.path.join('~', 'hifi', 'ngc1234', file_name)
hiClassObj = hiClass(data = myhtp, fileName = file_path)
```

5. The `hiClass` task is basically a wrapper around the 'HiClass' class which provides more flexibility but may be less user-friendly without the graphical user interface.

With `HiClass`, example 1 becomes:

```
from herschel.hifi.dp.tools.hiclass_tools import HiClass
h = HiClass()
h.add(obs)
h.saveToFits("my_obs.fits")
```

The advantage of this form is that you can add as many data as you want to the final exported product. This following example shows how to export only some *WbsSpectrumDatasets* called 'ds1', 'ds2', and 'ds3':

```

h = HiClass()
h.add(ds1)
h.add(ds2)
h.add(ds3)
h.saveToFits("my_3_datasets.fits")

```

## 6. Export data at different levels:

By default, with an *ObservationContext*, *hiClass* exports only the Level 2 data.

If you also want to export the other levels, you can do:

```

h = HiClass()
for level_number in ('0', '0_5', '1', '2'):
    level_name = 'level%s' % level_number
    try:
        p = obs.refs[level_name].product
    except AttributeError:
        print "Could not retrieve %s." % level_name
    else:
        h.add(p)
h.saveToFits('my_obs.fits')

```

## 7. Export system temperature:

You can also export the system temperature, *Tsys*. In this particular case, only the *Tsys* for a spectrometer and a polarisation is exported. If you want to do this, you need to provide the *Tsys* this way:

```

h = HiClass()
for backend in ('WBS-H', 'WBS-V', 'HRS-H', 'HRS-V'):

    tsys = obs.calibration.getProduct('pipeline-out').\
        getProduct('Tsys').getProduct(backend)
    if tsys is None:
        print "Couldn't retrieve %s." % backend
    else:
        h.add(tsys)
h.saveToFits('tsys.fits')

```

## 8. Export several FITS files:

- Either you instantiate one *HiClass* per FITS that you want to create:

```

h = HiClass()
h.add(dataset_for_fits_1)
h.saveToFits("fits1.fits")

h = HiClass()
h.add(dataset_for_fits_2)
h.saveToFits("fits2.fits")

```

- Or you can use the method *reset*:

```

h = HiClass()
h.add(dataset_for_fits_1)
h.saveToFits("fits1.fits")

h.reset()

h.add(dataset_for_fits_2)
h.saveToFits("fits2.fits")

```

There is really no good reason to use one rather than the other. It really depends on what you want to do.

### 23.2.3. How to read HIFI data in CLASS

In case you need to read HIFI data converted with `hiClass`, please make sure to use a version of CLASS released from April 2010 onwards.

- Old versions use Fortran 77 and will not be able to dynamically allocate the memory needed to read big spectra like WBS ones (8000 channels),
- Old versions do not know about the subscan number, and will not be able to make any difference between the different subbands of a spectrum,
- Old versions have troubles with reading double precision values from FITS files,
- Some versions (first half of 2009) have a broken code which totally prevents reading any FITS file with a long header,
- Note that data on the  $T_A^*$  scale have to be multiplied by  $F_{\text{eff}}/B_{\text{eff}}$  to scale to main beam temperatures,  $T_{\text{mb}}$ .

```
file out MyHIFISpectra.hifi mul
fits read MyHIFISpectra.fits
```

Now you have a CLASS file named `MyHIFISpectra.hifi` (you can use whatever you want as an extension) you can access like you always do in CLASS:

```
file in MyHIFISpectra.hifi
find
get first
set unit f i
device image white
plot
```

The `hiClass` task exports information that is stored in the CLASS fits file metadatum (or header), which can be viewed with the `list` command and are:

- *Source*: science datasets are labelled by the name of the astronomical source. Calibration datasets are named appropriately; 'comb', 'tune', 'hot', etc.
- *Line*: LO frequency in GHz (corrected for the spacecraft Doppler-shift), suffixed with 'USB' or 'LSB' according to the exported dataset.
- *Telescope*: 'HIF' for HIFI, subband number (01-16), spectrometer and polarisation ('HH', 'HV', 'WH', 'WV'), and LO band. Note that each subband is exported; you may wish to stitch spectra before using the `hiClass` task.
- *The offset positions (in arcsec)*: computed with respect to the intended source coordinates. Note that due to the imperfect co-alignment between the respective H- and V-polarisation aperture, those offsets may slightly differ between data from the two polarisations.
- *Scan, subscan*: reflect the respective BBtype and BBnumber of the observations, see the Preliminaries section to the Generic pipeline in the [HIFI Pipeline Specification Document](#) for more information.
- *The forward and, if present, beam efficiencies*: how `hiClass` fills these in depends on if you have applied the `doMainBeamTemperature` task to your data.
- **Data processed prior to HIPE 13:**
  - If you have not applied the `doMainBeamTemperature` task, the metadata of your Level 2 products have an entry `forwardEff`, which is set to 0.96, as it is the value that was applied

to the Level 1 data to bring them from the  $T_A$  scale to  $T_A^*$ . When `hiClass` sees this, it fills the Class header with `Feff=Beff=0.96`.

- If you have applied the `doMainBeamTemperature` task then the metadata of your transformed Level 2 products have now an entry `beamEff`, set to the frequency-dependent value that applies to your data, and you are in a  $T_{mb}$  scale. When `hiClass` sees this, it should fill the Class header with `Feff=0.96` and `Beff` with the applicable value.
- There should always be at least one of `forwardEff` or `beamEff` in the metadata of data that has been through the pipeline in a standard fashion. However, if `hiClass` is, for some reason, not able to find either of them it will export your data with `Feff=Beff=1`, essentially signifying that it does not know in which scale the data are. **Note again** that if you have simply multiplied "manually" your  $T_A^*$  data by `Feff/Beff` in HIPE, the metadata will still contain only the `forwardEff` and `hiClass` will thus fill the Class header with `Feff=Beff=0.96`.
- **Data processed with HIPE 13, and onwards:** the temperatures  $T_{A^*}$  and  $T_{mb}$  are determined from the `temperatureScale` metadata
  - If `temperatureScale` is  $T_{A^*}$ :
    - `beamEff = forwardEff = value of the metadata forwardEff`.
  - If `temperatureScale` is  $T_{mb}$ :
    - `beamEff = value of the metadata beamEff`
    - `forwardEff = value of the metadata forwardEff`
  - If `temperatureScale` is not available for some reason:
    - `beamEff = forwardEff = 1`
- In order to handle the old and the new schemes, `HiClass` will distinguish between these two schemes by testing the presence of the metadata `beamEff` and `forwardEff`.
- If both `beamEff` and `forwardEff` are present in the metadata, the new algorithm is applied, otherwise it is the old algorithm that is applied.
- To understand how CLASS reacts to these different items in the header, you are directed to the [CLASS documentation](#).
- The velocity information in the CLASS fits header is assuming `vlsr = 0.0`. `HiClass` sets the reference channel for the velocity to be the middle of the spectrum. You can use **modify frequency** (units MHz) to set as reference channel that of the frequency in the Local Standard of Rest of the line of interest.

The source velocity, as provided in the HSpot AOR, is also found as a FITS header keyword. At this time, the CLASS software does not do anything with this information. This information has been used to properly set the LO for the observation, but has not been used to apply any correction the frequency scale provided by the HIFI pipeline. Therefore, the velocity information present in the CLASS fits header is assuming `Vlsr=0`.

Please refer to CLASS documentation for further information about how to access spectra and headers in CLASS.

## 23.2.4. Exporting the results of deconvolution to Class

To export the results of deconvolution to Class, use the `hifiDeconToClass` task.

- Create a variable of the product labelled `level2.5/myDecon/myDecon_WBS-H (and/or -V)` or select the `decon_result` variable if you reprocessed your data

- When selecting the variable, you will find the task `hifiDeconToClass` in the *Applicable Tasks* menu where you can open the GUI
- Specify the full path name of the fits file you want to create, remembering to add the `.fits` extension
- In the command line, this is done as follows:

```
hifiDeconToClassObj = hifiDeconToClass(decon_result=decon_result, \  
fileName='/Users/Me/decon_result.fits')
```

**Warning**

Please be aware that not all header fields will be filled consistently when exporting spectral scan deconvolved data. In particular, there is no particular LO frequency to report in the line name, or any particular Tsys to populate the corresponding keywords. The same will be true for e.g. the calibration keywords (efficiencies, sideband gain, etc).

**Warning**

In case you are running a recent version of CLASS that allows direct import of the HIFI FITS file, the use of the `hifiDeconToClass` is not necessary. Still, if you wish to read FITS files created with this task, you should use a Gildas version older than nov16.

# Chapter 24. Sending HIFI spectra to VO tools

Last updated: 28 July, 2015

Sending data to VO (Virtual Observatory) tools is described in the [Working with the VO](#) section in the Import/Export Chapter of the Herschel Data Analysis Guide. HIFI spectra can be sent to VO tools, such as VOSpec, by right-clicking on the name in the Variables pane, and selecting a VO tool from the *Send To* list as usual. But you are required to run some tasks on the spectra first, some of which depend on the observing mode used.

Below are the steps needed to take a single Level 2 spectrum and convert it to the format required for export to VO tools. Since tools such as VOSpec do not know how to deal with flux units in Kelvin, this must be done before the export. Also, HIFI data products intrinsically deal with data as two dimensional which is not the standard convention for most ground-based spectrometers. Thus, there will also be a step to convert to a `SimpleSpectrum` product, which is a one dimensional spectrum for Herschel.

## Point spectra.

The Level 2 data can be converted for export to VO tools in the following way:

```
# First get the observation, here we use obsid 1342190183
obs = getObservation(1342190183,useHsa=True)
#
# Now select the Spectrometer (WBS or HRS), the Polarisation (H or V) and the
# Sideband
# (Upper Sidedand (USB) or Lower Sideband (LSB)). Point spectra at Level 2 contain
# only one
# dataset which will be a spectrum2d.
ds_whusb = obs.refs["level2"].product.refs["WBS-H-
USB"].product.refs["box_001"].product["0001"]
#
# The flux units for this spectrum can be converted to Jansky using the
# "convertK2Jy" task.
# Here a point source is assumed.
ds_whusb_Jy = convertK2Jy(ds=ds_whusb, size=0, cal=obs.calibration)
#
# Finally, create a SimpleSpectrum Product which can be sent to VO tools.
onedim_Jy = convertSingleHifiSpectrum(ds_whusb_Jy)
```

## Mapping data.

HIFI produces two "map" products. The first is the fully calibrated spectra in time (`HifiTimelineProducts`), these are found at Level 2. Since, the pointing of the satellite is changing over time, this corresponds to the pieces of the map. To make a spectral cube, these pieces need to be spatially convolved onto a grid. This gridded data is found in cubes in the `cubesContext` at Level 2.5 for data processed with HIPE version 9.0 onwards, or in the `cubes` product found at Level 2 for data processed with earlier HIPE versions. Both the un-gridded and the gridded data can be sent to VO tools. However, tools such as VOSpec do not deal with spectral cubes so the data must be sent as individual spectra.

For Level 2 HTPs:

```
# First get the observation, here we use obsid 1342248770
mapobs=getObservation(1342248770,useHsa=True)
#
# Pull out the HTP from the observation context
mapobs_level2_WBS_V_USB = mapobs.refs["level2"].product.refs["WBS-V-USB"].product
#
```



```
# Extract a single spectrum from the HTP. Here the 4th spectrum of the 4th scan leg
in the map
# (that is, the 4th spectrum in the 4th dataset in the HTP)
ds_4_4 = extract(ds=mapobs_level2_WBS_V_USB.refs["box_001"].product["0004"],
  selection=[4])
#
# Convert the units to Jy. The assumption below is that of a point source which is
technically incorrect.
cal=mapobs.calibration
spectrum_Jy=convertK2Jy(ds=ds_4_4, size=0.0, cal=cal, overwrite=False)
#
# Convert this to a simple spectrum
spectrum4_4 = convertSingleHifiSpectrum(spectra=spectrum_Jy)
#
# spectrum4_4 can be sent to VO tools
```

Now for HIFI cubes:

```
# We use the same obsid as above
mapobs=getObservation(1342248770,useHsa=True)
# For data processed with HIPE versions 9.0 onwards, obtain a SimpleCube from the
Level 2.5 product,
# here we take the cube for the first sideband of the WBS-H-USB
cube_WBS_H_USB_1 = mapobs.refs["level2_5"].product.refs["cubesContext"].product.\
refs["cubesContext_WBS-H-USB"].product.refs["cube_WBS_H_USB_1"].product
#
# To obtain the same SimpleCube from the Level 2 product for data processed with
older HIPE versions, uncomment the line below
# cube_WBS_H_USB_1 =
mapobs.refs["level2"].product.refs["cubes"].product.refs["cube_WBS-
H_USB_1"].product
#
# Extract a single spectrum from the cube using the following command. This command
gets the spectrum
# in the cube location of 4,6. No unit conversion has occurred yet, so flux is still
in K.
sp_4_6_K=extractRegionSpectrum(cube=cube_WBS_H_USB_1, \
regionType=herschel.ia.toolbox.cube.ExtractRegionSpectrumTask.Region.SINGLE_PIXEL, \
centerRow=4.0, centerCol=6.0)
#
# Convert this to a simple spectrum Product
ss_4_6_K=SimpleSpectrum(sp_4_6_K)
#
# Now this can be converted to Jansky before sending to VO tools
ss_4_6_Jy = convertK2Jy(spectrum=ss_4_6_K, size=0.0, cal=mapobs.calibration,
  overwrite=False)
```

### Spectral Scan data.

```
# First load an observation
surveyobsid=1342205334
surveyobs=getObservation(surveyobsid,useHsa=True)
#
# For data processed with HIPE version 9.0 and more recent you can extract the
deconvolved Level 2.5 product,
# here the WBS-H. This will be a Spectrum1d and the flux units will be in
Kelvin. Please note that for data processed
# up to HIPE 13, the SSB result is recorded as 'ssb' and for HIPE 14 onwards, the
SSB result is recorded as 'dataset'.

# up to HIPE 13
decon_result_ssb =
  surveyobs.refs["level2_5"].product.refs["myDecon"].product.refs["myDecon_WBS-
H"].product["ssb"]

# HIPE 14 onwards
decon_result_ssb =
  surveyobs.refs["level2_5"].product.refs["myDecon"].product.refs["myDecon_WBS-
H"].product["dataset"]
```

```
#
# For data processed with an older HIPE versions you must first run doDeconvolution
# before selecting the
# single sideband result (ssb). Note, the step below is a "blind" deconvolution,
# which by default is for the WBS-H.
# Consult the Sideband Deconvolution chapter (Chapter 14) for a description on how
# to get the best results
# from the Deconvolution step.
# Uncomment and run the following two lines for data processed with software older
# than HIPE 9.0.

#decon_result = doDeconvolution(obs=surveyobs)
#decon_result_ssb = decon_result["ssb"]

# or for data processed with HIPE 14 and later:
#decon_result_ssb = decon_result["dataset"]

#
# This needs to be specifically a Simple Spectrum for the conversion to Jansky
decon_ss_K=SimpleSpectrum(decon_result_ssb)
#
# Now convert to Jansky. The user will have to assign a source size for this
# conversion to work.
# For this demonstration, a point source (size=0.0) is assumed.
cal = surveyobs.calibration
decon_ss_Jy = convertK2Jy(spectrum=decon_ss_K, size=0.0, cal=cal, overwrite=False)
#
# decon_ss_Jy can now be sent to VO tools
```

# Chapter 25. Reference Frames in HIFI data

Last updated: Feb 18, 2011

## 25.1. Introduction

Given HIFI's high spectral resolution, a relativistic treatment is necessary when accounting for spacecraft motion. Lorentz transformations take observed frequencies from one inertial frame to another. Over the ten seconds or so of a single integration, we can ignore the spacecraft acceleration. We can likewise ignore General Relativistic (GR) effects due to Herschel's elliptical orbit; and we choose to ignore GR effects when observing near interfering bodies such as Jupiter.

### 25.1.1. HSO Frame

The internal frequency calibration schemes for HRS and WBS, executed within the HIFI Level 0.5 pipeline, produce observed frequencies in the spacecraft frame ("HSO"). They are in the IF scale; the frequency of the detected photon is simply  $IF + LO$ . There are two observables of interest: the frequency of the incident wave, and its direction. In fact, we don't know the direction from which the radiation was incident because the HIFI beam is of finite size (between 10 and 45 arcseconds FWHM), and additionally has a small pointing uncertainty (about 2 arcseconds). Often the signal will come from a resolved source. We assume the direction of incidence is the boresight of the beam as reconstructed in the pointing product; the frequency error due to the uncertainty in signal arrival direction is small, of order 1 kHz per arcsecond of error. Note that because of stellar aberration, the direction of incidence in the HSO frame differs from that observed in, say, the Solar System Barycenter (SSBC) frame, by up to about 23 arcseconds. In practice, the pointing information provided in the pointing product has been de-aberrated, and so is in the SSBC frame.

Only the frequency remains known in the HSO frame alone; everything else, including the HSO motion, is known in the SSBC frame.

### 25.1.2. SSBC Frame

The Solar System Barycenter is the fundamental inertial frame for calculations involving the motion of HSO. The state vectors  $(r,v)$  of solar system objects (SSOs), including HSO, expressed in this frame have as origin the SSBC and as reference directions the International Celestial Reference Frame axes.

The motion of HSO with respect to the Geocenter is determined to an accuracy of about 5 cm/s by the usual tracking techniques. The Geocenter is tied to the Solar System Barycenter through the JPL DE405 planetary ephemerides to a precision of mm/s. Because the HSO motion, target coordinates, and telescope pointing are all defined in this frame, the transformation from the HSO frame to any other passes necessarily through the SSBC.

It's important to keep in mind in which frame observables are defined; in the equations below we use the convention that subscripts refer to the object of interest, and superscripts to the frame in which the observable is measured. For example, the direction of a SSO as seen by the telescope is  $\hat{\mathbf{P}}_{SSO}^{HSO}$

and as seen by an observer at rest in the SSBC, it is  $\hat{\mathbf{P}}_{SSO}^{SSB}$

A signal is incident upon the spacecraft and detected at frequency  $\nu^{HSO}$

The transformation of HSO-centric frequencies to SSB-centric is described by the relativistic Doppler formula:

$$\frac{\nu^{\text{HSO}}}{\nu^{\text{SSB}}} = \gamma_{\text{HSO}}^{\text{SSB}} (1 + \beta_{\text{HSO}}^{\text{SSB}} \cdot \hat{\mathbf{p}}^{\text{SSB}})$$

1

where

$$\beta = \mathbf{v}/c, \gamma = 1/\sqrt{1 - \beta^2}$$

and  $\hat{\mathbf{p}}^{\text{SSB}}$  is the de-aberrated direction of telescope pointing (the J2000 coordinates of the beam boresight at the time of observation).

### 25.1.3. LSR Frame

It might be useful to review the definition of the Local Standard of Rest. Take any point in the Galactic plane, and imagine there exists a circular orbit about the Galactic Center that passes through that point. The circular velocity defines the Local Standard of Rest for that position. Such a point coincident with the Sun defines the Solar Local Standard of Rest (LSR). The Sun has a peculiar velocity with respect to the LSR, which can be estimated in different ways. The LSR so defined is also called the Dynamic LSR, referring as it does to the rotation curve of the Galaxy. In practice, the Sun's peculiar motion with respect to the LSR has also been inferred from the mean motion of bright stars in catalogues or in the solar neighborhood. The LSR defined by this calculation of peculiar motion is called the kinematic LSR (LSRk), and is the more commonly used convention. However, there is not a single standard value for the LSRk. Further, it is not very close to any physical velocity of interest; it is only a common convention.

We take as our definition of the LSRk frame: the motion of the SSBC with respect to the LSRk is 20.0 km/s toward ra, dec = 18h03m50.29s, +30:00:16.8 (J2000). This is a common observatory standard, and adopted by many astronomical software suites such as CASA, SLALIB, and CLASS.

Frequencies in the LSR frame are derived from the SSB frame by a Lorentz transform:

$$\frac{\nu^{\text{LSR}}}{\nu^{\text{SSB}}} = \gamma_{\text{LSR}}^{\text{SSB}} (1 + \beta_{\text{LSR}}^{\text{SSB}} \cdot \hat{\mathbf{p}}^{\text{SSB}})$$

And using the relativistic Doppler formula, LSR frequencies can be calculated directly from observed HSO frequencies and SSB-centric known quantities:

$$\nu^{\text{LSR}} = \nu^{\text{HSO}} \frac{\gamma_{\text{LSR}}^{\text{SSB}} (1 + \beta_{\text{LSR}}^{\text{SSB}} \cdot \hat{\mathbf{p}}^{\text{SSB}})}{\gamma_{\text{HSO}}^{\text{SSB}} (1 + \beta_{\text{HSO}}^{\text{SSB}} \cdot \hat{\mathbf{p}}^{\text{SSB}})}$$

2

### 25.1.4. Source (nonSSO) Frame

Because the 3-velocity of a star or other such target is unknown, a transformation to its comoving frame is impossible. What may be known about the object is that a spectral line appears shifted from its expected rest frequency, and that shift can be interpreted as a radial velocity. One would like to see the relative velocities of other spectral lines, with a view to constraining a dynamical model of the object. In this context, transforming frequencies to the source frame is only a shift of the frequency axis already defined in an inertial frame (e.g. SSB, LSR), and then expressing the frequencies as velocities according to some convention. There are three operative conventions for expressing redshift as a velocity:

Convention	$\beta \rightarrow \nu$	$\nu \rightarrow \beta$
radio	$\frac{\nu_{\text{rec}}}{\nu_{\text{emit}}} = 1 - \beta$	$\beta = \frac{\nu_{\text{emit}} - \nu_{\text{rec}}}{\nu_{\text{emit}}}$

Convention	$\beta \rightarrow \nu$	$\nu \rightarrow \beta$
optical	$\frac{\nu_{\text{rec}}}{\nu_{\text{emit}}} = \frac{1}{1 + \beta}$	$\beta = \frac{\nu_{\text{emit}} - \nu_{\text{rec}}}{\nu_{\text{rec}}}$
relativistic	$\frac{\nu_{\text{rec}}}{\nu_{\text{emit}}} = \sqrt{\frac{1 - \beta}{1 + \beta}}$	$\beta = \frac{\nu_{\text{emit}}^2 - \nu_{\text{rec}}^2}{\nu_{\text{emit}}^2 + \nu_{\text{rec}}^2}$

The relativistic definition would be correct if the relative velocity between observer and source were purely radial. The radio and optical definitions are two linearisations of the relativistic equation, and their difference is quite large at HIFI bandwidth  $\Delta\nu$  about 4 GHz and frequencies  $\nu_{\text{rec}} \approx 500$  GHz  $\Delta\beta \equiv |\beta_{\text{optical}} - \beta_{\text{radio}}| \approx (\Delta\nu)^2 / \nu_{\text{rec}}^2 \sim 5 \times 10^{-5} \Rightarrow \Delta v \approx 16$  km/s

The HIFI Pipeline task `DoVelocityCorrection` will, if requested, recast the frequencies in the data to the "source frame" of a nonSSO target, but it is not a real frame change and a Lorentz transform is not performed.

It is often interesting to view spectra plotted on an axis representing speed. In recent versions of HIPE (6.0.1360 or earlier), the Spectrum Explorer display tool uses the relativistic convention when displaying frequencies as velocities; however, the `ConvertFrequencyTask` uses the radio definition.

### 25.1.5. Source (SSO) Frame

When observing objects within the solar system, we know or can estimate their full 3-velocity, and so a real Lorentz transform to the object rest frame is performed. The manner is thus: the state  $(r, v)$  of Herschel is known as a function of time, as is the SSO's. A photon received by HSO at time  $t$  was emitted by the SSO at time  $t - \text{LT}(t)$ , where  $\text{LT}(t)$  is the light-travel-time between the two. The frame to which we transform is the comoving frame of the SSO at the retarded time  $t - \text{LT}(t)$ .

$$\text{LT} = \frac{|\mathbf{r}_{\text{SSO}}(t - \text{LT}) - \mathbf{r}_{\text{HSO}}(t)|}{c}$$

LT is computed iteratively; three iterations are sufficient to achieve a precision of less than a millisecond within the orbit of Pluto, which is less than the error due to ignoring General Relativity. Once the SSO state at the retarded time is known, frequencies are transformed by equation 2 (with SSO substituted for LSR).

## Chapter 26. Relative performance of the HIFI spectrometers

As the HRS always has a better spectral resolution than the WBS, it is expected from the radiometric formula that HRS data should have an rms noise greater than that of WBS spectra.

In addition, the HRS has an efficiency of, on average, 81%, which also has the effect of increasing the HRS rms with respect to the WBS rms, for a similar integration time and a similar spectral resolution.

Furthermore, there is a small degradation from the expected rms in HRS spectra in bands 1 and 2, because HRS attenuators tune on the hot load rather than the sky.

The overall theoretical ratios of the expected HRS rms with respect to the WBS rms for simultaneous observations, assuming the WBS efficiency and spectral resolution to be 100% and 1.1 MHz respectively, are:

<b>HRS resolution</b>	<b>Theoretical ratio</b>
Wide	1.68
Low	2.37
Nominal	3.35
High	4.74

A statistical study of data collected during the Performance Verification and Science Demonstration Phases of the mission showed that the HRS to WBS rms ratios are as good as expected, for all observing modes and spectral resolutions.

# Chapter 27. Dealing with memory issues and slow performance.

Last updated: 17 February, 2015

## Memory Issues.

On occasion, one can run into the `java heap space` error when using HCSS software, especially when running the pipeline. Here are some things to help:

1. Delete any unneeded variables.
2. Limit the running of the pipeline to one spectrometer at a time.
3. Force the pipeline not to create a new variable each time you run it by giving the output variable the same name as the input:

```
obs = hifiPipeline(obs=obs)
#
# This can only be done in the command line,
# the GUI will automatically append a number to an output variable with the same
# same name as an input variable.
```

4. When installing HIPE, allocate a sensible amount of memory to it, i.e., a sizeable fraction of your RAM, but leave enough to run your browser and email. For example, allocate 2.5-3 Gb on a 4 Gb machine or 6-7 Gb on an 8 Gb machine. If, after the installation, you wish to modify your memory allocation, you can do so 1) by editing the following file `.hcss/Hipe.props` (`java.vm.memory.min` and `java.vm.memory.max`), or 2) by editing the HIPE preferences from the preferences panel (General->startup & shutdown).
5. To ease memory problems related to GUIs, you can set:

```
java.vm.options = -XX:MaxPermSize=256m
```

But note that this option for the compiler will disappear in Java 8.

6. The "garbage collection" command `System.gc()` is also useful to force clearing memory. HIPE will automatically do this when memory becomes too full.

## 7. Swap Store Properties:

It is possible to use the hard disk as swap space to preserve the memory available in HIPE, and HIPE does this by default. The following properties are defined to preserve computer memory. This becomes especially useful when pipeline processing long observations on a laptop, or on a pc with a 32 bit Operating System (TBC), and with average or limited memories capacities. However, any Task that uses or changes any `HifiProduct` (e.g., `HifiTimelineProduct`) will benefit from the use of swap space.

The following properties can be modified (in the `user.props` file or using the "Hifi Product" tab in `propgen`) to set or to configure the Swap mechanism.

- **hcss.hifi.pipeline.product.memory = true:** Setting the value of this property to "true" enables the swap mechanism. Note that the default value is "false".
- **hcss.hifi.pipeline.product.swapstore = "swapStore":** This is the name of the `LocalStore` where the temporary data will be saved. The default location is: `${user.home}/.hcss/lstore/swapStore`.

- **hcss.hifi.pipeline.product.swapratio = 0.25:** This property determines how much the swap mechanism is used and is used to set the threshold level of free memory. When a new dataset is set or retrieved from the HifiProduct, the HifiProduct will check the size of the dataset and the free memory in the system. If the condition:

$(\text{memory free}) * \text{swapratio} < \text{dataset size}$

is met, then all the floating datasets contained in the HifiProduct will be saved in the swap store.

This property should have value between 0 and 1 and has a default value of 0.25.

If the value is 0 all datasets will always be stored in the swap store. This is safe, but it could create performance delay (in the time needed to process the pipeline) due to the access time to the hard disk.

In the case of long observations, setting the property to 1 could be dangerous because memory problems (like `Java . heap . space exception`), may still occur, although the pipeline will try to have the best performance possible.

- **hcss.hifi.pipeline.product.savedisk = true:** This property determines whether an existing observation in the swap store should be overwritten or not. It is strongly suggested to keep the value = true, otherwise the space used in the hard disk will increase in proportion to the number of times a product is saved in the swap store.



#### Note

**SwapUtil Class:** At the moment, the pipeline does not clean the swap store after the processing. To avoid the swap store completely filling the hard disk when many observations are processed, it is suggested one manually remove the swap store by either deleting the swapStore directory, or in HIPE:

```
from herschel.hifi.pipeline.product import SwapUtil
SwapUtil.delete()
```

#### Performance.

- The system opens every lstore pool on one's disk, so the more pools you have in this directory the more disk activity will occur.
- The HIPE session history is stored in `~/hcss/apps/hipe`. This is where information about preferences, and which files and pools you have accessed to is stored. If HIPE seems to be holding on to some old behaviour unexpectedly, then it might be helpful to try renaming this directory.