Herschel Data Processing HowTo Documents

Version 1.2, Document Number: HERSCHEL-HSC-DOC-1199
28 September 2009

Herschel Data Processing HowTo Documents:

Table of Contents

Chapter 1. HowTos Preface

1.1. Introduction

This document is intended to provide a general overview of the main interface to the Herschel Data Processing (DP) software referred to as HIPE (Herschel Interactive Processing Environment). HIPE provides a GUI plus command-line access to the data processing capabilities of the Herschel Common Science System (HCSS).

The intended audience is for the general astronomer new to the DP system who wishes to start with HIPE for doing standard data analysis. More specialist analysis is possible and scripting, allowing for batch processing, can be done within the system.

For those interested in becoming more advanced in the system the `"DP Basic User's Manual"` is also available from within the DP help system and on-line as PDF and HTML documents.

1.2. Change Log

The following items have been changed or updated between document version 0.6 and 0.7 Changes are with respect to user release 1.0 of the HCSS and DP system.

- Chapter 3: HSA interface -- updated to include the new interfaces allowed by HSA v1.5.
- Chapter 8: Save and restore from pools updated to include new views available on creating storages, registering them and using them.
- Chapter 2: HIPE overview -- updated to include short introduction to the new views in HIPE; Import Herschel data into HIPE, Export Herschel data from HIPE, Product Storage Manager and Save Products to Store.

The following items have been changed or updated between document version 0.5 and 0.6 Changes are with respect to user release 0.6.7 of the HCSS and DP system.

- PACS pipeline chapter -- complete update for photometer/spectrometer and different modes
- HowTo display spectra -- now includes SPIRE spectroscopic example.
- HIFI pipeline -- complete update for running in HIPE or via command line.
- SPIRE pipeline -- complete update to most recent photometer/spectrometer pipelines.
- Additional chapter on Spectral Cube analysis.

Changes were made in going from version 0.4 to version 0.5. The following items have been changed or updated. Changes are with respect to user release 0.6.7 of the HCSS and DP system.

- PACS pipeline chapter -- access to calTree defined
- HowTo plot -- Tex-like features and properties interactions updated. Several typos (including in example scripts) removed.
- HIPE overview updated to reflected changes to perspectives and views in most recent release.
- Minor updates (links, typos etc.) in Archive, DataAccess, Save and Tables HowTos.

Substantial changes were made in going from version 0.3 to version 0.4. The following items have been changed. Changes are with respect to user release 0.6.6 of the HCSS and DP system.

- HIFI pipeline chapter substantially updated
- SPIRE pipeline chapter updated to include sample output products
- HowTo chapters on spectral display, image display and arithmetic , spectral arithmetic, spectral fitting and image analysis all added.
- HowTo Access Data substantially changed to include updates in access to the Herschel Science Archive.
- HowTo Save and Read data to and from FITS and ASCII files -- information expanded.

The following items have been changed in version 0.3 from version 0.2. Changes are with respect to user release 0.6.5 of the HCSS and DP system.

- HIFI pipeline chapter added
- PACS pipeline chapter added
- SPIRE pipeline chapter updated to include sample output products
- HowTo plot chapter updated.
- HowTo save chapter updated, including missing image.

The following items have been changed in version 0.2 from the original version (v0.1). Changes are with respect to user release 0.6.4 of the HCSS and DP system.

- Update of HIPE overview to new views/capabilities of the HIPE environment.
- How to Save data section added.

Chapter 2. HIPE Introduction

-- Using the Herschel DP Interface

2.1. Introduction

The data processing application for Herschel Data Processing, Herschel Interactive Processing Environment (HIPE), provides an integrated suite of graphical interfaces that can interact with each other. It allows for interactively choosing your active data in your session, visualizing that data in various ways and selecting tools that can operate on the data. Both command-line and GUI interfaces to the user are available. High-level interactions, which can involve GUIs, are also echoed as commands on the command-line that allow the saving of commands used in a session and the generation of scripts from these interactions.

This chapter provides an overview of the fundamental elements of HIPE.

2.2. HIPE Philosophy

HIPE tries to embrace several aspects which affects both users and developers:

- An integrated application - giving access to all data processing functionality in a unified graphical interface
- A single look-and-feel for window layouts, toolbars, buttons and menus.
- A customizable layout - allow the user to decide which windows are relevant and how these windows are layed-out on screen.
- User guidance - including command-line echoing of main graphical functionality, allowing the user to learn the scripting language by interacting with the system interactively.
- Extendible application - allow the developer to add new bells-and-whistles which are automatically integrated

2.3. Installation and Startup of HIPE

Important

In case of any problems during installation please contact the Herschel Helpdesk via the Herschel Science Centre website.

HIPE is part of the Herschel Data Processing system. You can download an installer from the Herschel Science Centre at http://herschel.esac.esa.int/HIPE_download.shtml. Installation instructions are provided at the bottom of the page.

If you are an internal user, you can also download the software from these locations:

- More installers at http://www.rssd.esa.int/SD-general/Projects/Herschel/hscdt/docsDpInstallInfo.shtml.
- Installers for release candidates at http://www.herschel.be/twiki/bin/view/Hcss/HipeReleaseCandidates.
- Development versions via the Continuous Integration system at http://herschel.esac.esa.int/hcss/build.php. See links on the page for instructions.

HIPE needs the Java SE Runtime Environment version 6. We recommend to install the latest update for best performance. To see which Java version is installed on your machine type the following in a terminal window:

```
java --version
```

You can run HIPE on a server or individual workstation under one of these operating systems, either 32-bit or 64-bit:

- Windows XP (Windows Vista is not formally supported).
- Sun Solaris.
- Any Linux distribution compliant with the Linux Standard Base version 3.0 or higher. A list of compliant distributions is available on this page. HIPE may work on non-compliant distributions too, but this is not guaranteed.
- Mac OS X 10.5 Update 2 or later.

The minimum amount of memory needed is two GBytes. Four GBytes are the highly recommended minimum for data processing; more may be needed in some cases (e.g. large PACS maps).

The maximum amount of memory available to HIPE can be set by choosing the *Advanced* installation. To change the available memory afterwards, modify the properties `java.vm.memory.min` and `java.vm.memory.max` in the relevant property file, as shown below (see the *User's Manual* for more information on property files: Section 1.4):

```
java.vm.memory.min=64m
java.vm.memory.max=512m
```

The **Advanced** installation also allows you to specify the name and location of a Versant database. Only advanced users and calibration scientists should need to specify a database: if you are unsure about this option, you probably do not need it. Interaction with databases is explained in the *User's Manual*: Section 12.2.

Recommended browsers to visualize the HIPE Help System are the following:

- Microsoft Internet Explorer 6 or newer.
- Netscape 7 or newer.
- Firefox 1.5 or newer.
- Apple Safari.

Once the software is installed, HIPE can be started by several means. Using Windows, Herschel software can be started under the Start menu after a standard installation. Alternatively, HIPE can be started from a command line:

```
hipe
```

The installer will tell you where the `hipe` executable is located.

When starting HIPE under Windows, you might see two command prompt windows called `hipe` and `ia_hipe.exe` appear alongside the HIPE main window. Both can be safely closed once HIPE has started.

Upgrading and uninstalling

To uninstall HIPE, either follow the usual uninstallation procedure for your operating system, or execute the **uninstall_hipe** command in the `uninstall_hipe` folder.

To upgrade to a new version of HIPE, uninstall the existing version and install the new one.

2.4. Obtaining help in HIPE

You can open the HIPE Help System via the Help menu. Help pages will open in a new tab of your default web browser.

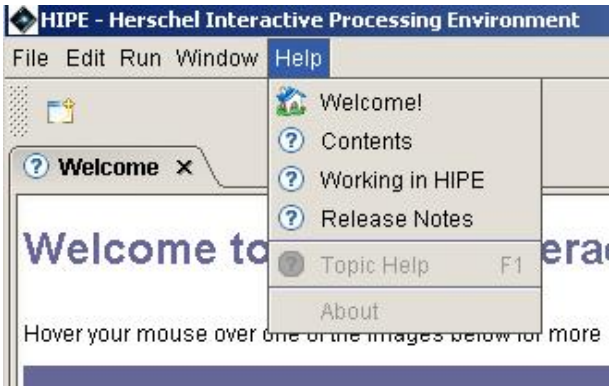


Figure 2.1. Accessing the HIPE Help System.

You can access the following help pages from the Help menu:

- **Contents:** this option will open the main help window. The table of contents on the left displays the available documentation, with more advanced material appearing towards the bottom (see Figure 2.2).
- **Working in HIPE:** this option will open Chapter 2 of the *HowTo's* manual, the same you are reading now (see Figure 2.3).
- **Release Notes:** this option will open a document with the most recent additions and changes to the system.

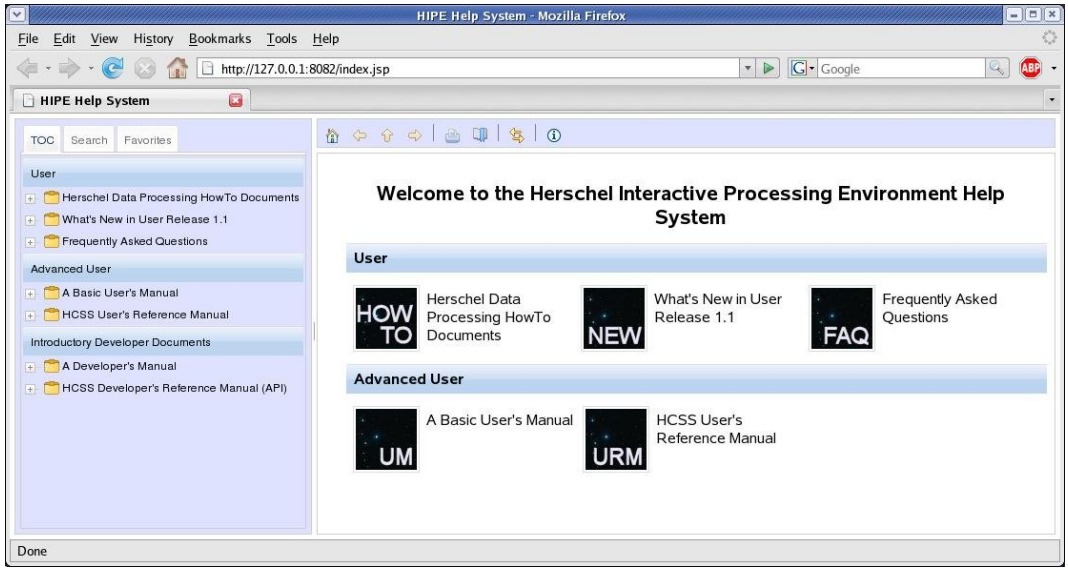


Figure 2.2. The HIPE Help System home page.

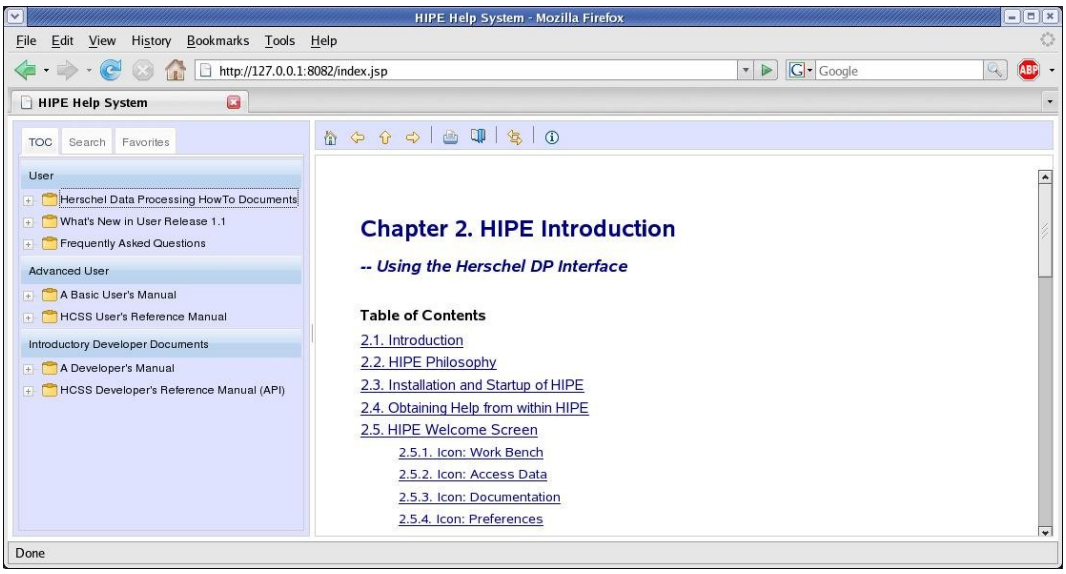


Figure 2.3. Help on working with HIPE.

2.4.1. Obtaining help outside HIPE

When you quit HIPE, links in any help pages will not work anymore. Even if you restart HIPE, you will have to open the Help System again.

To access help outside HIPE you can use the **show_help**, which resides in the same folder as the **hipe** executable. This will bring up the following window:

Figure 2.4. The standalone Help System.

Clicking on the Display button will start the HIPE Help System.

For advanced users: accessing the Javadoc

If you are comfortable with Javadoc documentation, you can access it in the HIPE Help System home page by clicking on *HCSS Developer's Reference Manual (API)* in the table of contents. To obtain the traditional frame-based Javadoc layout, click on the *FRAMES* link on any Javadoc page. To get back to the HIPE Help System layout you will have to use the Back button of your browser (clicking on the *NO FRAMES* link will not work). To have both layouts available, you may want to open the Javadoc layout in a new tab or window of your browser, by right-clicking on the *FRAMES* link.

2.5. The HIPE Welcome Screen

When you start HIPE you are greeted by a Welcome screen with six icons, which are described in the next sections. Note that the *Updates* icon is not yet available. Placing the mouse over each of the icons on the screen provides a small description that appears along the bottom of the HIPE window (see Figure 2.5).

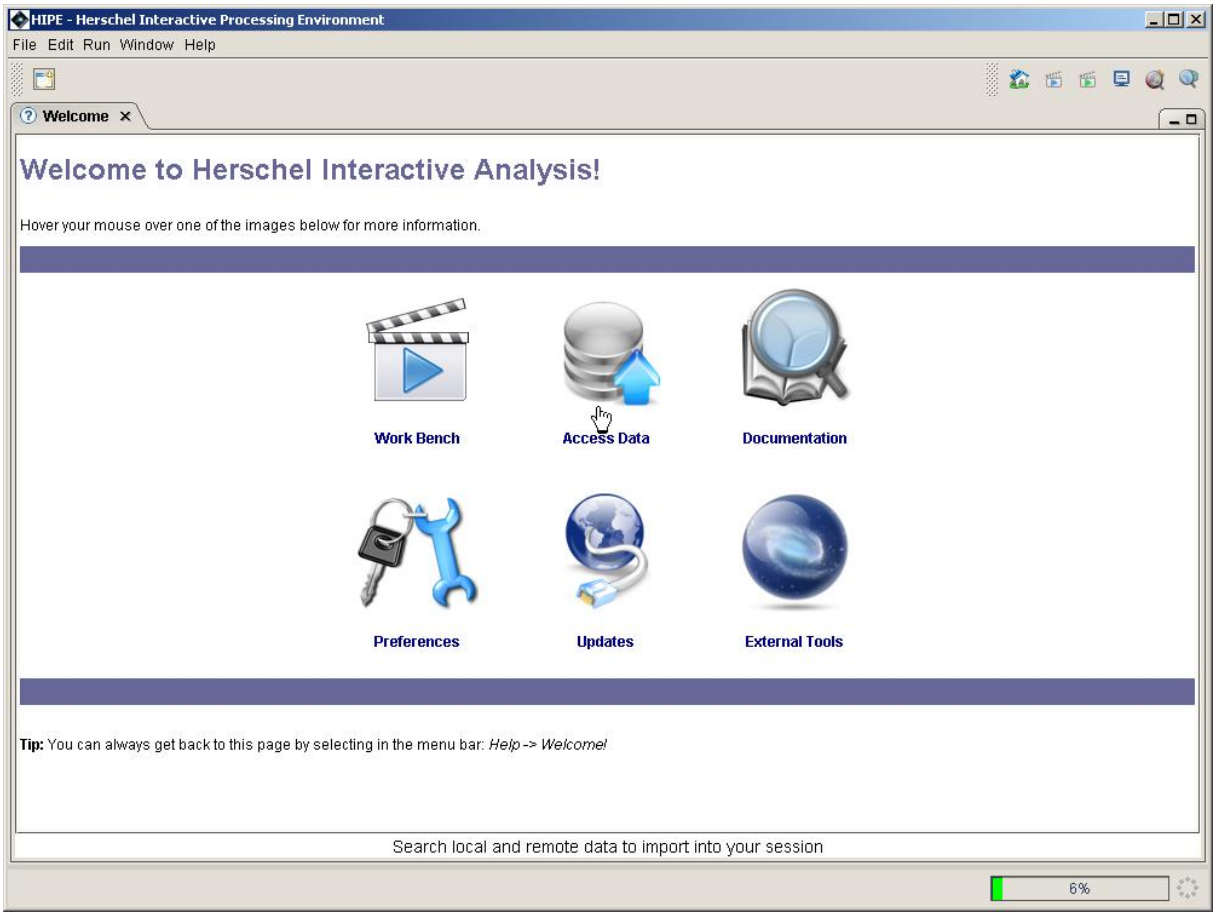


Figure 2.5. Information on 'Welcome' screen icons. See bottom strip of the HIPE screen for the explanation of each icon the mouse is placed over. In this case the Access Data view is stated as being accessible via the icon the mouse is hovering over.

Note that a tool bar exists at the very top right of all window displays of HIPE. This tool bar and its uses are discussed in the section on HIPE perspectives (see Section 2.6). However, just to note here that the Welcome screen can always be returned to by using the 'Help' pulldown menu to 'Welcome' (see Figure 2.6)
. The Welcome screen is also available using the first icon in th elist to the top right of the HIPE screen ( ).

Figure 2.6. HIPE Welcome screen access.

2.5.1. Icon: Work Bench

Clicking on the  icon takes the user to the workbench perspective (for information on perspectives in HIPE see Section 2.6). The default view of the workbench is shown in Figure 2.7. This is the main work area for doing data analysis. Here we can look at data values, plot spectra and images, create scripts for batch processing and run analysis tools. The contents of the workbench can be updated with various "Views" available under the Window pull-down menu (see Section 2.8 on available Views).

The current default work bench is a somewhat slimmed-down version of the full work bench. Either perspective on the system can also be be provided via use of the "Window" menu. Selection of the "Show Perspectives" and either "Full work bench" or "Work bench" provides the two main default perspectives for when doing work in HIPE.

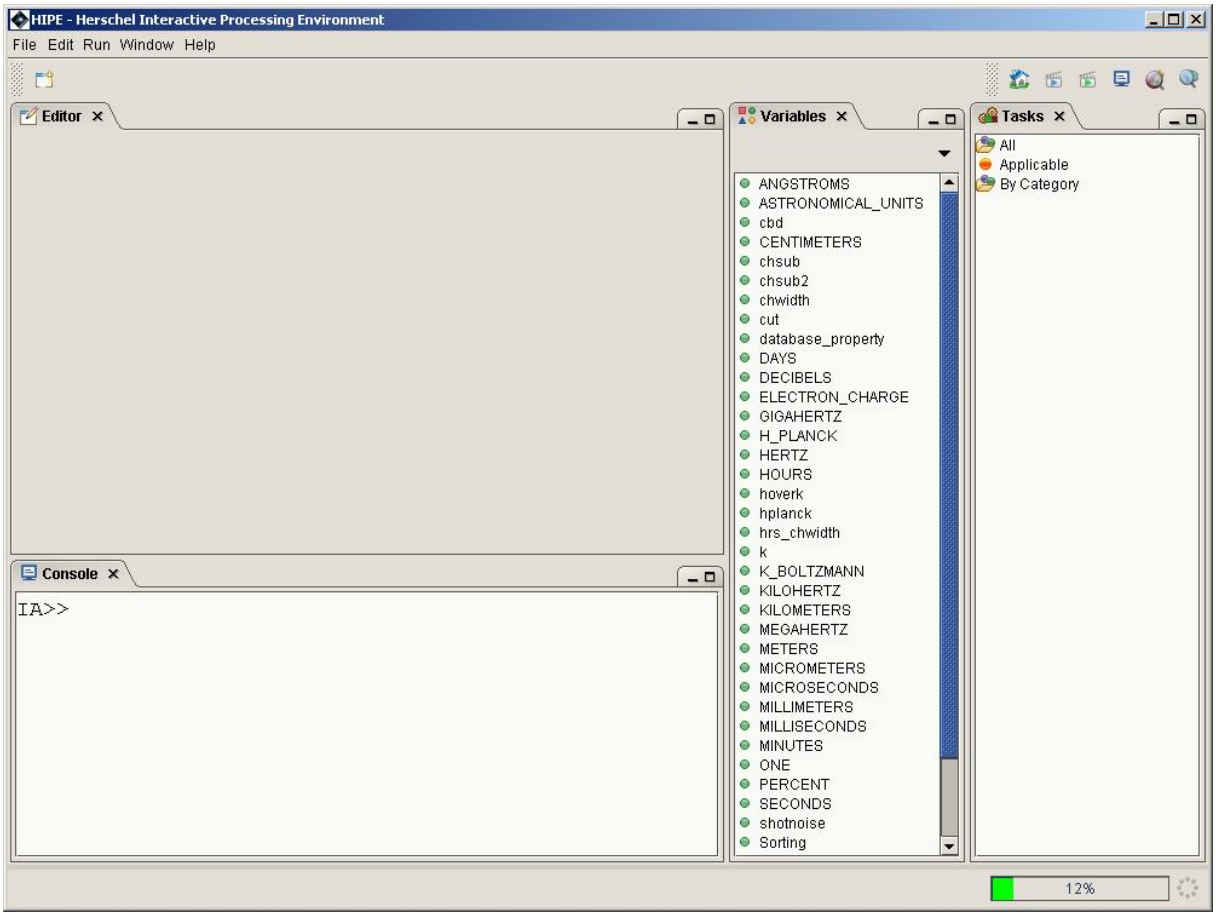


Figure 2.7. HIPE default view of the workbench perspective.

2.5.2. Icon: Access Data

The  icon opens up a replacement window in HIPE that provides access to data held in databases both locally or at a remote site (for example the Herschel Science Archive). It also allows the import of FITS and ASCII table files into and out of a DP session.

The access tools allow the user to search and do queries on stored data and its attributes in order to make it accessible within the processing session.

Four icons appear that allow import/export to databases, direct access to the Herschel Science Archive or import/export of FITS or ASCII table files (see Figure 2.8).

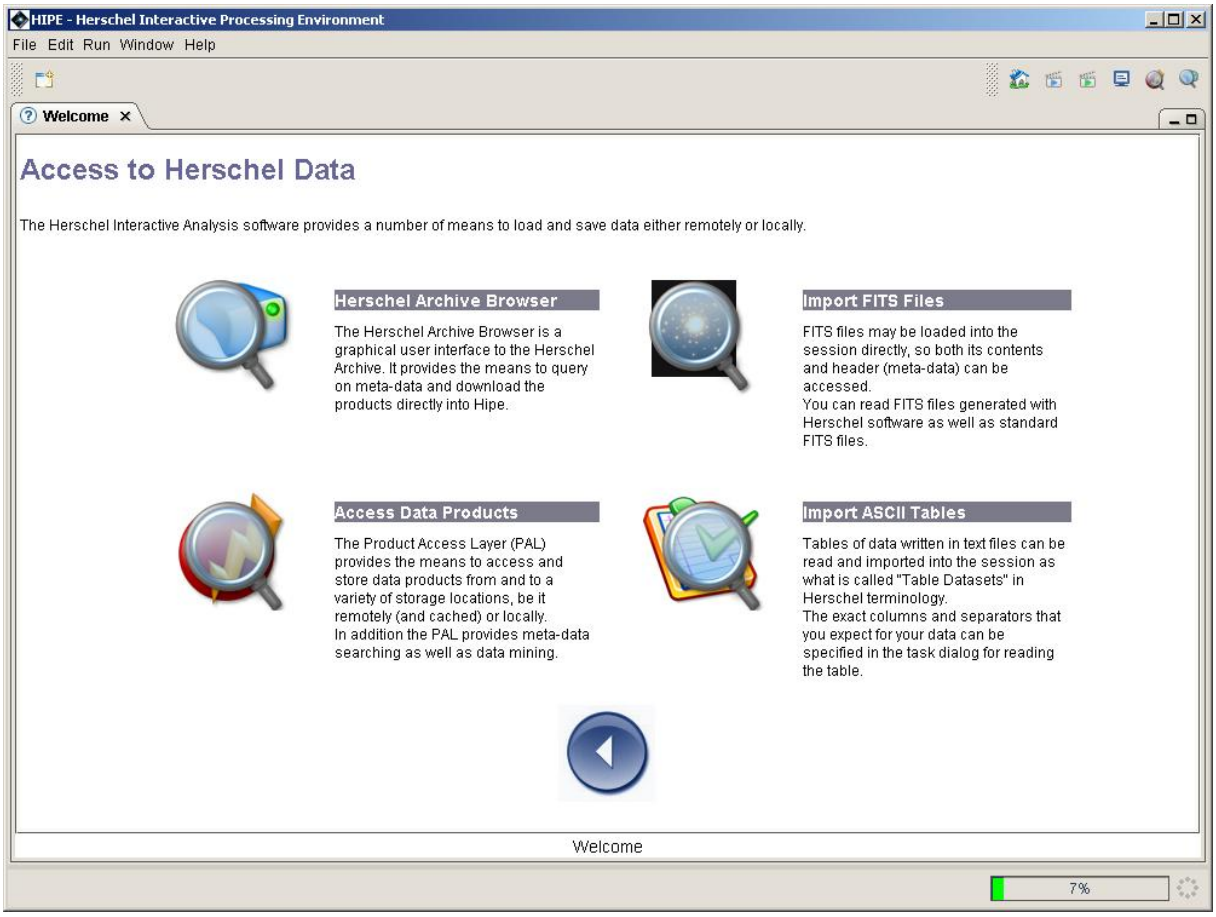


Figure 2.8. HIPE default view of the workbench perspective.

Clicking on the "Herschel Archive Browser" icon opens up the Herschel Archive perspective, while the Data Access icon takes the user to the Product Access Layer perspective (see Section 2.6 for information on these perspectives). The means for actually bringing data into the system is described in detail in the *HowTo Access Data*.

For FITS and ASCII I/O the other icons produce perspectives that allow for this which are based on the default Work Bench plus the simple FITS archive tool or ASCII archive tool respectovely. These are discussed more thoroughly in the *HowTo save and restore data (ASCII and FITS)*.

At the bottom of the screen is a large back-arrow button that allows the user to return to the original "Welcome" screen.

2.5.3. Icon: Documentation

The  icon allows access to the complete DP release documentation tree. After clicking on this icon, documentation is provided via a web browser and uses the Eclipse software system which comes with the HCSS build. The user is able to get the top-level How-To information that explains such basic functionality as accessing data in a database, displaying images and spectra, plus basic image and spectral analysis for Herschel (also see Section 2.4).

Links are also provided to documentation that explain the scripting capabilities and use of the commands on the command-line of a console window. This allows the user access to the full power of the system as well as the creation of his/her own batch mode processing. The scripting language has

great similarity to the Jython scripting language and borrows many of the items from that language. This is contained in the "Basic User's Manual".

User task commands, numerical package and product storage information is also available in the User's Reference Manual (URM). The URM provides a short inrtroduction to any of the commands available. Help for a given command displays the URM contents for that command.

Developer documentation for the complete system is available. These are in JavaDoc format (described in Chapter 9 of the "Basic User's Manual"). Any of these commands may be used at the console command line or within scripts produced for the DP system.

2.5.4. Icon: Preferences

The  icon allows you to access the *Preferences* dialogue window shown in the following figure:

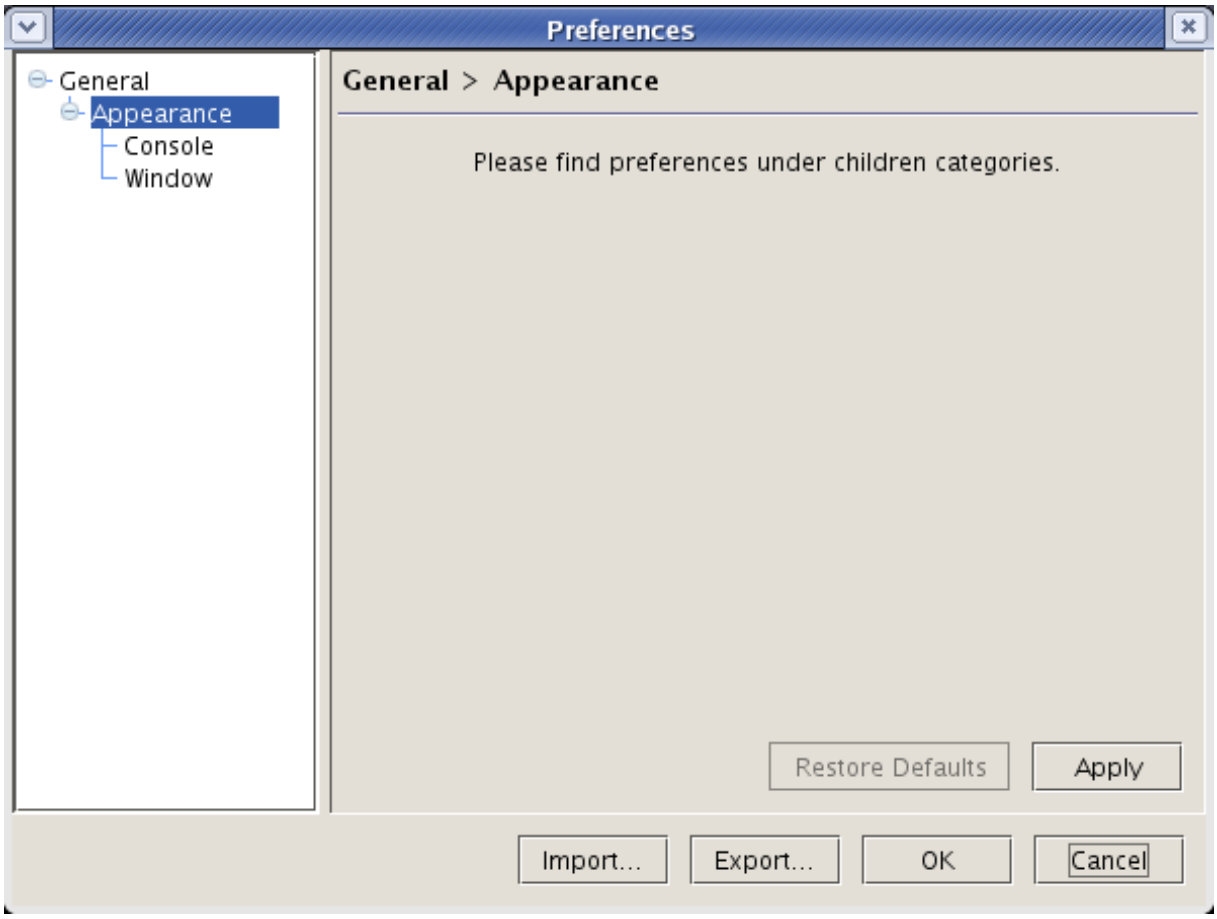


Figure 2.9. The HIPE preferences window.

You can set preferences in the available categories and revert to the default values at any time. You can also export/import the values of the preferences to/from an XML file.

2.5.5. Icon: Updates

The icon  functionality is NOT IMPLEMENTED YET. Clicking on this item will (in future) allow the user to search for software updates available from the Herschel Science Centre.

At the bottom of the screen is a large back-arrow button that allows the user to return to the original "Welcome" screen.

2.5.6. Icon: External Tools

The icon  takes the user to a set of icons linking to Virtual Observatory tools, including VOSpec, VOPlot and Aladin. Included in this listing is the Herschel Science Archive (HSA) browser, as Herschel components are VO-compliant. The HSA also uses a VO-like interface with HIPE. Clicking on any of the icons launches the external VO tool. Help and assistance with these tools are provided separately from within the tools or associated administrative website, except for the HSA browser interface which is described in this manual.

At the bottom of the screen is a large back-arrow button that allows the user to return to the original "Welcome" screen.

2.6. HIPE Perspectives

When going to the workbench or using the welcome icon link to the data access capabilities of HIPE, the user is presented with a "perspective". A "perspective" is a presentation of the system that is made available to the user through a set of "views" (basically separate windows within the environment that provide particular capabilities). The following section discusses the views the user can have, but in this section we describe perspectives and in particular the default workbench perspective. We also discuss how the user can control a perspective to make it as simple or as complex as wished.

HIPE is built-up from several graphical elements, of which the fundamental ones are shown in Figure 2.10, which provides the full work bench. A perspective is a collection of graphical windows ("views") organized in a way to focus the user on doing a specific job within the whole suite of jobs that a user can and will do within the system. It may consists of one or more views and, optionally, the editor area; these windows are then organized in tabbed panes and split panes. Many of the views also contain their own toolbars. These toolbars are in addition to the toolbars for HIPE displayed at the top left (editor capabilities for editor window view) and right of the HIPE screen (icons providing access to full set of defaults perspectives -- hover mouse over icon to view perspective name).

Figure 2.10. A single element (view) for a HIPE perspective.

2.6.1. Available Default Perspectives

There are five perspectives that come pre-packaged in the system. *These can always be obtained by using the toolbar at the top of the HIPE window. Click on "Window" and pull down to "Show Perspectives", which provides the list.*

2.6.1.1. Product Access Layer Perspective

The **Product Access Layer** perspective provides a convenient means of getting and briefly viewing data from databases and data stores -- both locally and remotely stored. This is illustrated in Figure 2.11. There are 4 windows ("views") including an editor where DP scripts can be created (see the DP User's Manual).

Data can be queried from a locally stored database (default is under the ~/.hcss directory) or remotely registered database using the "Data Access" view seen to the left of the perspective (see Section 2.8.3). More information on how to get data from databases and the Herschel Science Archive is available from the chapter *HowTo Access Data*.

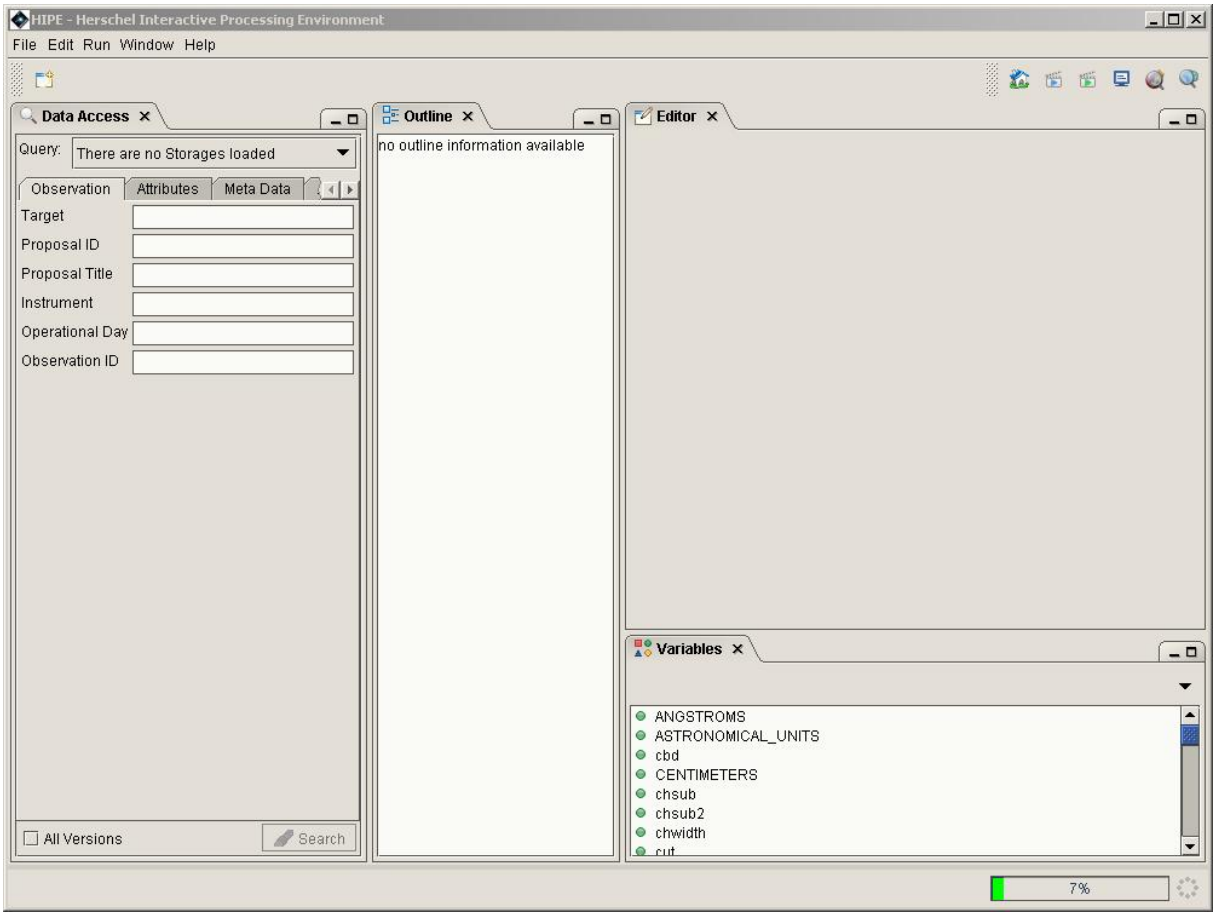


Figure 2.11. HIPE Product Access Layer (PAL) perspective. This provides access to data stores both on-line and on the user's own computer.

2.6.1.2. Classic(JIDE) Perspective

The **Classic JIDE** perspective provides a scripting environment with 3 windows that provide an editor/debugger window, a console window and a log window. This is the basic view of the system used during earlier development of the DP system (see Figure 2.12). A new Jython (DP) script window can be added by clicking on the  icon at top left of the Editor window. More information on the Editor view can be found at Section 2.8.4. The DP User's Manual, available under the Help menu, also provides significant further help on JIDE itself and the HIPE/JIDE view in creating user scripts.

The same perspective can be obtained by clicking the  icon to the top right of the HIPE window.

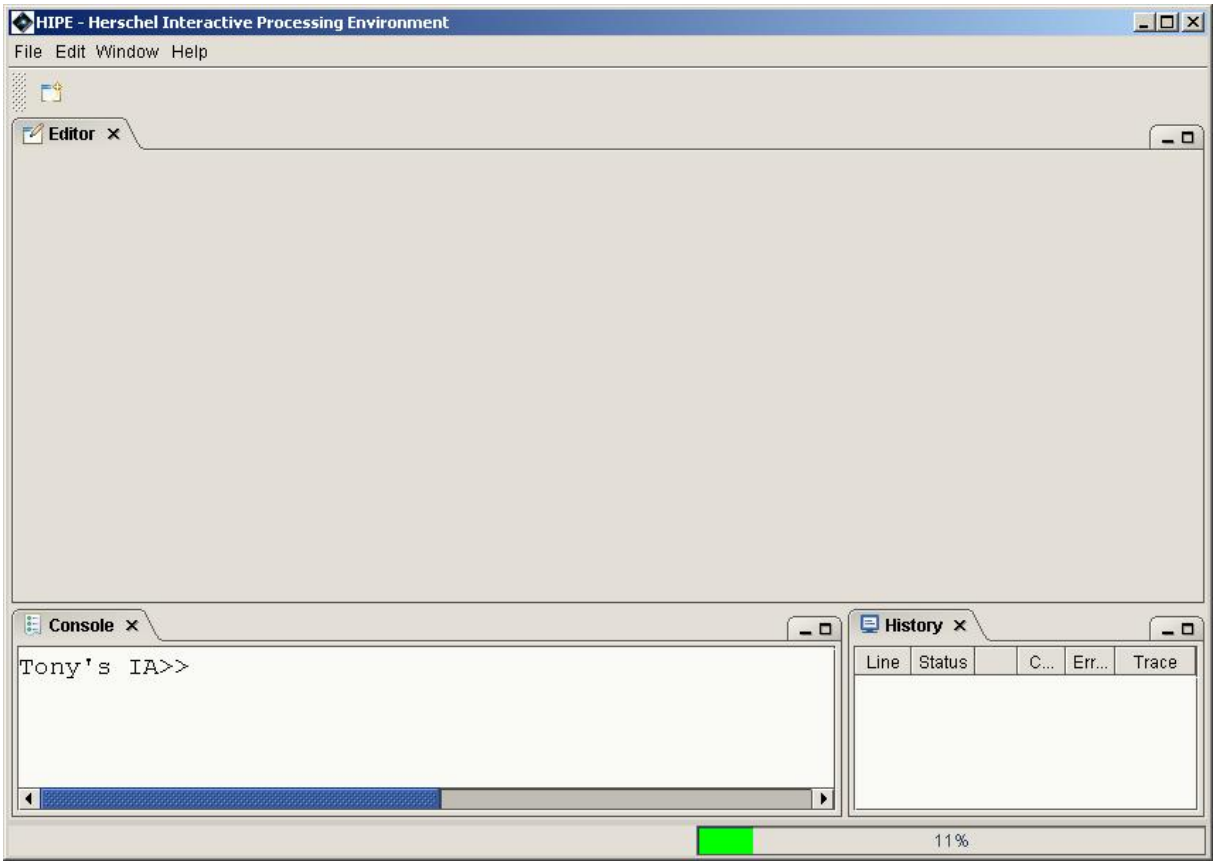


Figure 2.12. HIPE's 'classic' JIDE perspective.

2.6.2. The Full Work Bench Perspective

The **Full Work Bench** perspective provides a general environment with multiple windows, five of which are prominent (editor, console, variable list, outline, run tools). Other windows/views are available by clicking on the tabs, e.g., Navigator, Classes (see Figure 2.13).

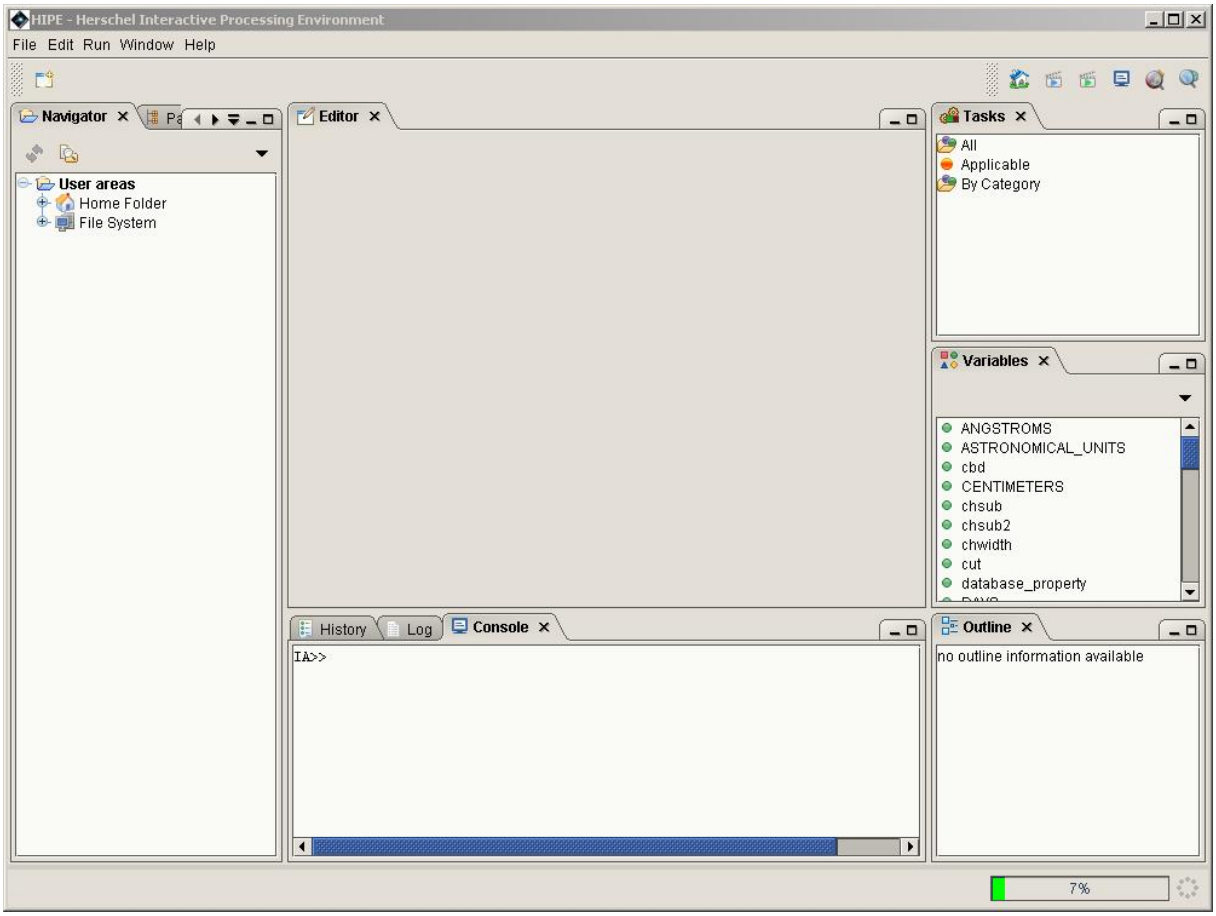


Figure 2.13. HIPE view of the full work bench perspective.

2.6.3. The Work Bench Perspective

The **Work Bench** perspective provides a slimmed-down general environment similar to the work bench but with only with four windows (views). The editor, console, variable list, outline, tasks views are available (see Figure 2.14).

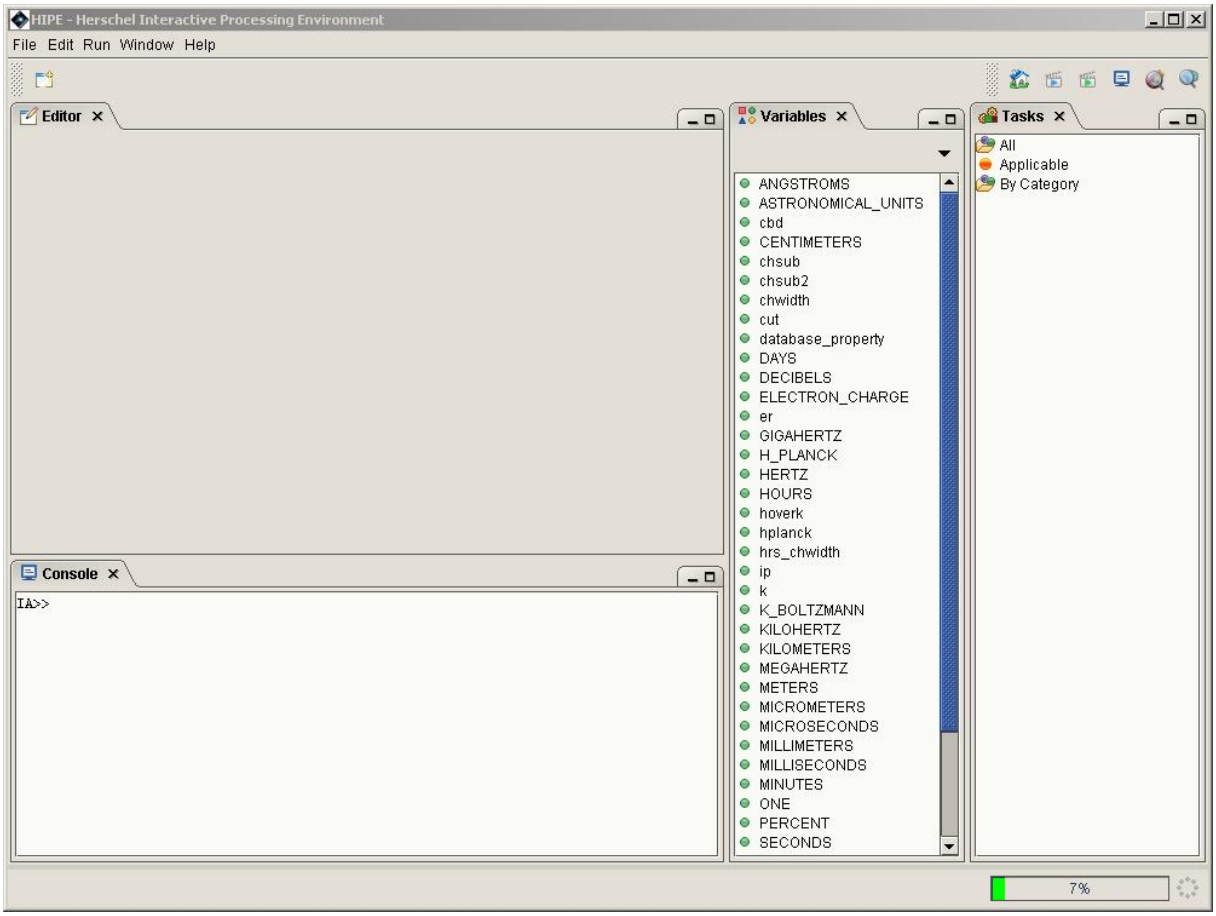


Figure 2.14. HIPE default view of the work bench perspective.

2.6.4. Archive Browser

The **Archive Browser** perspective provides a convenient means of querying and obtaining data from the Herschel Science Archive (HSA). There are three views related to providing log-in information for the HSA, the connection to the HSA (via plastic VO protocol) and the loading of selected data from the archive (see Figure 2.15).

Queried data appear under a single, selectable variable in the DP session (under Variables view) and a click on the variable allows its outline to be provided in the Outlines view. These two views are described in more detail later in this chapter. Further information on how to get data from databases and the Herschel Science Archive is available from the chapter *HowTo Access Data*.

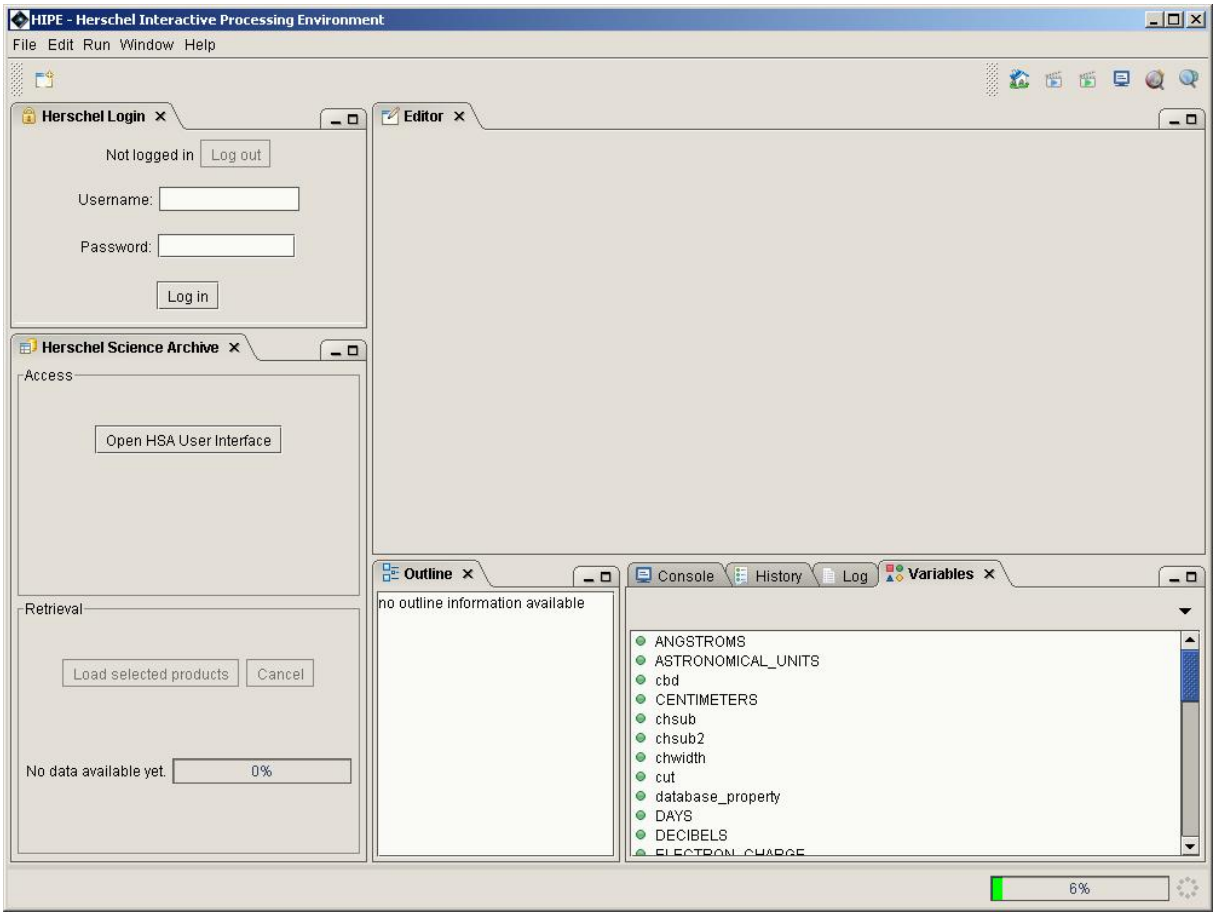


Figure 2.15. HIPE Archive Browser perspective. This provides access to the Herschel Science Archive (HSA).

2.7. Changing HIPE Perspectives

Changing a perspective to a worksurface that a user prefers can be done in various ways. Each window can be resized or dragged to different areas of the workspace. Also, new views can be added to a perspective.

2.7.1. Adjusting Individual Views

Each individual window can be adjusted in the following ways.

- **Window resizing**. These can be adjusted in a standard way. To the top of each window, the cross (X)on the tab being clicked removes the window/view. The underscore line minimizes a window (_) while the window can be maximized or returned to its original size by clicking of the box figure in the tab at top right. Minimized windows appear to bottom left of the workspace (see Figure 2.16). Holding the right mouse button down while on the window tab also provides a menu which includes the same options.

  Clicking and dragging borders of each of the windows allows for expansion in any direction of any of the views.

- **Window Tab Placement**. A right click on the view tab provides a pull-down menu that allows some default window resizing and also tab placement and direction of written label (see Figure 2.17).

- **Moving Views**. Windows can be moved inside the HIPE workspace by clicking on the window itself and dragging to another part of the worksurface. Outline black boxes appear on the screen indicating where the window would be if the mouse button was released at that point.

It is also possible to completely *Undock* a window view by holding the right mouse button down while on the window tab. Pulling down on the menu to "Undock" allows the view to become a separate window that can be moved completely off of the HIPE surface (see Figure 2.18 for an example). To move this undocked window the user need only click on the top, blue part, of frame of the window and drag to wherever he/she wishes on the screen surface.

- **Moving Between Windows in a View**. Windows can be moved inside a view using the arrow buttons to the top right of the view. The left and right arrows toggle through the windows available in a view, while the down arrow allows window selection from a list (see Figure 2.19).

Figure 2.16. Minimized window appearance at the bottom of the HIPE window.

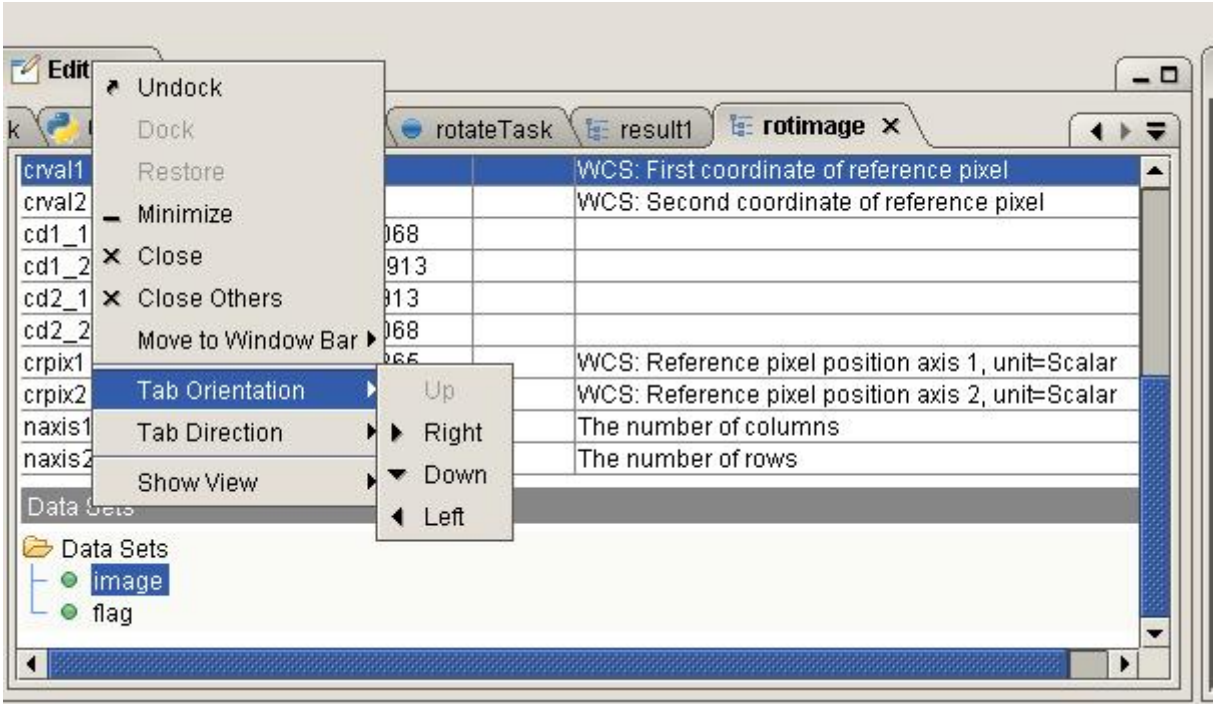


Figure 2.17. Changing tab positions in a HIPE view.

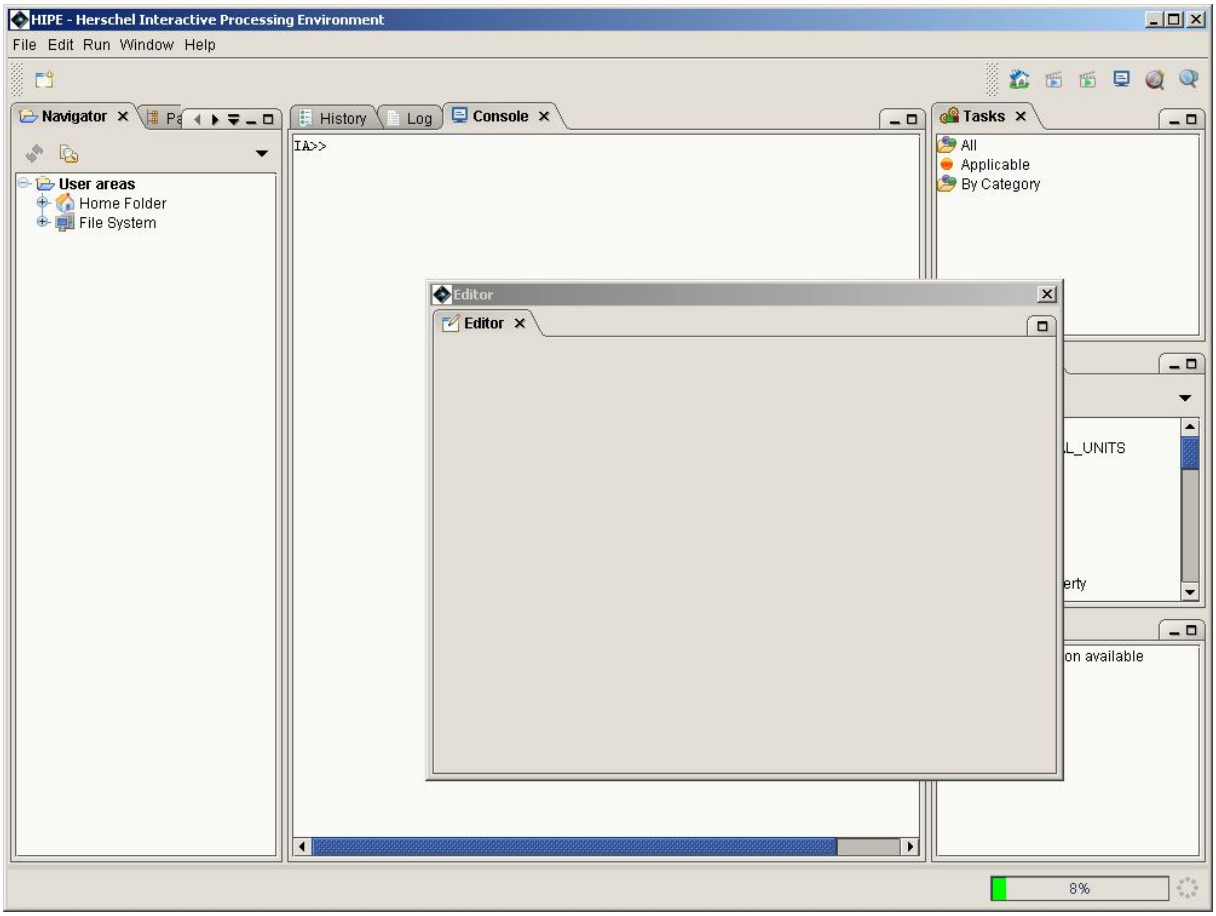


Figure 2.18. Example of undocking a view using HIPE. In this case the Editor view has been undocked and now sits "over" the HIPE worksurface and can be dragged to anywhere on the user's computer screen.

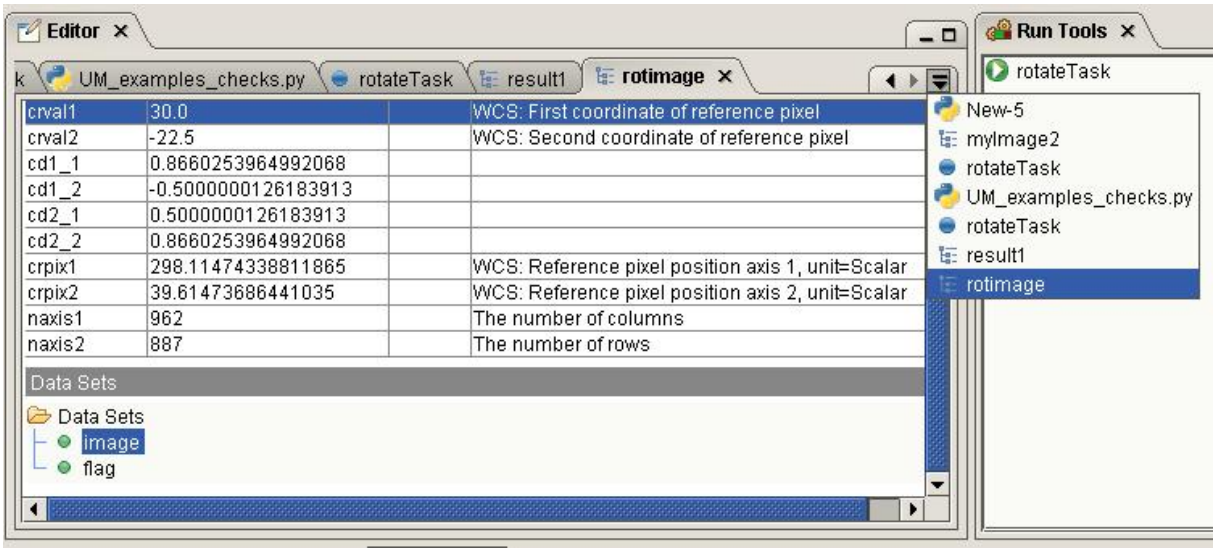


Figure 2.19. Selecting windows within a view. The down arrow shows the list of windows available in the Editor view that the user can move to.

2.7.2. Adding New Views to the Perspective

Several additional views can be added to a perspective. The complete list is obtained from the Windows menu on the toolbar at the top of HIPE. Pulldown to "Show Views" to show the available views in the system. Click on one to add that view to the current worksurface (see Figure 2.20).

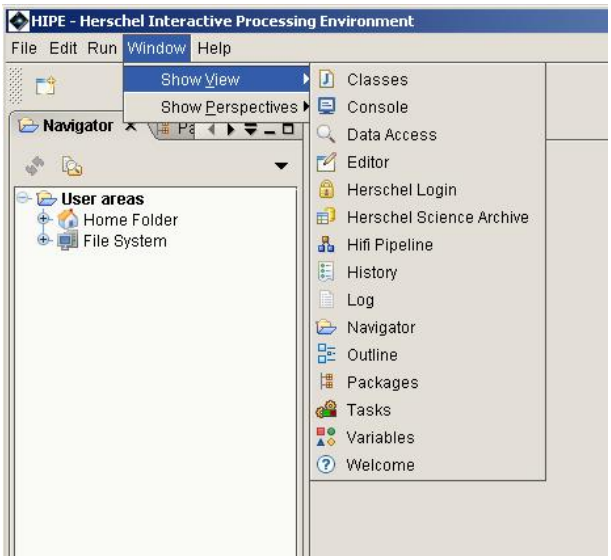


Figure 2.20. The 'Show Views' selection from the Windows pulldown menu lists the views that can be added (note: if the view already exists then a new one is not added).

2.8. Available Views And What They Allow You To Do

Each view has particular capabilities that can be combined to provide a powerful interactive environment. However, the environment can be simplified to a few windows to make a perspective, as noted above. The views available under the HIPE "Window" pulldown menu on the toolbar can be added to any perspective.

2.8.1. Classes

This view allows the user to see all the classes (routines) currently available in the session. These can include scripts written and loaded into the system by the user. Help information for any of the classes can be obtained by use of a right mouse button click. This brings up a small menu which provides access to Help.

Help information on a class appears in the "Topic Help" view.

Both these views are available as default "Workbench" perspectives (see Section 2.6.1).

2.8.2. Console

The Console view is also available in the default workbench window. It provides a terminal-like input for the DP system where command-line DP inputs can be made. A prompt (editable in a user's properties) is provided.

Re-running commands. It is possible to cut and paste command lines into the window. It is also possible to rerun commands by clicking on the window then hitting the up arrow key until the command that requires repeating is reached. Editing of the command line can then be done before hitting return again to rerun the (edited) command.

Note that the console inputs are the same as for the classic JIDE case and its full use is described in the *Basic User's Manual*. Outputs such as plots or images will appear as separate tabbed windows within the editor view (see below for more information on the Editor view).

The console window is also where printed output from routines appears. So a routine that involves a print output will provide that printed output to the Console view (see Figure 2.21).

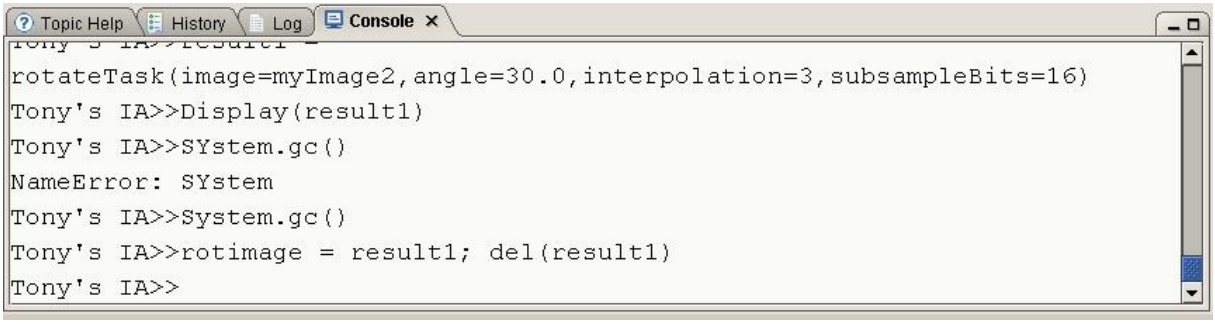


Figure 2.21. The Console view is where command-line input can be made and where feedback command-lines appear following the use of a GUI.

2.8.3. Data Access

This view brings up the interface for downloading data into a session (see Figure 2.22). This provides a mechanism for interacting with a set of data on a user's machine or data contained in remote databases, including the Herschel Science Archive (HSA). The data can be accessed by several means;

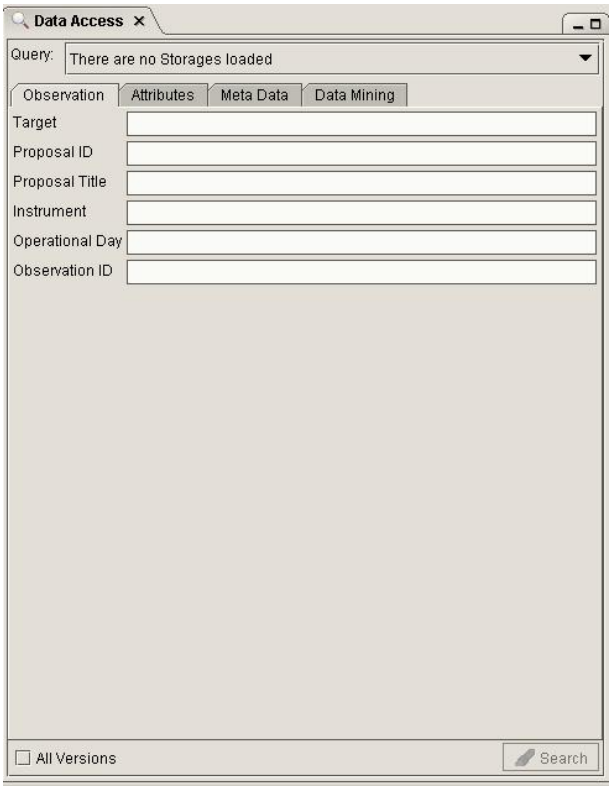


Figure 2.22. Outline of a variable in the DP session is shown in the Outline view.

- **Observation:** which allows querying for observations by target, proposal information, instrument or observation id/day of observation.
- **Attributes:** which allows data selection via attributes in the data products such as creation date and instrument model.
- **Meta Data:** which allows selection based on metadata associated with the products in the database (TBD).
- **Data Mining:** which allows selection based on information contained within the science data themselves (TBD).

2.8.4. Editor

The Editor view is where scripts are displayed and can be edited (see Navigator view information). When scripts are viewed a complete editing environment is made available allowing for standard editing capabilities. These are available under the "Edit" toolbar at the top of the HIPE work area (see Figure 2.23). A number of edit capabilities are also shown by icons () which can be clicked to perform certain tasks. Hovering the mouse over each of the icons indicates the edit function it performs, e.g. the torch icon provides find/replace edit capabilities and the double arrow icon allows the loading of everything showing in the current editor window. The run icon will run the highlighted lines shown in the currently shown edit window. If no lines are highlighted then the one line will be run from where the cursor is presently placed at the side of the script. An arrow appears marking this position when the mouse is clicked next to a line -- in left hand side grey region -- in a script window (see Figure 2.24).

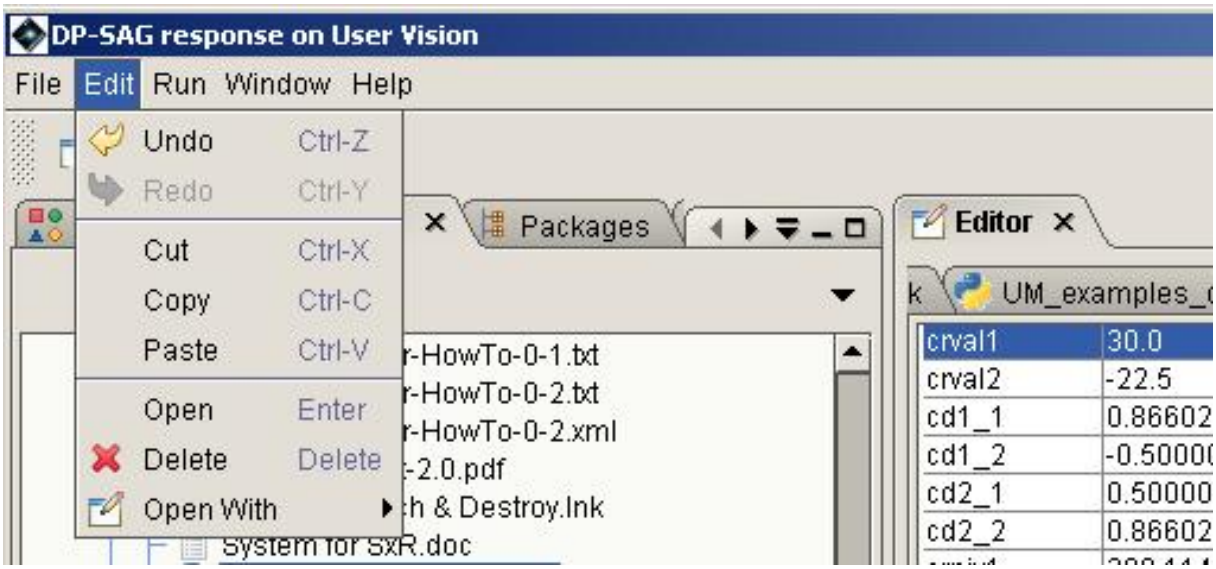


Figure 2.23. The Edit toolbar.

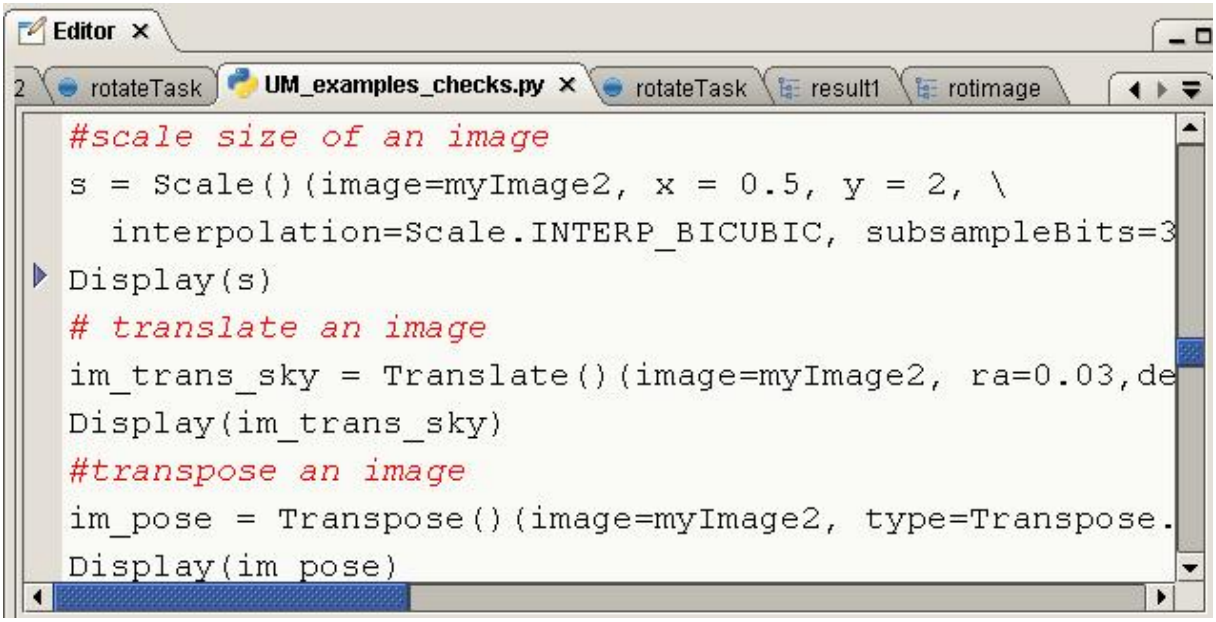


Figure 2.24. The Edit arrow is placed next to the line the user wishes to execute next. In this case, the Display task would be called once the Run button was clicked.

Once a script is initiated, it can be halted by clicking the red highlighted square (Stop) icon. NOTE: the current line of the script will be completed before the script stops running. This can lead to a delay before coming to a halt.

The Editor view is also where informational overview or the contents of a DP file type are displayed -- when requested. It is also the area where plots -- which are in themselves editable, e.g. zoom, pan, change of labeling, task dialogs etc. -- are placed. Examples are shown in Figure 2.25 and Figure 2.26.

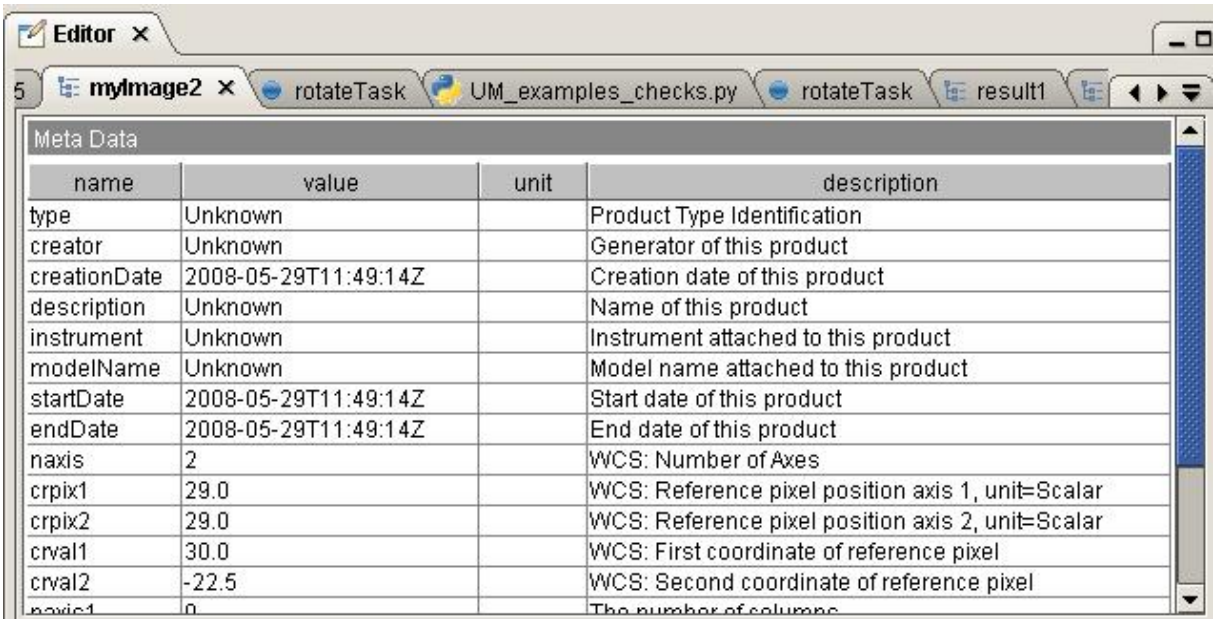


Figure 2.25. A window shows metadata associated with an image within the Editor view.

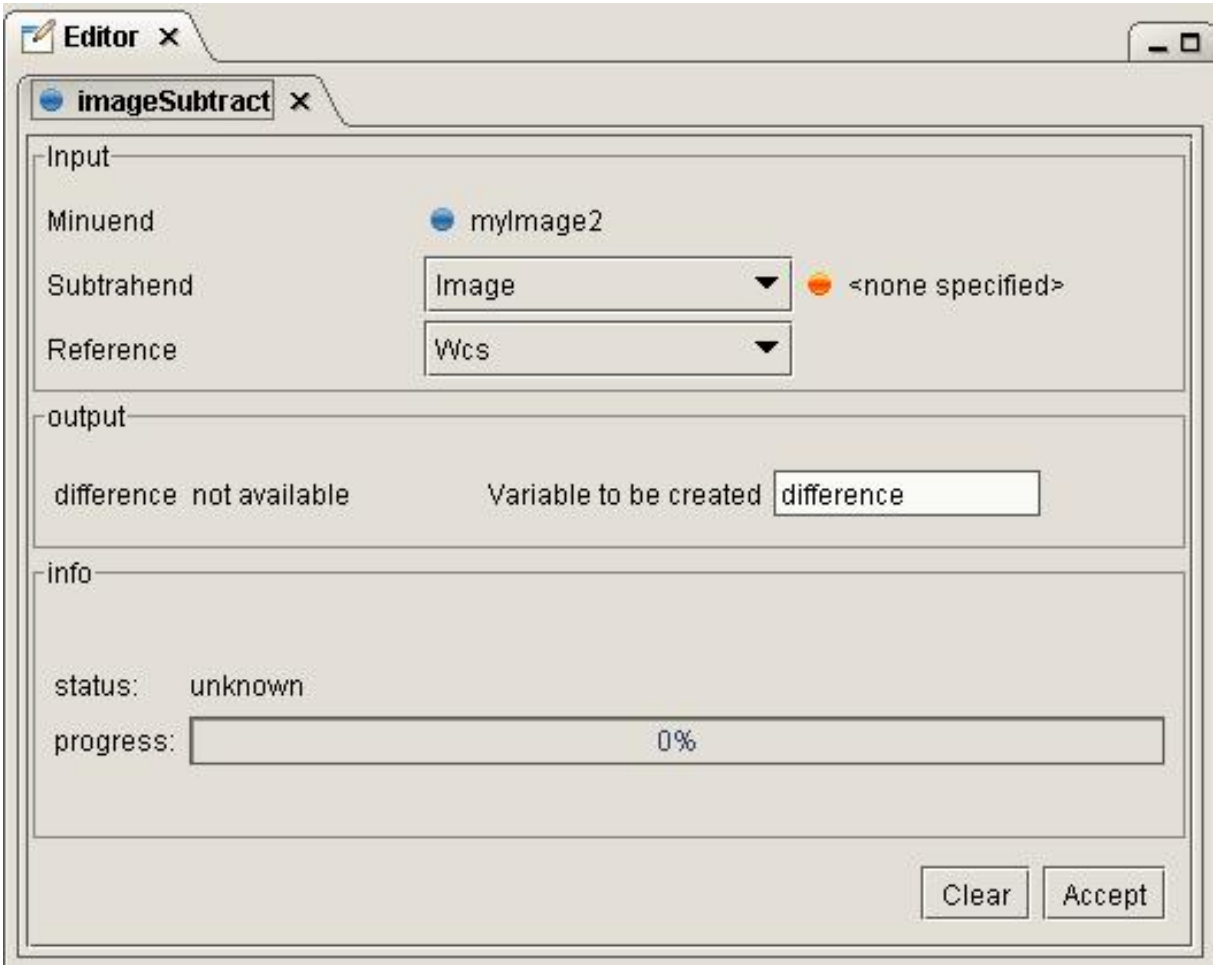


Figure 2.26. Window showing a task dialog associated with an image rotation within the Editor view.

The area can hold several (tabbed) windows so multiple plots/scripts/file contents can be open at one time.

2.8.5. Export Herschel Data from HIPE

This view allows Herschel data to be exported from the session into a directory structure which is identical to the one found in the tar file of observations that are received fromt he Herschel Science Archive following a request for data. See Chapter 8 for examples.

2.8.6. Herschel Login

This view allows the user to login to the Herschel Science Archive (HSA). It is also available as part of the Access Data perspective noted previously. See Figure 2.27. The user enters username and password which allows certain priviliged access to the archive system.

Figure 2.27. The Herschel Science Archive login screen provided by the Herschel Login view.

2.8.7. Herschel Science Archive

This view allows the user to access the Herschel Science Archive (HSA). It is also available as part of the Access Data perspective noted previously. See Figure 2.27. The user can get the HSA interface by clicking the "Open HSA User Interface" button. Once selection is done then the "Load Selected Products" button will bring selection into the HIPE session. More information is provided in the HowTo chapter on Data Access.

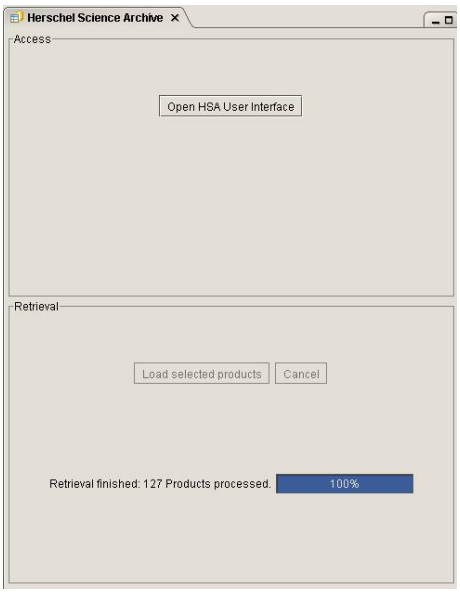


Figure 2.28. The Herschel Science Archive interface view.

2.8.8. HIFI pipeline

This has not been fully implemented yet, but will be a specific view from within which it will be possible to run HIFI pipelines (in part or full).

2.8.9. History

The History view provides a listing of the commands executed at the console or lines executed from the Jython script window of the Editor. This also shows whether the command was successful or not. A tick () indicates the command supplied was successfully executed. A white cross in a red circle () indicates that there was a problem when performing the command. A click on the small plus sign in a circle () next to this will expand out the error information including a complete traceback (see Figure 2.29).

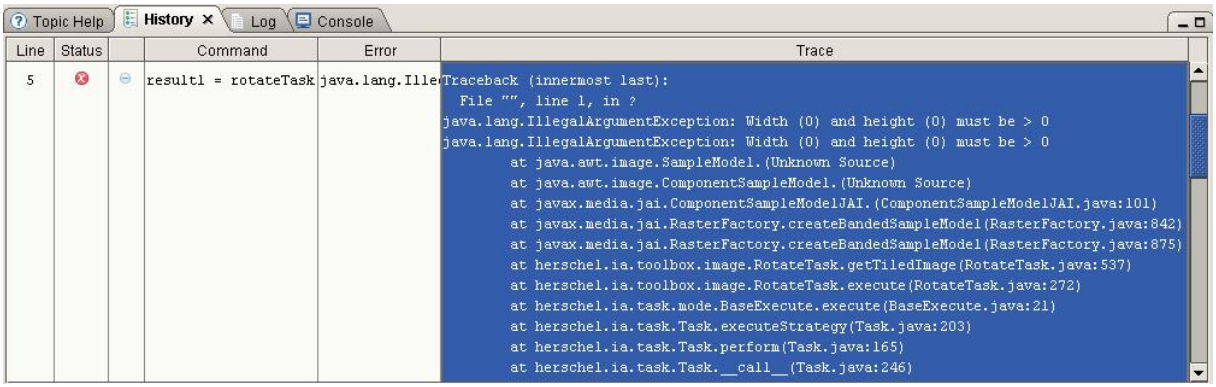


Figure 2.29. A traceback of errors is available from the History window.

History can be saved and used for later batch processing. A right click on the History window allows the commands listed to be either copied or saved to hard disk, enabling the contents to be used within scripts edited in the Editor window or storage of the command listing. Later sessions can then easily import the saved history into the Editor view for re-execution.

2.8.10. Import Herschel Data into HIPE

This view allows Herschel data to be imported from the tar file of observations that are received fromt he Herschel Science Archive following a request for data. This creates a context for the observation in HIPE that is easily navigable. See Chapter 8 for examples.

2.8.11. Log

The log screen provides a logging of the commands that have been executed from the command-line or the equivalent from dialog interactions in HIPE. It also indicates warnings generated in the system. The warning system level can be adjusted by the pulldown menu available at the arrow to top right of the window, from FINE to SEVERE warning levels (see Figure 2.30).

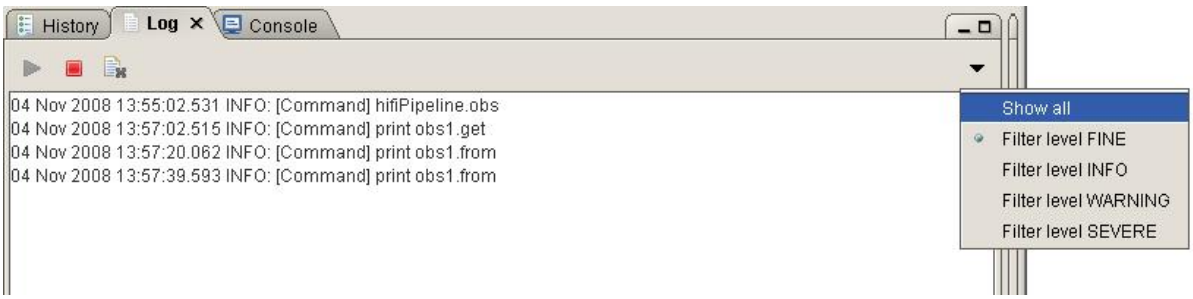


Figure 2.30. The Log screen with pull-down menu showing warning levels that are to be filtered and displayed in the Log view.

2.8.12. Navigator

The Navigator view provides access to the user's directory environment. By default it provides a listing of the user's home directory. Certain types of stored information can be brought into the session and displayed. A right click on an item in the Navigator list provides items indicating what may be done with the particular file (see Figure 2.31).

A prime example of using the Navigator tool is in loading a Jython script (file ending with .py). A right click and pull down to "Open With..." then "Jython Script Editor", will open the script up in an Editor view window (the Editor view can hold several, tabbed, windows). Scripts can also be run directly from the same menu, with the "Run Script" item appearing on the menu. Although the scripts need to be self-contained requiring no parameter inputs.

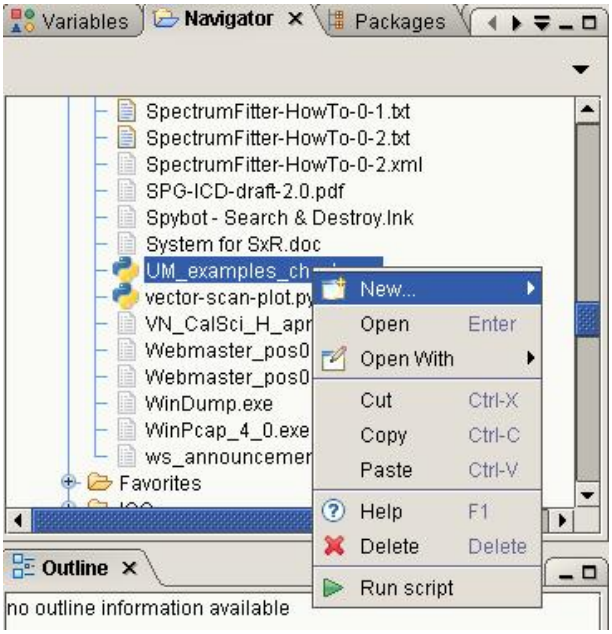


Figure 2.31. The Navigator view for HIPE showing the options available for the selected item on the user's system. Double-clicking on a ".py" script will open the script in a new Editor view window.

2.8.13. Outline

The outline information on a given variable is placed in this uneditable view. Clicking on the variable in the "Variables" view (see Section 2.8.18)provides an output of its name, variable type (class) and the herschel package in which this variable type is defined. In Figure 2.32, the DP session variable `myImage2` is shown to be an image dataset which could be viewed using the available DP Display task (for example).

Figure 2.32. Outline of a variable in the DP session is shown in the Outline view.

NOTE: This window provides information only, and its contents can not be manipulated by the user.

2.8.14. Packages

This view brings up a panel that provides access to the packages that are currently available to the session (see Figure 2.33. In order to get further information on what is available in a given package the user can double-click on one of the folders displayed. Package documentation associated with the available commands ( ) can be obtained by clicking on the command or right click on the item in the Package view and pulldown to "Help". Documentation appears in the "Topic Help" view. Note that the documentation provided at this level is not for the general user but more for those wishing to use to use package elements to develop scripts etc. within the HCSS.

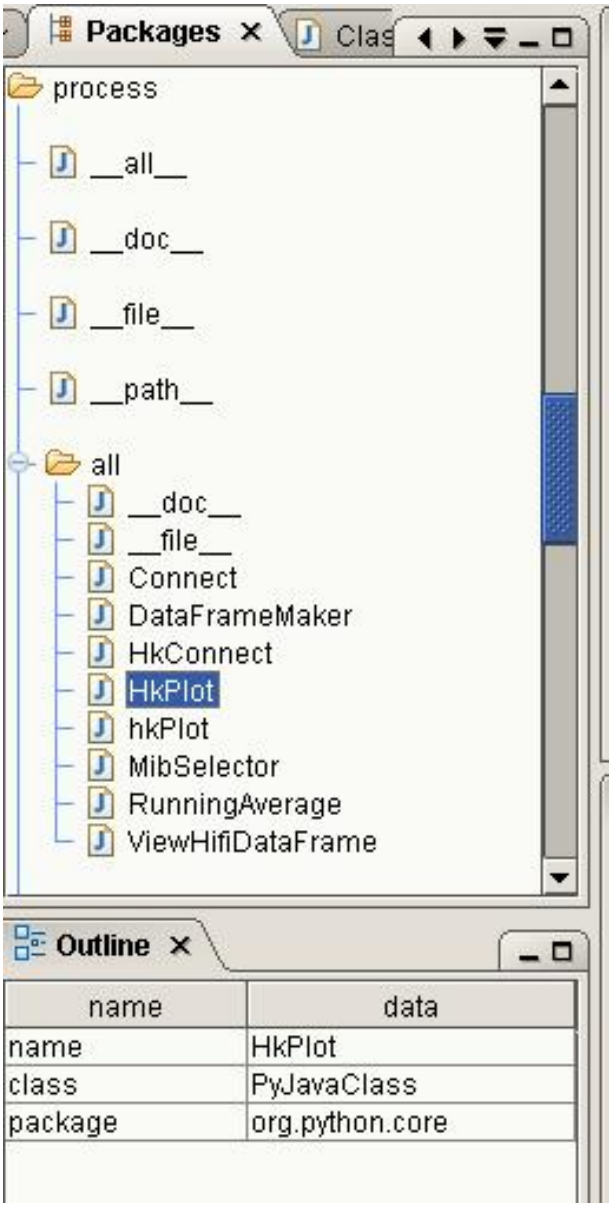


Figure 2.33. The Package view which is one of the tabbed views to the top left of the default Workbench perspective. Information on a package item is shown in the Outline view.

Information on the package item is provided in the Outline view when the item is highlighted.

2.8.15. PAL Storage Manager

This view allows the user to setup storage areas (mini databases storing data in the structures used to store Herschel observations. These areas can then be queried and read from and data can be stored in them. See Chapter 8 for examples.

2.8.16. Save Products to Storage

This view allows the user to save data (Herschel data products) from their session into a store of their choosing. See Chapter 8 for examples.

2.8.17. Tasks

This provides a list of tasks and tools available to the user from HIPE. These can be applied to variables of the appropriate type in a session. A right click on the available task allows a menu with a pulldown

that includes "Open With...". This allows a Task dialog where the task can be applied to a given set of data. Some example workflows are given below.

When a variable (see Variable view, below) is highlighted, available tasks appear as available in the "Tasks" listing. The Tasks are available in three folders; All, Applicable, and By Category. The "All" folder shows all available tasks in the system, while "Applicable" tasks are those that are designed and registered to run with data of the type associated with the highlighted variable in the Variables view. The folders can be opened or closed with a double-click. To start a task working on the data variable highlighted, simply double click on the task shown in the list (see Figure 2.34).

Figure 2.34. Tasks available for a given DP session variable are automatically made available in the "Tasks" view. Applicable tasks are shown in the Applicable Tasks folder.

Most of the tasks needed for basic data analysis can be accessed in this fashion. More information on how to use tasks for general data analysis is provided in the set of HowTos that are available in the main Help window (e.g., go to "Help" in the main toolbar at top left of HIPE, which opens up a window with access to the full user documentation).

Double-clicking any task in the Tasks view brings up a GUI dialog in the Editor view. This can be used to run the task in the appropriate way. In all cases an "Accept" button, to bottom right of the dialog, under the progress bar, should be clicked to run the task with the given inputs (see example task dialog at Figure 2.26).

2.8.18. Variables

This view shows the variables established in your session that you can use. You can always see what they are in two ways.

- Click on the variable in the Variables view. It's description and outline are shown in the Outline view (see Section 2.8.13).
- You can print the contents to the screen in the Console view (see Section 2.8.2) by the command

```
print <variable name>
```

Clicking on a variable in the Variables view enable you to see what type of variable it is (this appears in Outline view, Section 2.8.13). In this way it is possible to look at the structure of a complex item in your session containing multiple groups of spectra or images.

A right mouse click on a variable allows a short menu to appear which provides the possibility to do the following:

- Get help information on the variable type. The help information appears in a new browser window tab, and is the User Reference Manual information for the given variable type(see Section 2.4).
- Delete the variable from the session. Note that the equivalent command-line will appear in the Console view (see Section 2.8.2).
- "Open with" allows a list of ways to view the variable other than in outline (e.g., if it is a table you can use a Dataset Viewer, see Figure 2.35 or Spectrum Viewer for spectra). These viewers currently provide output in the Editor view (see Section 2.8.4).

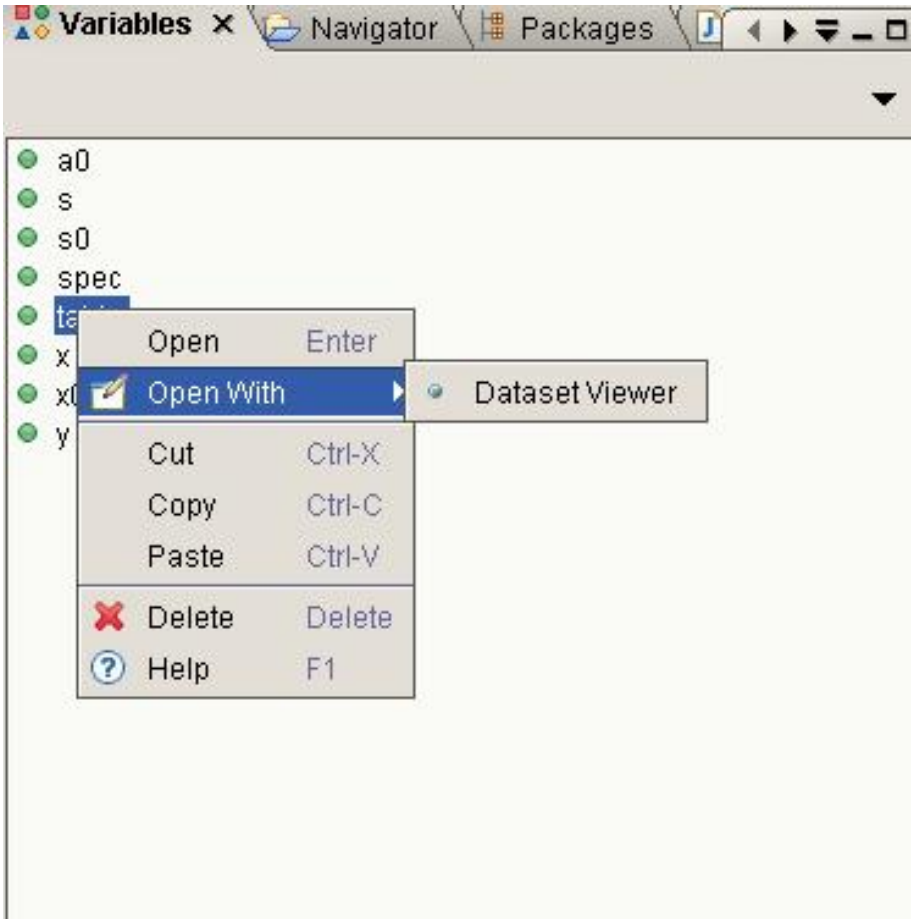


Figure 2.35. Tools available for a given DP session variable are automatically made available in the "Tasks" window.

2.8.19. Welcome

Opens up the initial startup window.

2.9. Viewers in HIPE

A convenient feature of HIPE is that recognizes the type of variables held in a session (is it a dataset, a spectrum, an image, a scalar constant etc). Items appearing in the "Variables" or "Outline" views,

with a green dot beside to their left, can potentially be opened. A right mouse click on any of these variables appearing in a DP session will provide a small menu of options allowing the user to "Open" the variable or "Open With..." or "Delete" or get help on ("Help Selection") the variable chosen. As previously noted in the "Variables" section.

Choosing "Open" allows opening with the first item in a list of available viewers for the selection. But there can be more than one viewer. These are shown under "Open With...". One of the viewers is chosen as the default for a double-click on the variable -- and this is shown with a dot beside it.

An example is shown for `SimpleImage`. A right-click on a variable of this type in the "Variables" window shows there are two viewers (see Figure 2.36). The Product viewer will show associated metadata and array values while the second viewer displays the image (more is provided in the HowTo on Display and Manipulation of Images).

As examples, viewers are available to show information on headers (metadata), and datasets (numerical arrays), enable table plotting and exploration, show images and/or spectra.

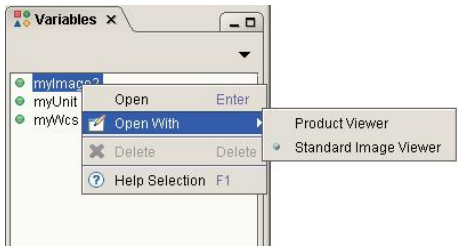


Figure 2.36. Available viewers are shown with a right-click.

Chapter 3. HowTo Retrieve Data from the Herschel Science Archive

3.1. Introduction

The Herschel Science Archive (HSA) is the main repository for the observational data products from Herschel. It is available via a web interface to the Herschel Science Centre, and directly from within a HIPE session.

The following sections explain how to access data and bring it into a HIPE session.

If you have a problem, please consult the FAQ to see if your issue is covered there: Section 7.

3.2. Retrieving Data from the Herschel Science Archive User Interface

Data can be accessed directly from the HSA using the Herschel Science Archive User Interface (HUI; see Figure 3.1)

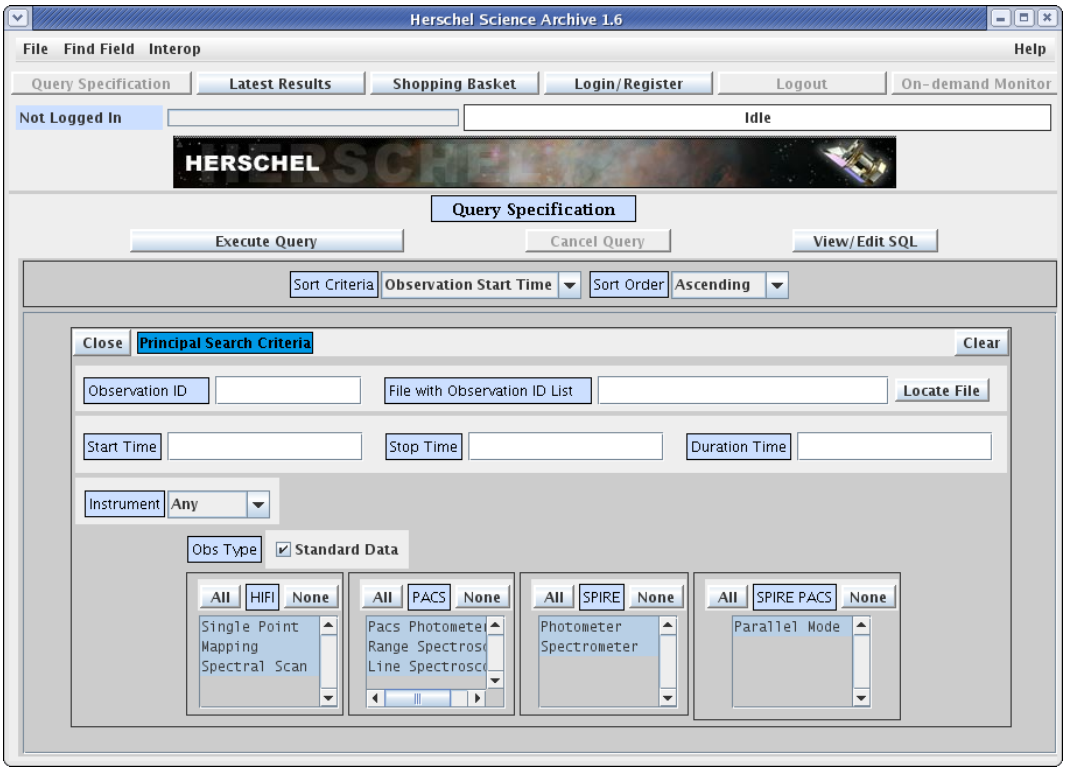


Figure 3.1. The Herschel Science Archive interface.

There are to ways of retrieving data from the HUI:

1. Directly downloading data items one by one

2. Via a *Shopping Basket* filled in with a number of observations

Both methods allow the same retrieval options:

- *Auxiliary* to retrieve only auxiliary files (pointing product, event logs, etc...)
- *Calibration* to retrieve only calibration files
- *Level 0* to retrieve only level 0 products
- *Level 0.5* to retrieve only level 0.5 products
- *Level 1* to retrieve only level 1 products
- *Level 2* to retrieve only level 2 products
- *ALL* to retrieve the whole "Observation Context" which includes all the files listed above
- *MULTIPLE* can be used to combine any of the above options in a single retrieval request

The option *ALL* should be selected for running pipeline scripts.

In order to retrieve data from the HSA you have to be a registered Herschel user. To register with the Herschel system please go to Herschel Archive Registration and follow the appropriate instructions. This registration page is also accessible through the *Login/Register* page of the HUI (*Register as New User*) Figure 3.2).

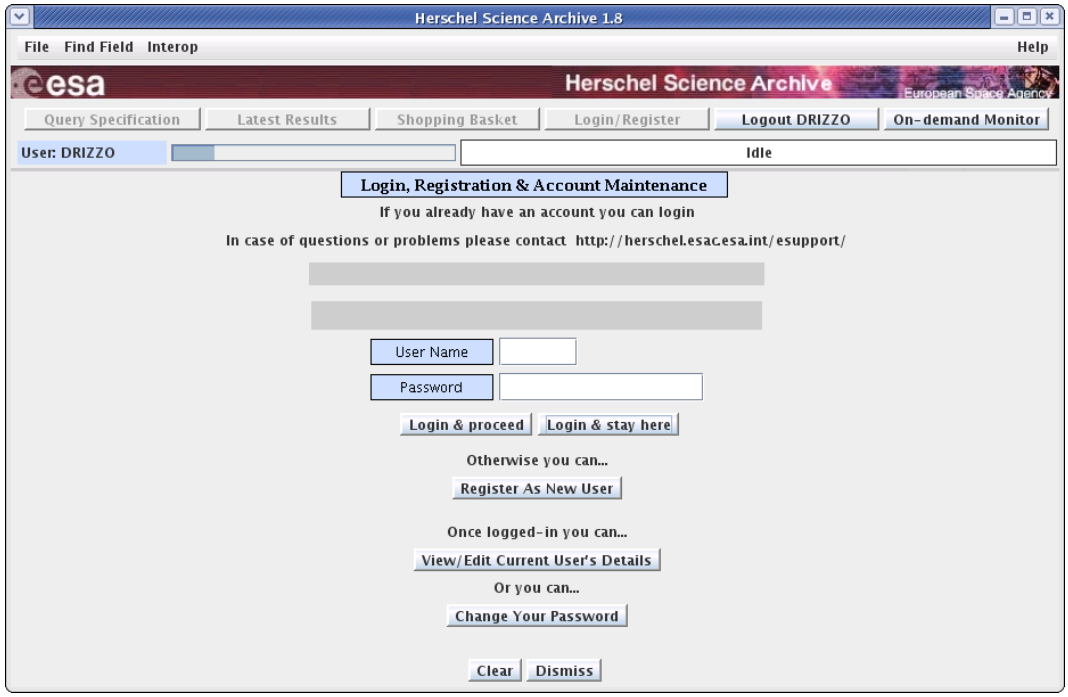


Figure 3.2. Login/registration in the HSA.

You do not need to register if you intend solely to browse the content of the archive. Registration is, however, a precondition to retrieve data from the HSA.

Only authorised users can access data covered by proprietary rights. The same rule applies to the viewable quick-look products of observations, as well as to proposal-related files. They can only be viewed by the observation owner, provided he or she has logged in with his/her registration identifier.

After executing a query in HUI, you are automatically moved to the *Latest Result* page (see Figure 3.3) which contains the list of observations matching the query.

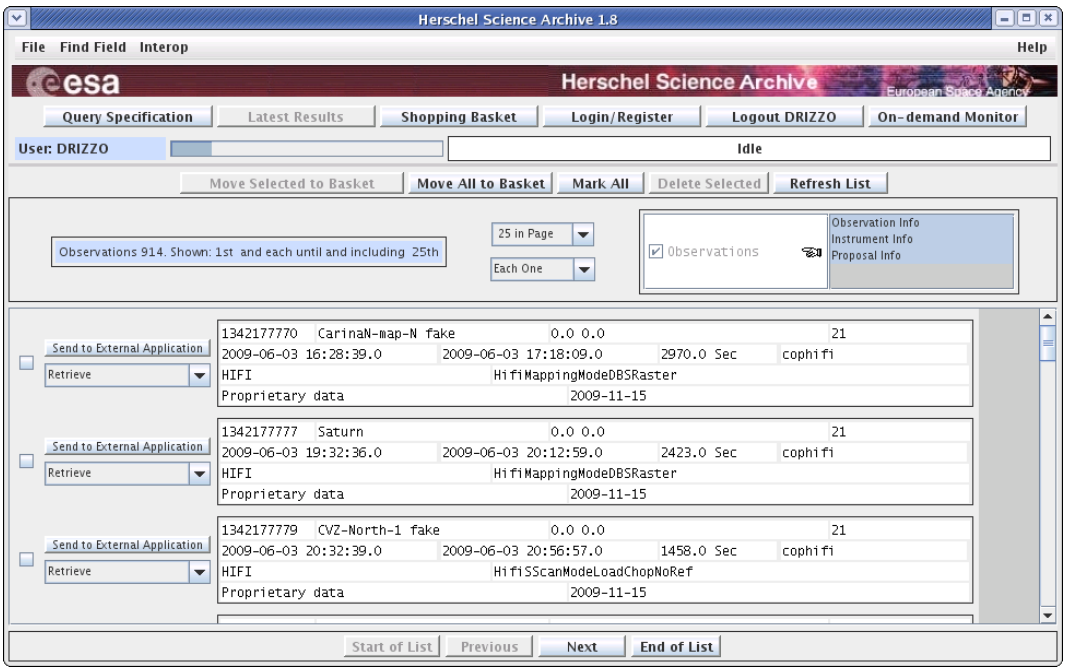


Figure 3.3. Result of query of the HSA

3.2.1. Direct data download

A direct download button (Retrieve) is available on this page next to each observation record. This button activates an FTP session, downloading a tar file with the data products corresponding to that observation.

The usage of this button is only recommended for individual observations.

3.2.2. Retrieving data through the shopping basket

With this method the records of several observations can be transferred into a *Shopping Basket*. This method is envisaged for mulit-observation requests. Once the shopping basket contains all the datasets to be retrieved, these can be transferred to a secure FTP area.

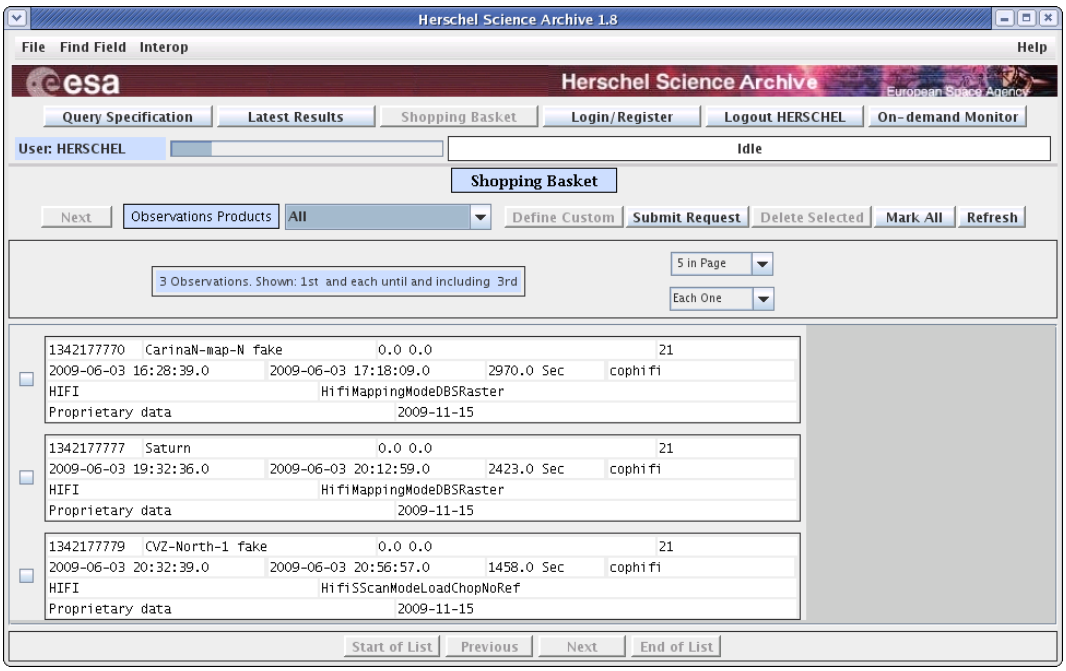


Figure 3.4. The shopping basket of data to retrieve from the HSA

To select data for retrieval, click on the check box to the left of each record of the observations list, and then click on *Move selected to basket* at the top of the panel. To see what is in your shopping basket, click the *Shopping basket* button. The observations moved shall now have disappeared from the *Latest Results* page and appeared in the *Shopping Basket* page.

To request *all* the observations that a query produced click on the *Move All to Basket* button. This will move all the observations into the Shopping Basket and it will also display the Shopping Basket page so that the request can be done straight away.

The shopping basket contains the instantaneous list of all the observation records, whose data are available for retrieval. Items can be deleted from the shopping basket by clicking on the check box to the left of them and then clicking the *Delete Selected* button to the top right.

Once the shopping basket has been filled with all the data you intended to retrieve, you are asked to finally confirm the choice, by submitting the data retrieval request. The generation of the dataset to be retrieved is then started (see Figure 3.5).

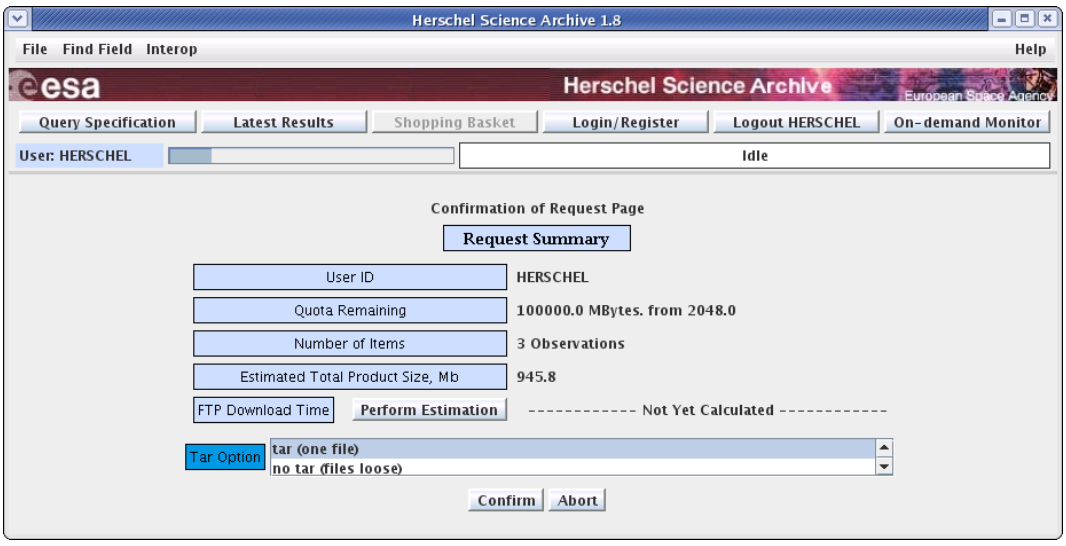


Figure 3.5. Data retrieval request in the HSA

The last step prompts the overlay of the Request summary panel. In this panel the number of requested observations, the size of the dataset and the remaining quota are reported. Return back to the Query specification or to the Latest Results page is possible in order to change the query or the shopping basket selection, respectively. At the moment only the tar option is working. The request should be confirmed with the button at the bottom of the panel and then a request ID is shown. This means that the generation of the dataset has started. An e-mail will be sent to the request originator as soon as the data are available on the FTP area for retrieval.

The tar file with the data retrieved from the HSA contains FITS files ordered in a well-specified (hierarchized) directory structure. Once the tar file is decompressed in a user directory, it can be registered in HIPE as a pool (see the next section for details).

Warning

If you use WinZip (and possibly other compression programs) to decompress your tar files, your FITS files may be corrupted. For more information on how to solve this problem, please see the section *Corrupted FITS file after unzipping* in the *User's Manual*: Section 10.4.3.2.

3.3. Accessing HSA Data within HIPE

There are two methods to get HSA data into HIPE:

1. Importing the HSA tar file into HIPE

2. Directly accessing the HUI from HIPE

3.3.1. Importing/exporting Herschel data to/from HIPE

The tar file provided by the Herschel Science Archive (HSA) can be registered in HIPE as a pool (see Chapter 8 and the *Advanced User's Manual* for information on local stores and pools) using the view *Import Herschel data to HIPE*. Select a directory in which the files coming from HSA are placed (see Figure 3.6). Pressing the button *Show Contents* all the observations included in that directory will be shown. Select the ones you want to save into a pool, select the pool and press *Import*. The observations saved into the pool are referenced automatically in HIPE.

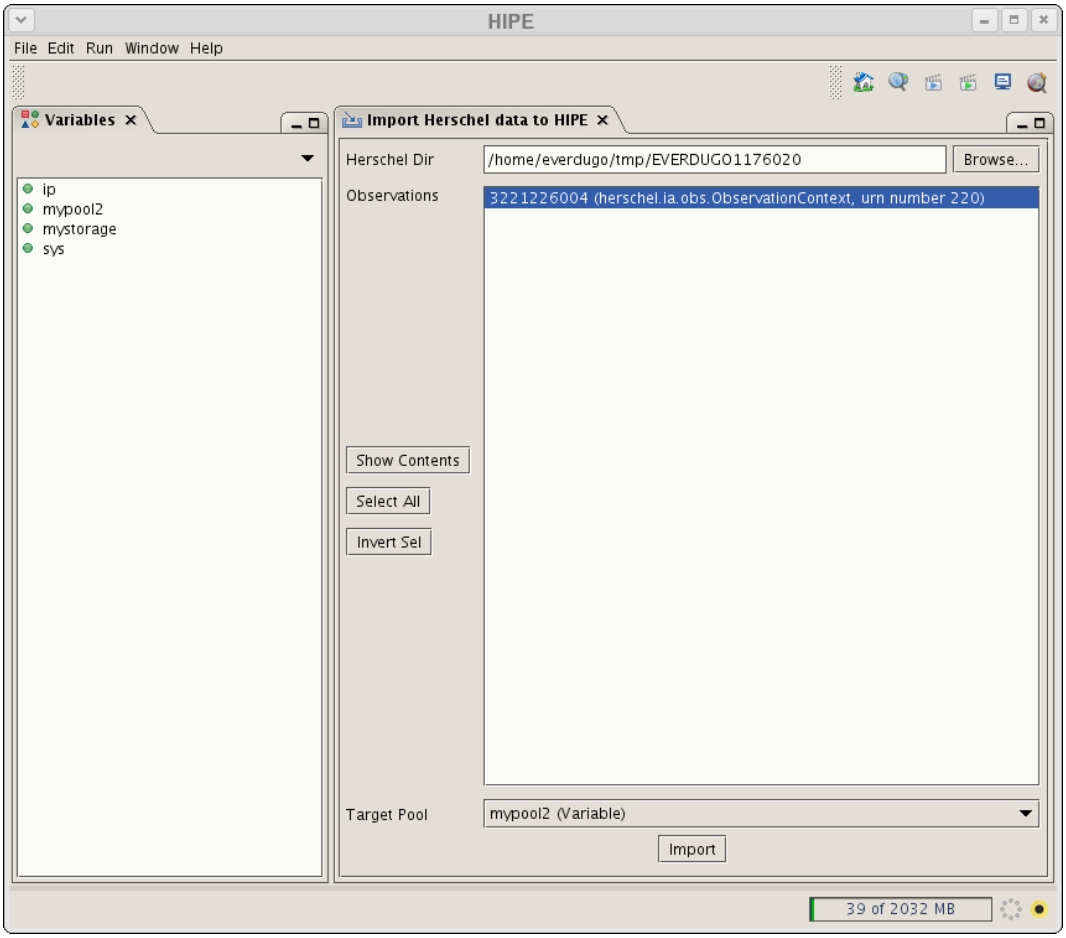


Figure 3.6. Product loading into HIPE from the HSA tar file.

This view provides the following:

- A way to navigate and find the directories extracted from the HSA tar files.
- A way to select the pool.
- A way to scan the observations (inside the pool).
- A way to select multiple observations for a bulk import-export.
- A button to generate a sequence of task execution requests: this also provides as a consequence, logging, tracking and *replayability*.

In the same way, pools can be exported to the standard (hierarchical) directory structure by using the *Export Herschel Data From HIPE* view. Select one observation from a pool and an output directory in the view, and press *Export* (see Figure 3.7).

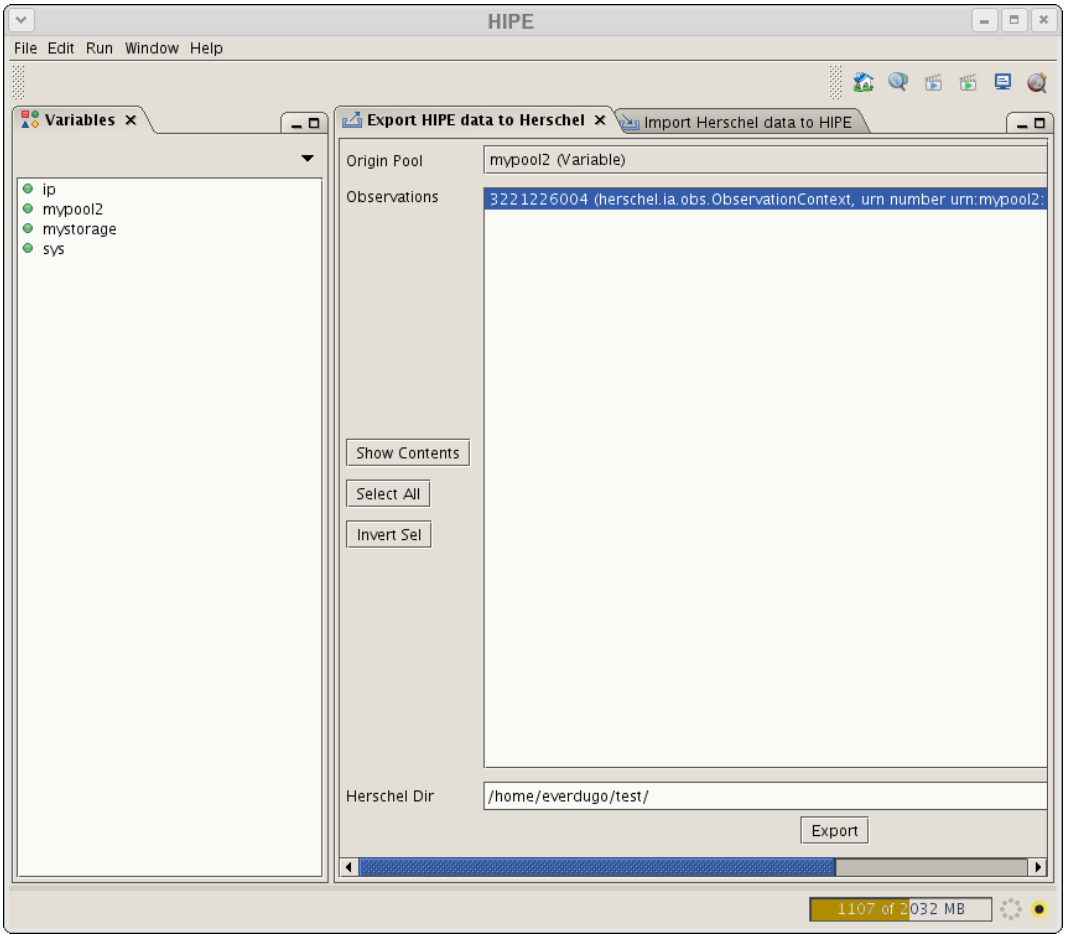


Figure 3.7. Product export from HIPE into standard Herschel directory structure.

Note

These views make use of the tasks `importUfDirToPal` and `exportPalToUfDir`, which are documented in the *User's Reference Manual*.

The XML file needed by these two tasks is included in the HSA tar file under a directory called `.exported/`.

3.3.2. Direct HUI access from HIPE

This method provides a fast way to visualise products without having to save them first.

To access the HUI from HIPE, simply click on the Herschel Science Archive icon on the *Data Access* page or select the *Herschel Science Archive* view via the Windows menu (see Figure 3.8).

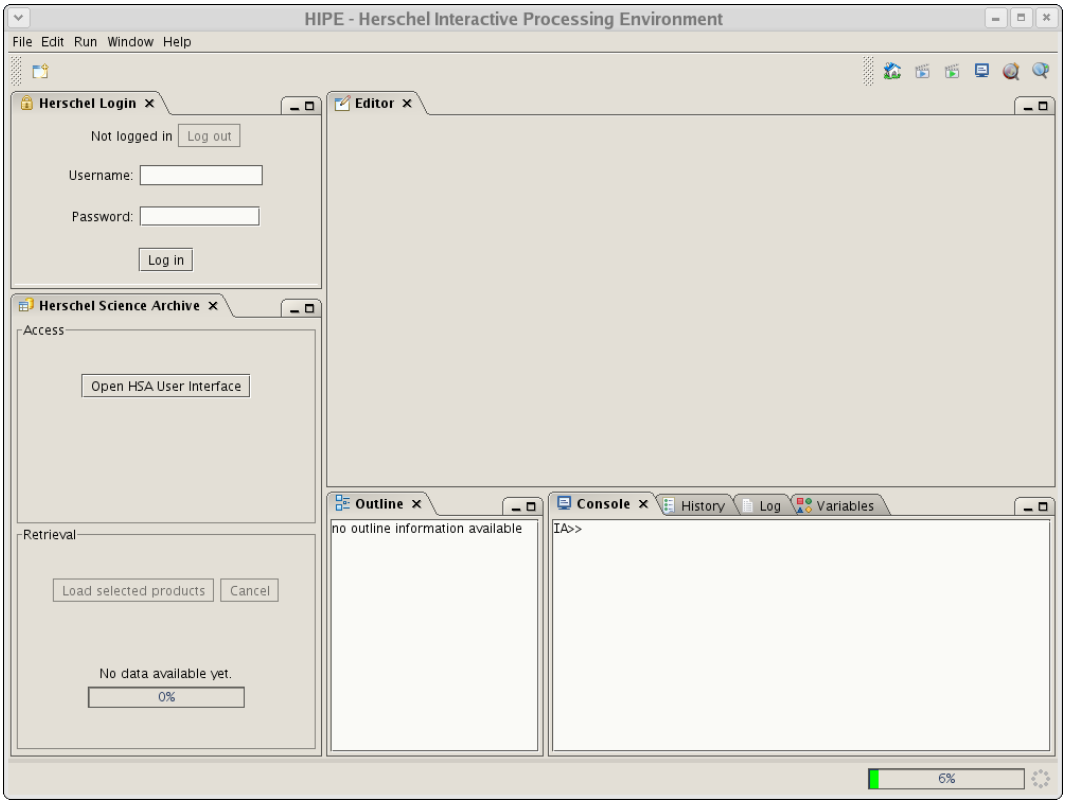


Figure 3.8. HIPE perspective for the HSA

Click on *Open HSA User Interface* to access data in the HSA. Note that if an HUI was opened before starting HIPE, opening a new one is not needed as the Plastic connection will be established automatically between them.

Warning

If you get a message about *Java WebStart* (`javaws`) not being present, it probably means you are using a 64-bit version of Java prior to 1.6 update 12 (1.6u12). To find out which version of Java you have installed, issue this command from a terminal window:

```
java --version
```

Java WebStart is a piece of software needed to fetch from the Internet the HSA User Interface. You can obtain it either by switching to a 32-bit version of Java or (recommended) by updating to Java 1.6u12 or newer.

Query the HSA for the data to be retrieved and press the button *Send to External Application* next to the observation records. Immediately the HIPE window will display a message indicating that there is one product ready for retrieval (as in Figure 3.9). Many observations can be sent to HIPE which will be accumulating the products ready to be retrieved.

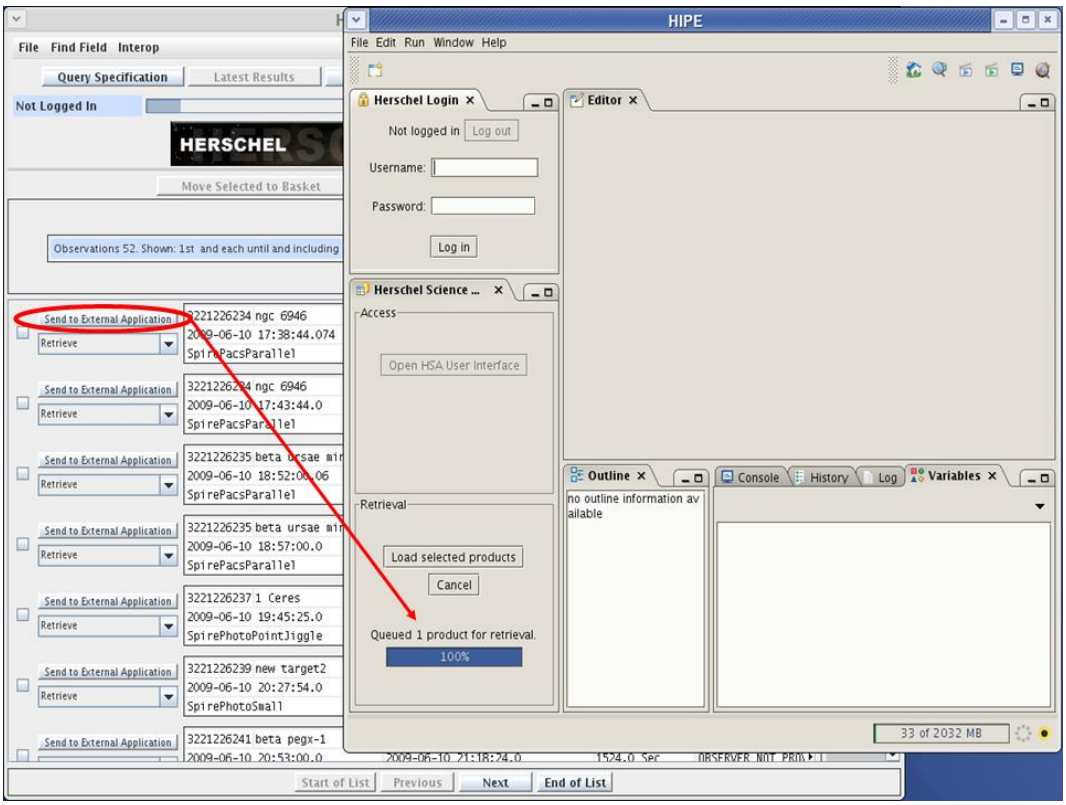


Figure 3.9. Retrieving observations from the HSA into a HIPE session.

Press *Load selected products* and the data will start to be loaded into HIPE (see Figure 3.10). During the operation an indicator shows in the HIPE display that loading is taking place and the system is busy.

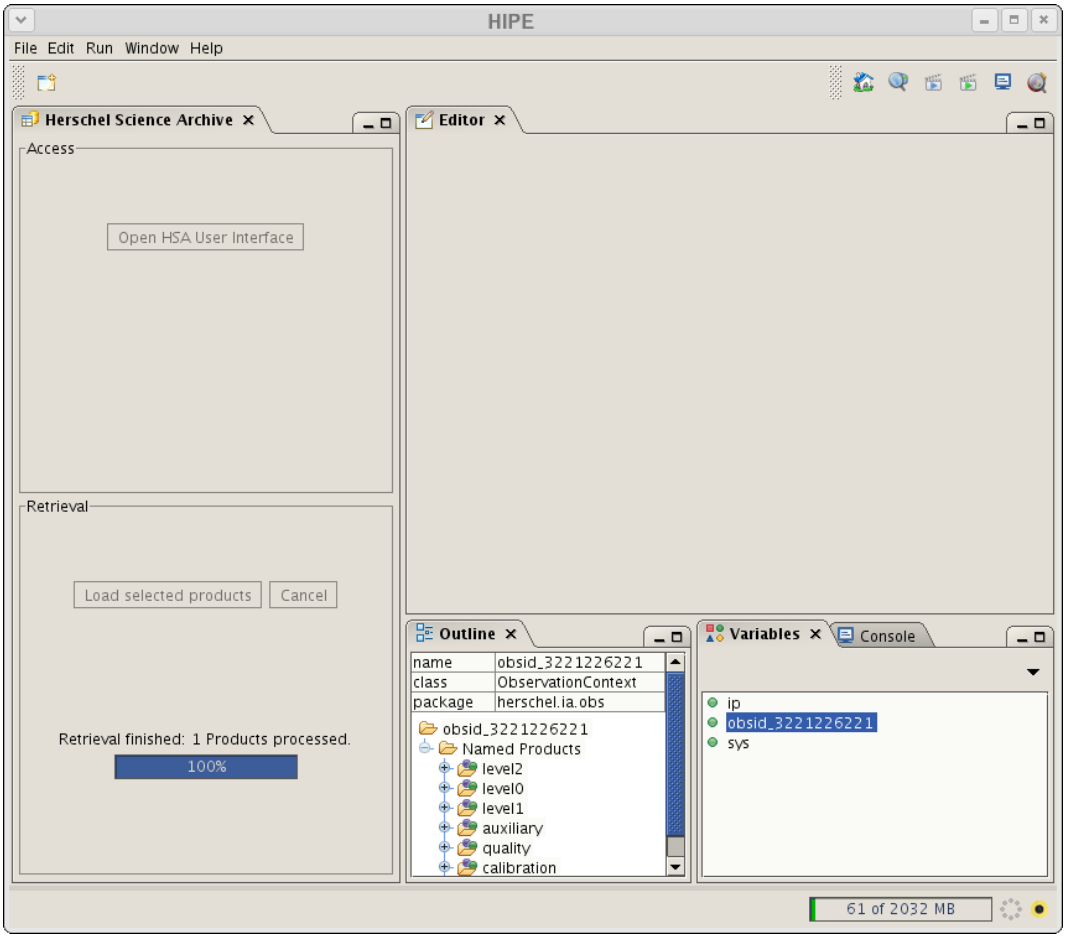


Figure 3.10. Product loaded into HIPE from the HSA.

Note that in this way the data is not stored on your machine, but it is referenced for fetching as needed within your working session. So this simply makes the data available in the HIPE session. Products can be inspected, analysed, plotted, and so on. Note also that for this, *the internet connection must be kept open*, since the products are being read from the HSA.

The products can be saved/stored into pools later on using the *PAL Storage Manager* and *Save Products to Storage* views (see Section 8.3).

Chapter 4. HowTo Store and Access Data

Herschel Editorial Board

4.1. Introduction

This HowTo chapter describes the means by which data can be accessed and stored and using the HIPE interface. It should be noted that reading and writing of FITS data is held in a separate HowTo.

In order to access remote areas of data storage users must be on the internet. For access to data in the HSA users must have an appropriate username and password. Full use of the HSA is discussed in the HowTo on Archive Access. Users of the HSA will be allowed access to their own data as well as publicly available data within the HSA.

Users can store their data locally in data product pools which can be accessed for reading through the Data Access area of HIPE. This means that the user's stored data (processed or unprocessed) can also be selectable by queries that can become quite sophisticated.

4.2. Creating and Saving Products in a Pool

Any product (an example being a complete observation in the form of an ObservationContext) can be placed in a pool, or storage area, on the hard disk of the user. The setting up of various types of storage is discussed in Chapter 12 of the "DP Basic User's Manual" available as part of the release documentation. For this HowTo we will simply illustrate how to set up a set of stores (which act a bit like mini databases) in which a user can place any output data that is in the form of a product, such as an observation.

A pool can be set up and populated in the following fashion via the command line.

```
store = ProductStorage() # -"store" is the name (handle) we will use
                         # in our session to place things
myPool = LocalStoreFactory.getStore("myTestPool") # -"myTestPool" is actually
                           # the directory name where the store will be
store.register(myPool)    # now link it/register it under the name of store
store.save(prod1)     # -....and now we merrily add our products called
store.save(prod2)     # -...."prod1", -"prod2" and -"prod3" into the store.
store.save(prod3)     #
```

Note that if you start a new HIPE session you will need to register your pool again via something similar to the first three lines.

The directory on your disk where the data physically resides in the directory ".hcss/lstore" which you will find under your login directory. You will see that the information is actually held as a hierarchical set of FITS files that can be treated like a database, allowing us to query and search for data in similar ways to other databases.

4.3. Registering and accessing other data stores

It is possible to register other stores that can then be searched from the data access view, but they first have to be registered in the system (you need to tell the system where they are, in effect). For data

stores elsewhere on your machine other than the default area this can be done by using the following lines of code which can be entered at the command line.

```
#import Configuration components into the environment
from herschel.share.util import Configuration
# get a local store (or create a new one if not already existing) with
# an id of -"test". The Configuration command changes the directory
# where the store is
Configuration.setProperty('hcss.ia.pal.pool.lstore.dir', -'C:\\.hcss\\myData')
datastore=LocalStoreFactory.getStore("test")
myStore=ProductStorage()  # tell the system it is a store of products
myStore.register(datastore) # register it
# -"myStore" is now one of the selectable data stores on the Data Access menu
myStore.save("myProduct")
# will save a Product in the DP session called -"myProduct" in the storage area
```

Note

The process of registering/adding pools that the user can use in a session is expected to be made into a simplified tool in the future.

4.4. Data access via the HIPE GUI

Other than interactions with the Herschel Science Archive (which are discussed in another HowTo), access to data is via the Data Access view. The Data Access view is available via two routes within HIPE. The first is via the data access icon on the Welcome page of HIPE (see Section 2.5. The second route is via the "Windows" pulldown on the HIPE menu. Go to "Show View" and pull down to "Data Access". This brings up the page shown in Section 2.8.3 described in the introductory section of the HowTo documentation.

4.4.1. Types of Stored Data

All data is stored in the form of `Products`. These products are kept as FITS files on the local computer system, but are organised into pools of Products/FITS files. This allows querying on the contents of the `Products` (e.g., the metadata or header information). The `Product` wraps information such as images, spectra or tables of data into a storable component. An example is a single Herschel observation (which actually has several products wrapped up into one).

When the user obtains data from the Data Access view it enters the DP session as a `Product` and an overview can be obtained using the "Product Viewer" via a right-click on the name of the product in the session (see information on viewers in the HIPE overview chapter). Datasets such as tables or spectra contained inside the products can be accessed and viewed using Dataset or Spectrum Viewers as described later in this HowTo.

4.4.2. Using the Data Access View

When selecting the Data Access view the user will have certain "pools" of data available. These allow access to data stored in registered data storage areas (basically areas accessible to the user on his/her own computer or via the internet to another computer). Storing data in user-named pools is described in Section 4.3. All pools currently need to be explicitly "registered" to tell the system where to look.

4.4.2.1. Using the Data Access View to Query for Products

There are several ways of searching through your stores of data to get the products you want. You can search for complete observations -- such as those you are PI on which exist in the Herschel Science Archive -- attriibutes or metadata values, or you can go into data mining which involves searches based on the data itself.

For all cases, setup of the data query can be done based on observation data, the attributes of data, meta data or all data (data mining). Once the query of the data store has been set up the search can

be done by clicking the Search button to the bottom right of the Data Access view. If the user wishes to access all available data in a data storage then this can be obtained by placing nothing in any of the input boxes of the query.

When the search button is clicked the equivalent command-line version of the request appears in the Console view (see Section 2.8.2). This can be saved and edited and used in batch mode processing. This helps to avoid syntax errors by the user in setting up queries on data stores.

Doing a Search

In order to do a search the user needs to do the following.

- Open the "Data Access" view.
- Select an available pool from the pull-down menu at the top of the view next to the word "Query". If none are available (greyed-out) then you need to first register a pool for access (see earlier sections of this chapter).
- After inserting an appropriate query, click on the "Accept" button to bottom right of the view. Note that if nothing is placed in the query then the total contents of the pool will be obtained. This is a good way to see the total contents of a pool.

Search by Observation

In this case we are dealing with high-level information. The data is part of certain proposal or uses a particular instrument on a particular day. Clicking on the "Observation" tab in the Data Access view allows searches at this level based on instrument, proposal ID, proposal name, observation ID (unique observation numbers or operational day (See Figure 4.1).

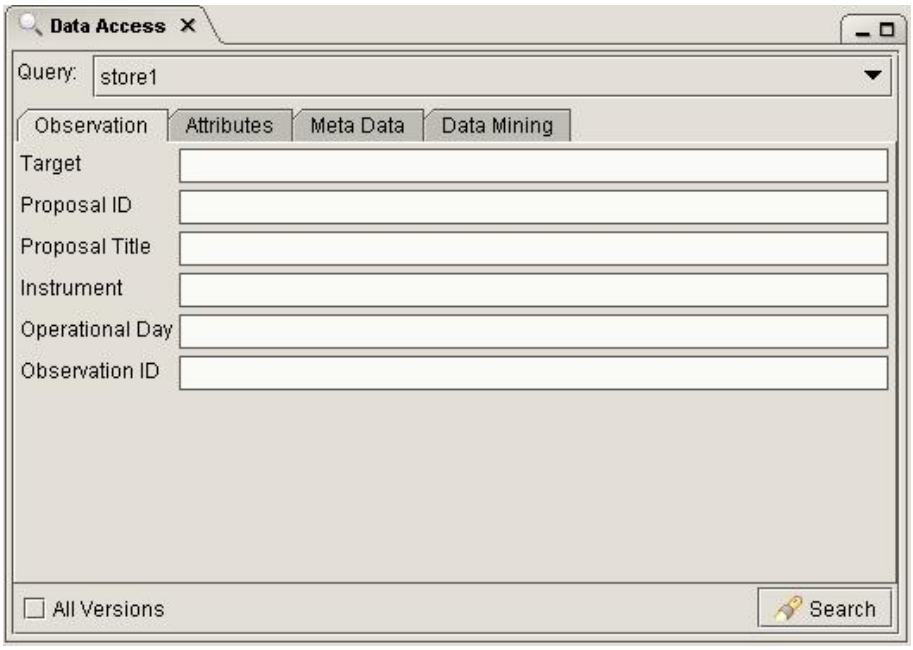


Figure 4.1. HIPE store selection and panel for searching by information on stored observation information in a product.

Search by Attributes

The attributes of a set of data are standard to all (See Figure 4.2) and it is possible to do a search on values in this given set of attributes -- which are listed in the query interface.

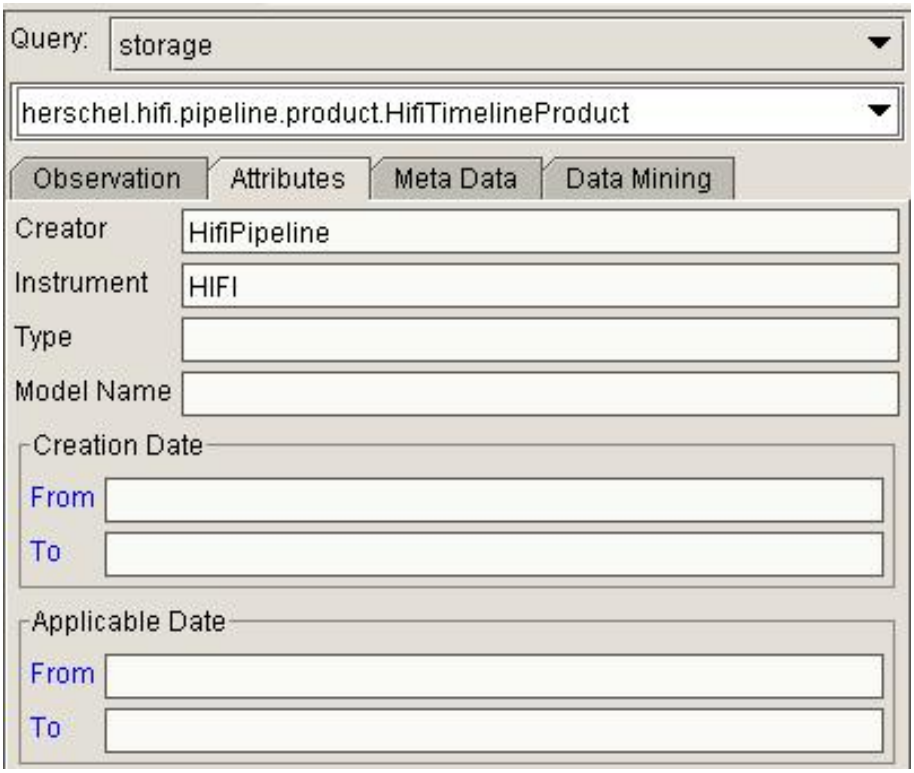


Figure 4.2. Attributes available for search.

Search by Meta Data

Meta data (like FITS header data) is data more specific to a given observation (See Figure 4.3).

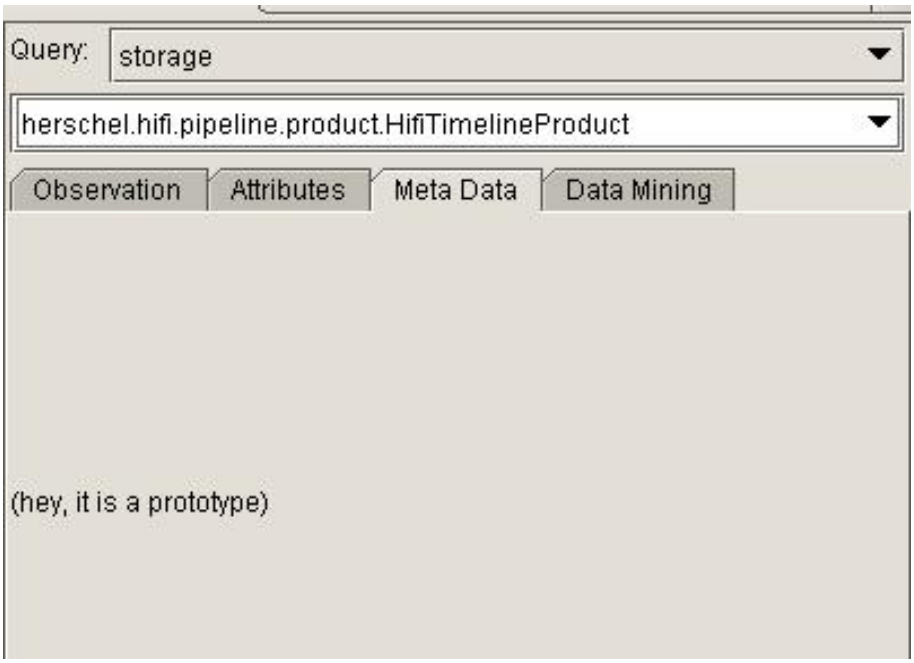


Figure 4.3. Metadata search.

Search by Data Mining

For data mining it possible to search on specific information contained within the science data itself rather than the meta data. YET TO BE IMPLEMENTED (See Figure 4.4).

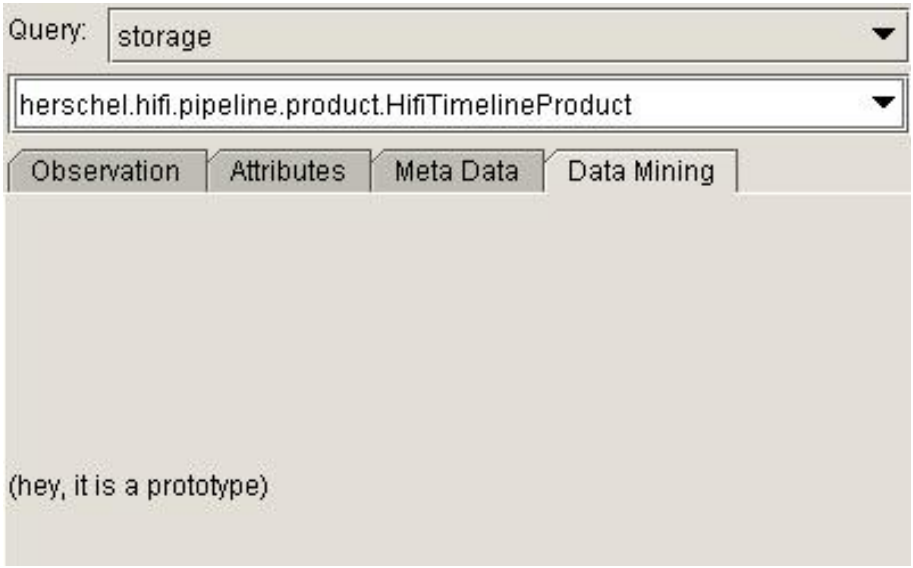


Figure 4.4. Search via data mining.

4.4.2.2. Output from a Query and Searching a Query Result

The output from the first query produces a result "QUERY_RESULT". This will be a group of products (e.g., observations) which can then be looked at by the user. The "QUERY_RESULT" name is highlighted in the Variables view (where the name can also be edited to something more appropriate if desired). This result is also automatically fed back to the Data Access pulldown menu, allowing for a search to be made on the result of the initial search.

The query output can be viewed by double-clicking on the result variable, e.g. "QUERY_RESULT" in the Variables view. This brings up the query results viewer in the Editor view part of HIPE. This lists the selected items. It also makes the outline available in the "Outline" view.

Clicking on one of the results shown in the query viewer extracts the chosen result (for example, the first product in the list is then available as "prod_0" in the session). Clicking on the name of this extracted product when it appears in the "Variables" view allows further assessment of its contents and viewing of any datasets it contains.

4.4.2.3. An Example of Search to Display of Data

In this case, we have partially processed some HIFI data to level 1, which has the format of a HifiTimelineProduct, and stored several versions of this processing in a store given the handle under the HCSS of "store1". This appears under the Data Access view pulldown menu as a selectable store item. The following now leads to displaying some data that has been extracted from our data store.

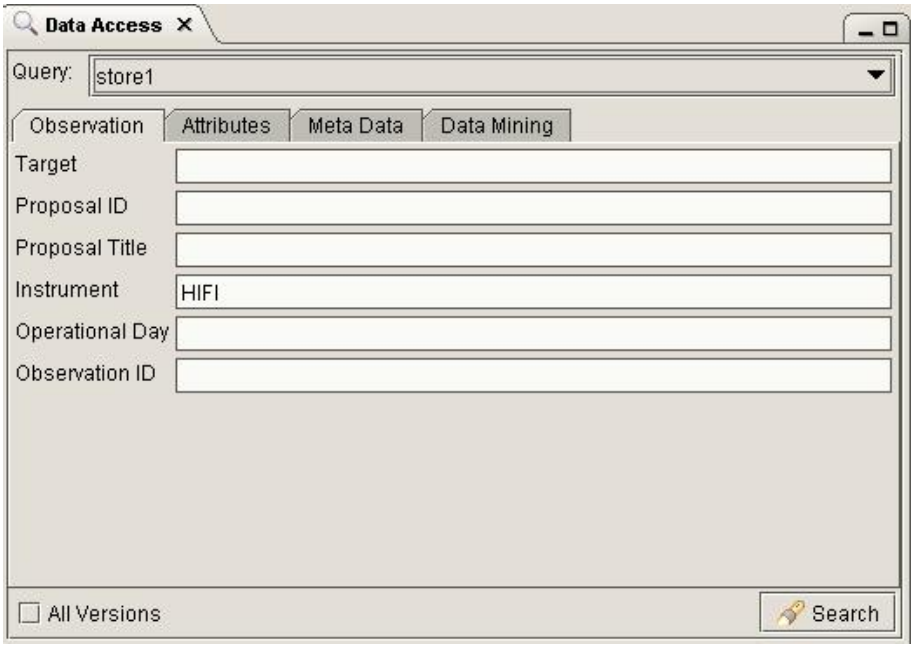


Figure 4.5. Set up of a query for data out of our store.

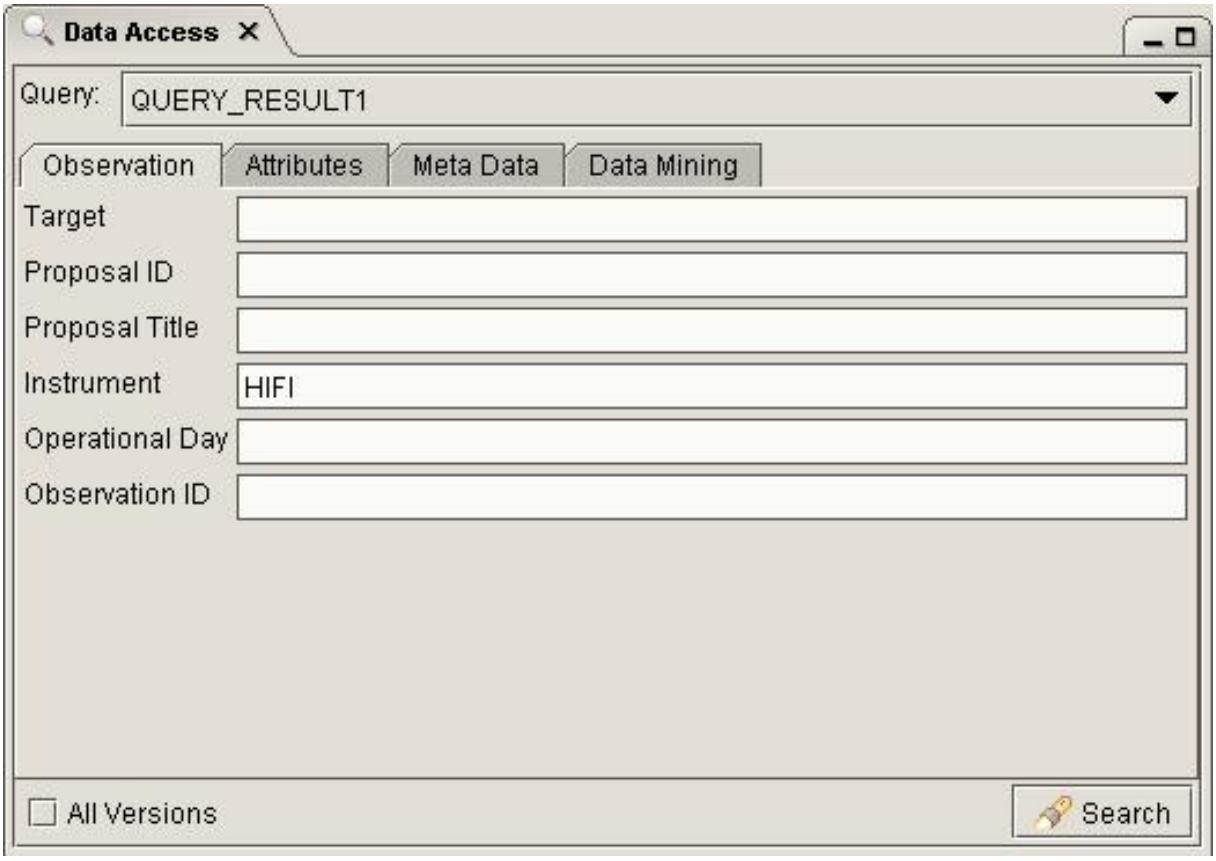


Figure 4.6. Query result obtained.

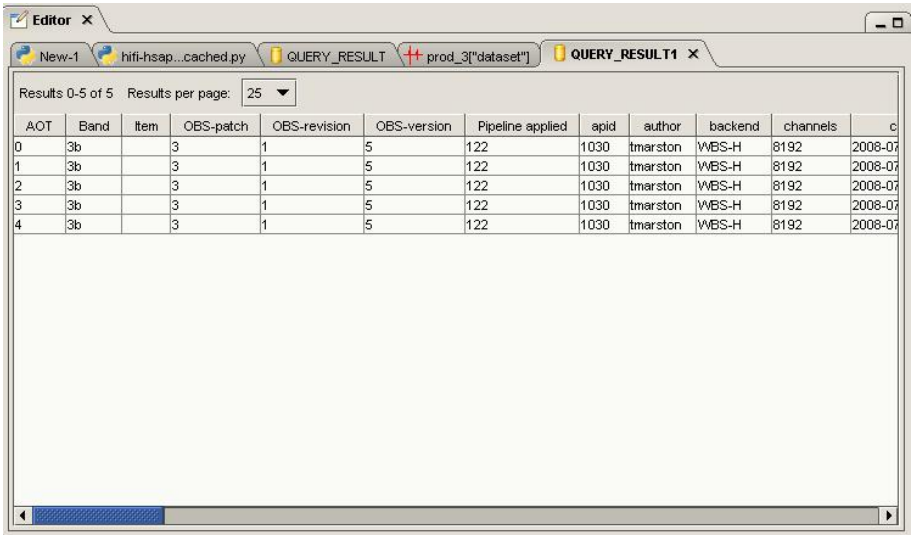


Figure 4.7. List of query results appear in editor window.

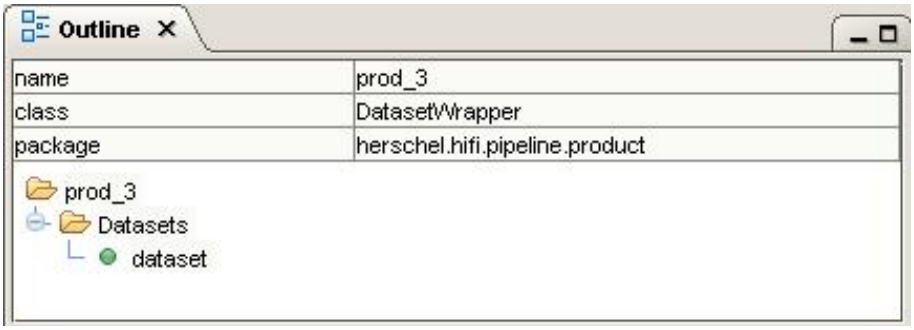


Figure 4.8. One of the items is selected with outline of contents shown bottom left.

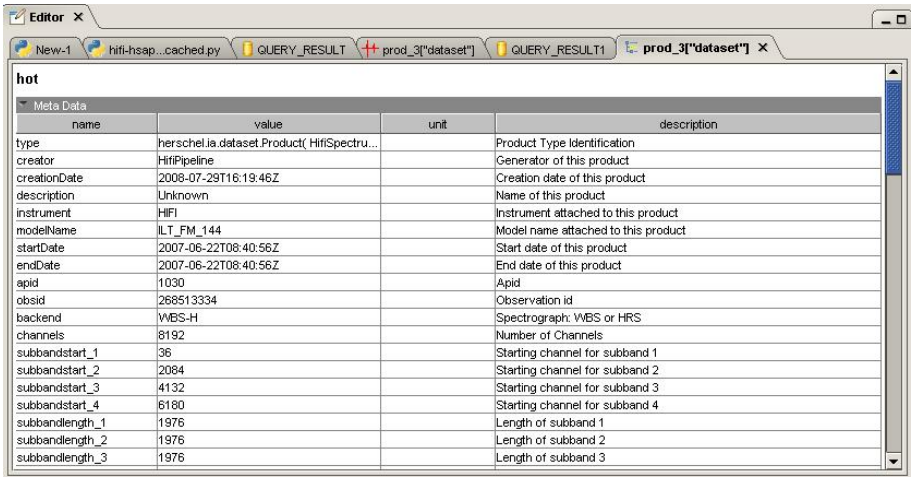


Figure 4.9. Metadata (header) display for the extracted spectrum.

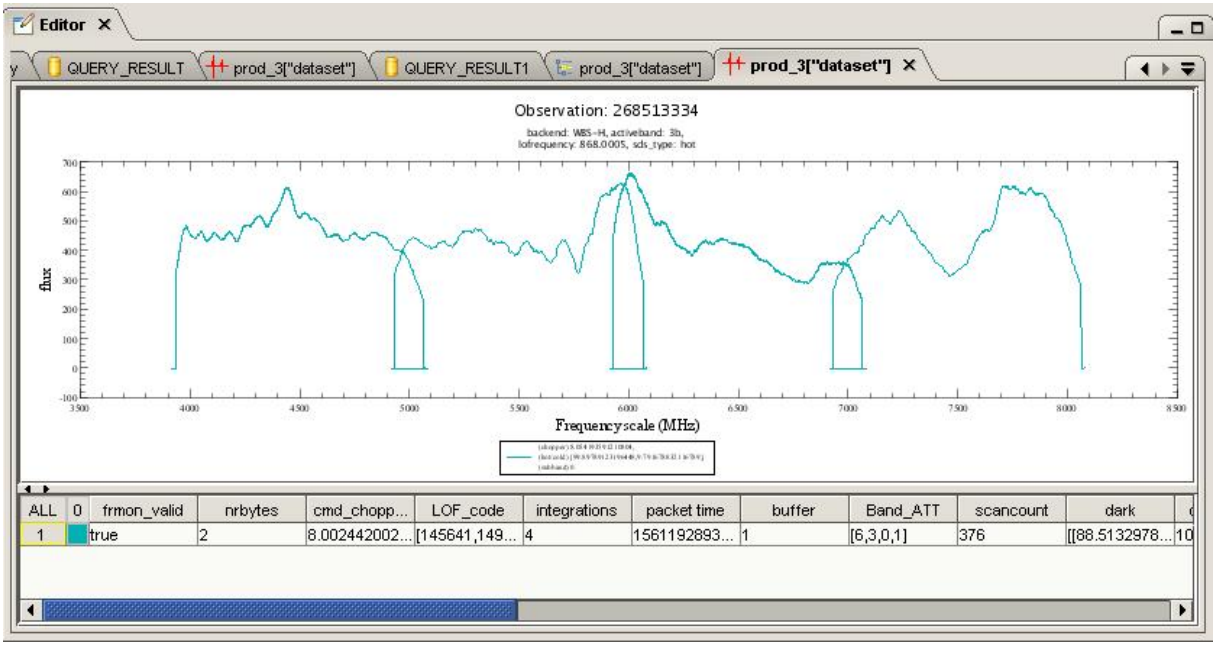


Figure 4.10. Displaying the extracted spectrum. Note that the view has been expanded using the capabilities of the "Spectrum Explorer" viewer.

1. We now intend to search for all HifiTimelineProducts (see second pulldown menu on the screen) with instrument=HIFI within this store by searching on these attributes. The setup should look like the screen shown in Figure 4.5.

2. Once this has been setup we click the "Search" button and the appropriate results are extracted and placed in a query result (see Figure 4.6). A highlighted "QUERY_RESULT1" (the number automatically placed at the end will increase depending on the number of queries you make) appears and the data access store available for querying -- at the top of the Data Access view -- immediately changes to QUERY_RESULT1 ready for further searching on the initial query results.

3. Select the query result in the Variable view (QUERY_RESULT1) via a double mouseclick. This provides a Query result viewer showing a listing in the Editor view of the query results items (see Figure 4.7).

4. Double-clicking on one of the results shown in the editor view creates the item (product) in the session. It allows us to pull out one of the selected products (e.g., "prod_3" for item number 3 in the query viewer) which can be manipulated in standard ways. For example, if we click on this product in the Variables view we get an outline of its contents in the Outline view (as in Figure 4.8).

5. We see that it shows a single folder in the Outline view. Clicking on the first folder, it opens up to show its contents which include a single dataset (as in Figure 4.8).

6. A right-click on the word "dataset" in the Outline view provides a set of viewer options. The Dataset viewer will show the associated metadata (header) information plus a table of various values associated with the spectrum, include flux/count values per channel (as in Figure 4.9).

7. Alternately, we can simply view the extracted spectrum dataset by selecting the "Spectrum Explorer" viewer instead (see Figure 4.10 and HowTo on displaying spectra).

4.5. Data Access via the Console View Command Line

Within the Console View (see Section 2.8.2) it is also possible to access data directly from the command line. The commands for doing this are actually generated in the Console view when using the dialog interactions noted above.

In the following we show how the information can be extracted into "newVariable1" (as per above example) which is then ready for display, fitting etc.

1. First we do a query on attributes in our data store, which was labeled "storage", looking for HifiTimelineProducts and instrument=HIFI. The following should all be on one line and the easiest way to get it is when it is copied to the Console view when using the Data Access view as noted above.

   ```
   QUERY_RESULT1 = StorageResult( store1,
    \
   herschel.ia.pal.query.MetaQuery(herschel.hifi.pipeline.product.DatasetWrapper,
    \"p", -"(p.meta.containsKey(\"instrument\")
    \ and p.meta[\"instrument\"].value == \"HIFI\")") -)
   ```

   We can always print to the screen the contents using

   ```
   print QUERY_RESULT1
   ```

   Queries on the observation or meta data windows are MetaQuery's rather than AttribQuery's.

2. Now we extract the third in the list of results found.

   ```
   prod_3 = QUERY_RESULT1[2].product
   ```

   Again, we can use the print command to see its contents. Which is actually several products (in this case 5).

3. Get the dataset out of the product.

   ```
   data_3 = prod_3["dataset"]
   ```

   The Outline view of data_3 will show it is of the form WbsSpectrumDataset and that it is a HIFI pipeline product.

   A full explanation of how to handle displays and manipulations (arithmetic and fitting) of spectrum (and image) datasets are covered in other HowTos.

When accessing Query results from the command line, it is harder to know which U RN in the list of references is the one we require. To aid us choose the correct one, we can sort the Query result by metadata. For example, to sort by creation time and put the reference to the oldest product first:

```
MetaComparator.sort(QUERY_RESULT1, ["creationDate"])
```

4.6. Interoperating with the Virtual Observatory

The Virtual Observatory is a community-based initiative. A number of national and international projects are organized in the International Virtual Observatory Alliance, whose mission it is to *"facilitate the international coordination and collaboration necessary for the development and deployment of the tools, systems and organizational structures necessary to enable the international utilization of astronomical archives as an integrated and interoperating virtual observatory"*. This initiative has led to the definition and implementation of technologies on different topics: Integration of applications (such as HIPE and spectral analysis tools), integration of general data analysis tools and archives of different missions, and more.

The most relevant technology at this point is the integration of applications. In practical terms, the integration means being able to view and manipulate data in one application, send it to another

application with the click of a button, view and manipulate the data there, and send it back to the original application. The technology currently used for this interaction is *Plastic*, to be replaced in 2009 by *SAMP*. Plastic and SAMP are very similar, but Plastic was intended as a prototype and SAMP consolidates the protocol, resolving various minor issues.

Plastic and SAMP work using a so-called message hub: An independent, very light-weight application (the hub) is started on the user's desktop and all applications interested in communicating register on the hub. The communication works by sending a message to the hub, which will deliver it to the intended target application (broadcasting to all applications is also an option). The message can contain the data that is sent, but generally the data will be written to a temporary file, and the message is used to pass the location of the temporary file, plus additional information, such as the units of the data.

All ESA archives are VO-aware already, but access to VO-aware archives in HIPE is not available yet, it is planned for the future. Aladin provides an interface already to many data sources (such as the ESA archives). So it is possible to access the ESA archives by retrieving the data using Aladin and sending it to HIPE from there.

4.6.1. Getting practical: Sending products to other applications from HIPE

The HIPE main window provides a menu in the `File` menu called `Interop`. In this menu, currently options are available to register with a Plastic hub (a hub will be started automatically if none is found to be running already), unregister from the hub and the possibility to send products to other applications.

To send a product from HIPE to another application, e.g. VOSpec, launch the other application. The button "External Tools" on the HIPE Welcome page lists a number of VO applications, and provides buttons to launch the applications (among these are Aladin and VOSpec). In HIPE, choose File → Interop → Register with Plastic to connect to the Plastic hub.

Note

If the above command gives an error, you have found a bug in the software. Please report it.

The File → Interop → Send Product To menu will now be filled with all applications that have registered with the hub. Normally this will already include the other application we just launched, because many applications connect to Plastic at start-up. If the other application is not listed in the Send Product To menu, make sure the application is registered with Plastic.

To send a product, select the product, for example by selecting its corresponding variable in the Variables View. Then simply choose the desired application from the File → Interop → Send Product To menu, and the product should appear in the chosen application. Note that this can only be done if there is an overlap between the VO interfaces supported by HIPE and the other application. If the applications have no supported interface in common, no data can be exchanged. This is indicated by the external application appearing in grey in the Send Product To menu.

Note that only *sending* of data is supported. To receive the product back into HIPE, it must be sent from the other application. Refer to the documentation of that application to find out how this is done.

Chapter 5. Running the HIFI pipeline

The HIFI pipeline is used for processing data received from one or more of the four HIFI spectrometers on-board Herschel into a final product that is suitable for interactive analysis.

Data obtained from the Herschel Space Archive has been processed with the HIFI pipeline within the Standard Product Generator (SPG) of the Herschel Science Centre, and is available as Level 0, 0.5, 1 and 2 products. If you desire, it is possible to reprocess data to (and from) any level within HIPE using the hifiPipeline Task. It is designed to: obtain data from a local store of data (or a database, if you have access to the ICC databases); remove instrument-related properties of the data; calibrate the resulting spectra; and, then combine the separate spectra from a single observation. The final product is dependent upon the observation mode and is either a calibrated spectrum, a set of co-added spectra or spectral 3D cubes.

For more information about the pipeline steps and their results, please read the HIFI Standard Product Specification Document, or the HIFI Pipeline Specification document.

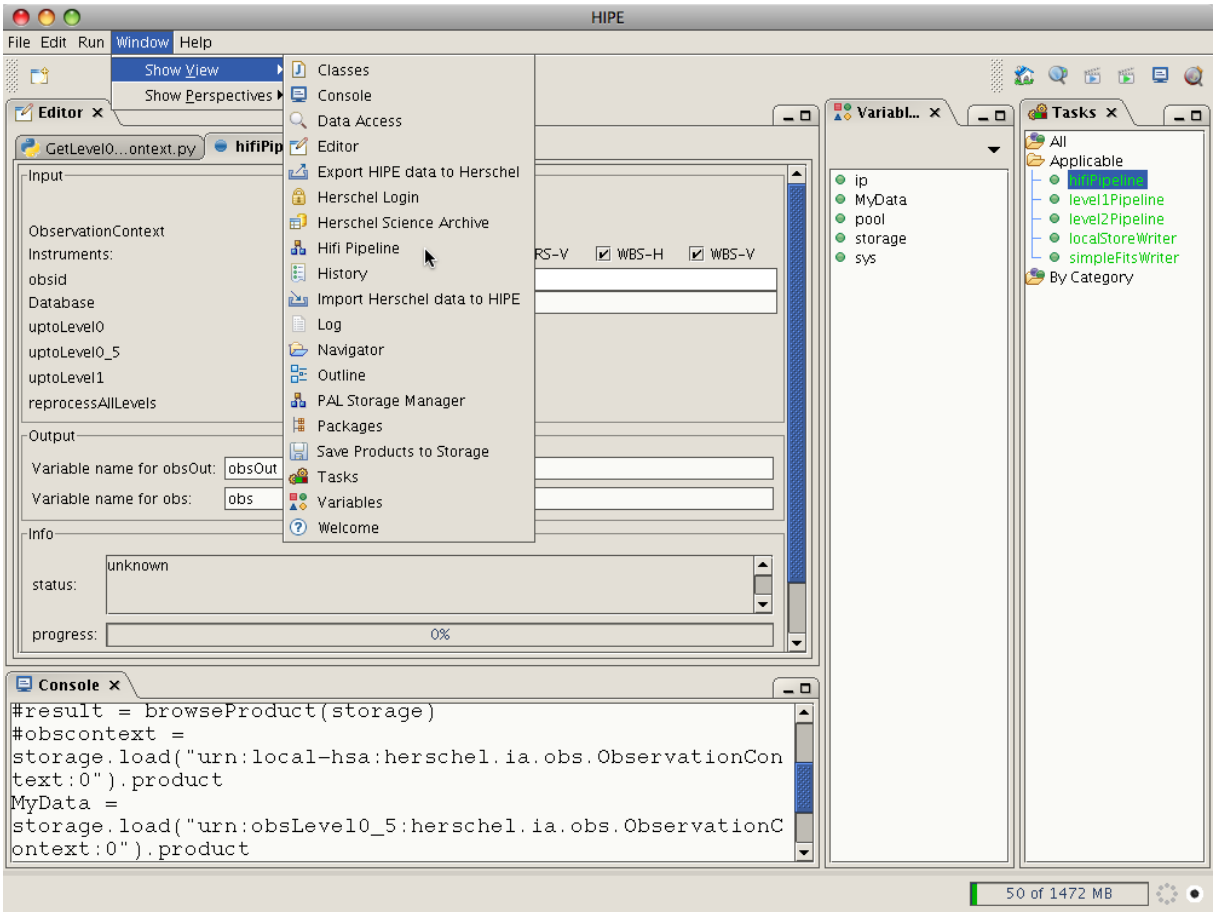
5.1. Running the Pipeline

We can run the HIFI pipeline within HIPE in the following fashion.

1. • 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.

   • A "Hifi Pipeline" View is also available from the HIPE Window menu (under Show View) but it is not fully implemented yet.



Select "hifiPipeline" from the Windows menus or the Task view. If a HIFI observation context has been selected in the Variables view then the "hifiPipeline task" appears in the "Applicable Tasks" folder.

Figure 5.1. Starting the HIFI pipeline task

2. The default (or basic) dialogue allows you to re-process an already existing observation context, e.g. from the Herschel Science Archive, through the pipeline.

   - The way the data is to be reprocessed is defined in the Input section:

   a. 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 observation context bullet.

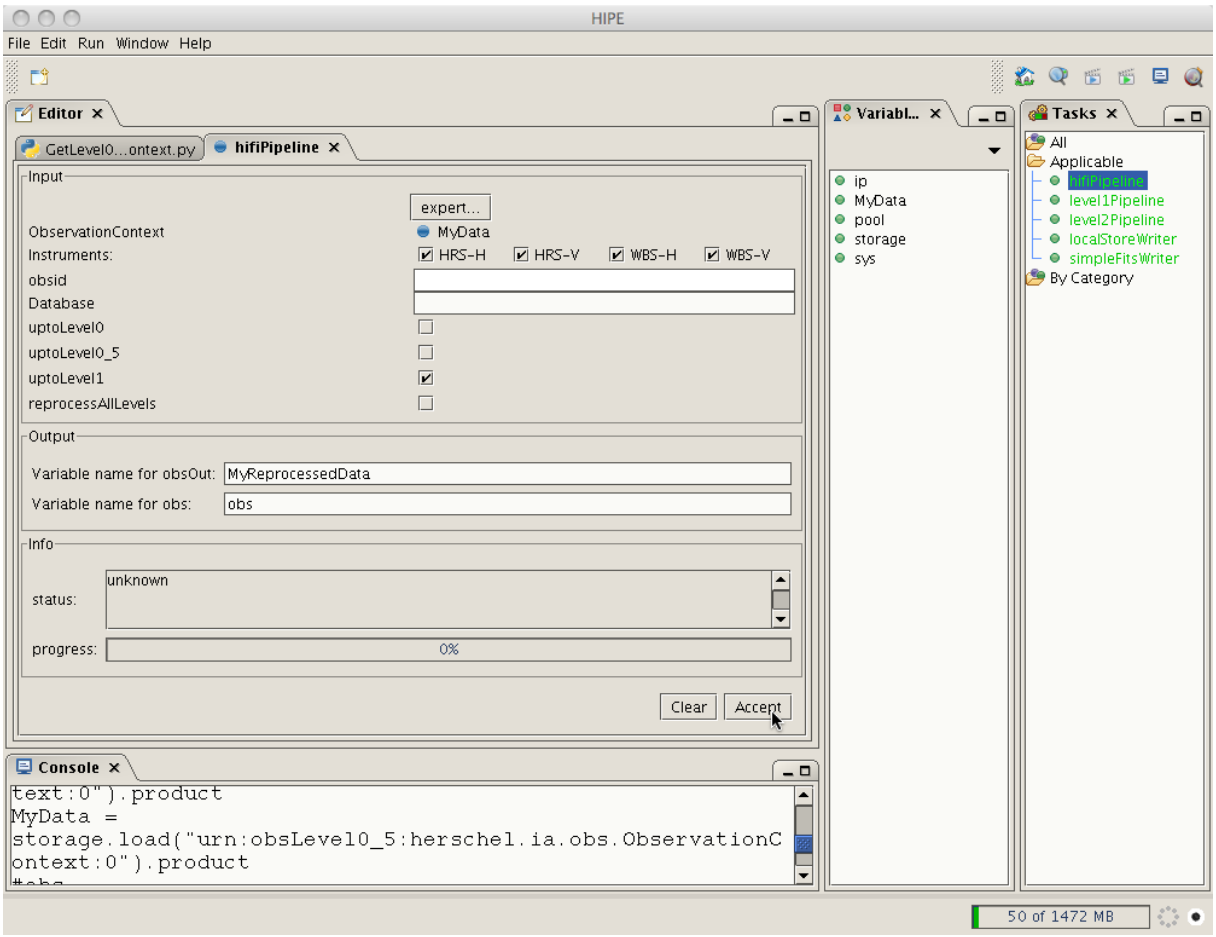
   b. Select the spectrometers you wish to process data for by checking the desired instrument(s) and polarisation(s). Both H and V polarisations of both the Wide Band Spectrometer (WBS) and High Resolution Spectrometer (HRS) are checked by default.

   c. Select which level to (re)process to (0, 0.5, 1, or all levels) by checking the appropriate box. The default set-up will pipeline data through all levels.

   d. If you have permission to access the HIFI ICC databases (only read access is possible), you can process data by typing in an obsid and database name. Note that when using the ICC database calibration information is required in order to process data through the pipeline. You can access this information through the Versant databases at the HIFI ICC or install the `hifi-cal` database locally on your machine. See ??? for details.

- In the Output section, choose the name of the observation context that will be produced or use the HIPE default, `obsOut`. The observation context contains all the products generated by the pipeline task and is stored in a "hifi-pipeline" lstore (~/.hcss/lstore/hifi-pipeline). (The variable `obs` is also produced in order that the pipeline can be re-run without the need to reset IO parameters.)

- 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.



In this example, an already processed observation, 'MyData', is being reprocessed up to Level 1, both polarisations of both spectrometers are included.

Figure 5.2. HIFI pipeline task: default view

3. By clicking on the "expert" button, you may additionally control more detailed aspects of the pipeline set-up. The following items are available to experts only and are generally expected to be used only by HIFI calibration scientists.

   - If you wish to use your hifi-cal lstore, set this up as usual in myconfig and leave the "hifi cal" box blank.

   - Write out comments on the quality of data and processing steps by checking the "quality" box.

   - To set the mode of the observation, type it into the "obsMode" box (required for ILT data).

   - To use a self-defined palStore, drag its name in from the Variable view.

   - Define the test environment by typing in the tmVersion (e.g., ilt-fm)

- Set the execMode

a. It must be "INTERACTIVE" in order that the resulting Observation Context be stored in your palStore.

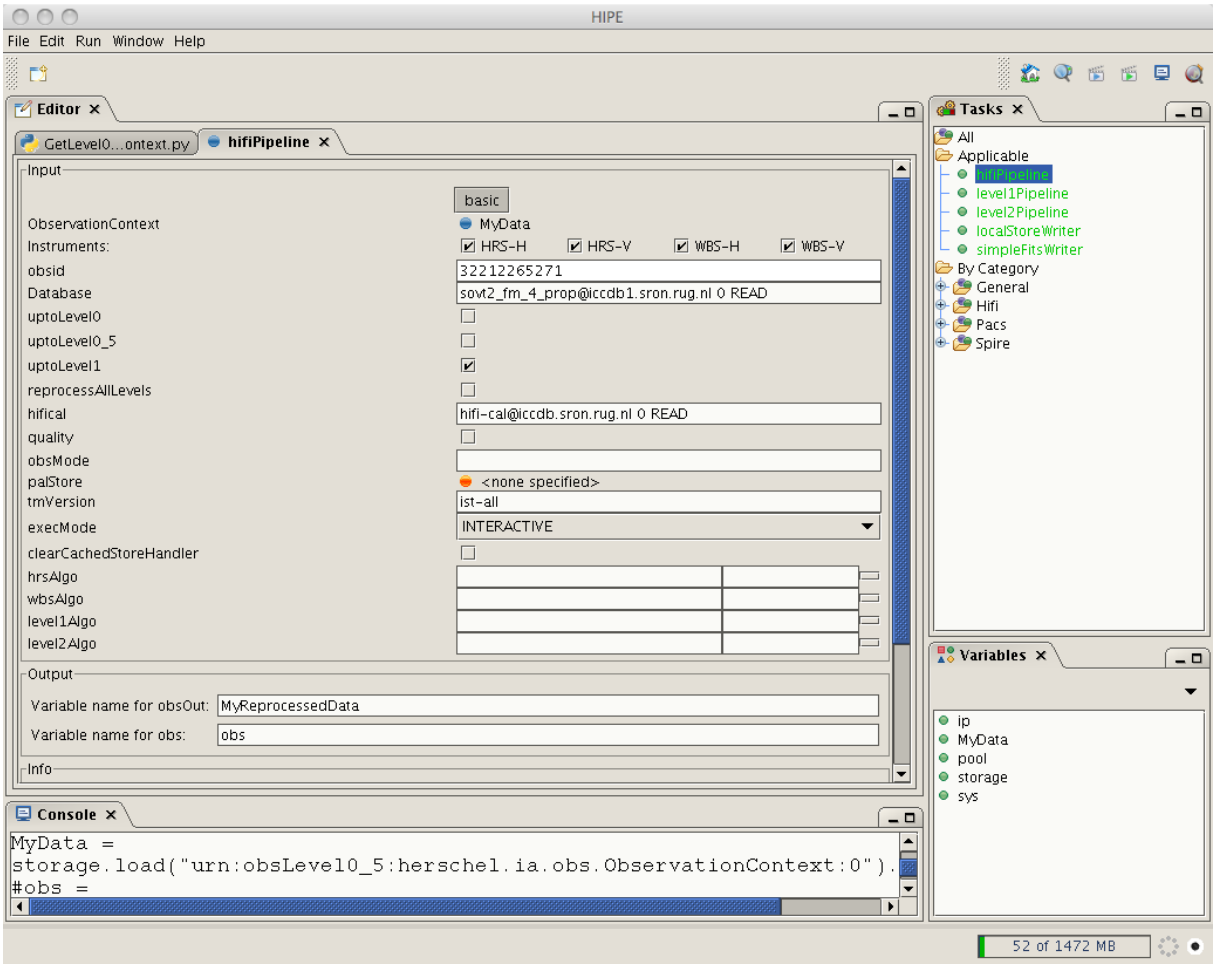
b. "SYSTEMATIC" will update the Observation Context and save it to memory rather than the store.

c. "ON-DEMAND" and "TEST" are used within the SPG environment

- Check the "cache" box to clear the cache store.

- Read in from file your own version of pipeline algorithms.

Toggle back to the default dialogue by clicking on the "basic" button.



Expert users could set up the pipeline from this view.

Figure 5.3. HIFI pipeline task: expert view

5.2. Using the HIFI Pipeline task

Below are several examples showing how to use the HIFI pipeline task.

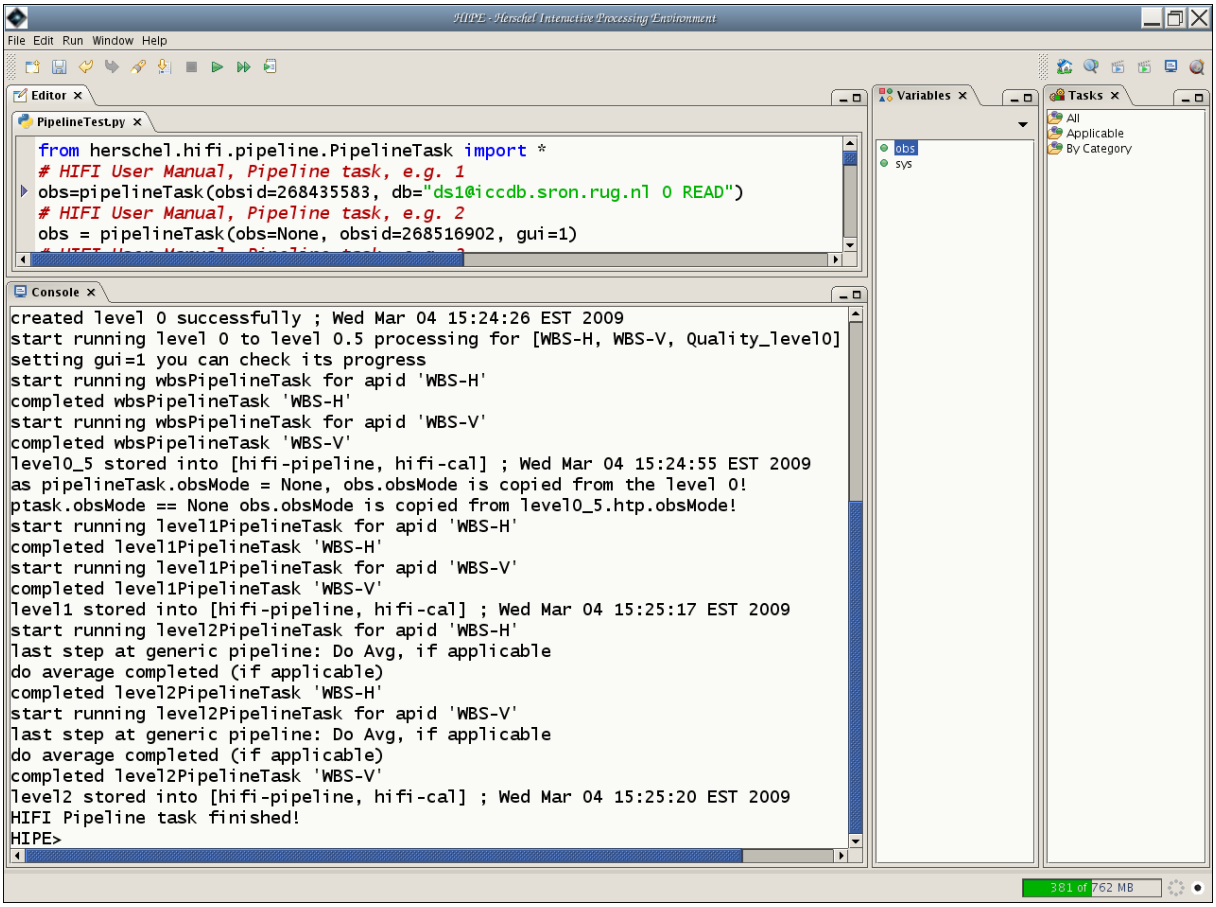
The hifiPipeline task is called into the session by:

```
from herschel.hifi.pipeline.PipelineTask import *
```

1. Run the pipeline and generate an observation context ('obs') from scratch (Figure 5.4):

```
obs = pipelineTask(obsid=268435583, db="ds1@iccdb.sron.rug.nl 0
READ")
```

   If you do not specify the database with `db="..."` then the default (var.database.devel) set in the user's properties will be used.



Eg 1. Running the HIFI pipeline task

Figure 5.4. Running the HIFI pipeline task

2. To re-use the pipeline task, ensure that all the IO parameters are reset by setting `obs=None`

```
obs = pipelineTask(obs=None, obsid=268435583,
db="ds1@iccdb.sron.rug.nl 0 READ", gui=1)
```

   You can use a GUI to show a progress bar (test only with no clear information, at the moment).

3. For ILT data, provide the obsMode name - the data itself does not have it:

```
obs = pipelineTask(obs=None, obsid=268516902,
db="ilt_fm_5_prop@iccdb1.sron.rug.nl 0 READ",
obsMode="HifiPointModeLoadChop")
```

4. Redefine the tmVersion if the selected database requires a different mission phase than the default in your binstruct property `hcss.binstruct.mib.pal.tm_version_map` (hifi default = `"ilt-fm"`):

```
obs = pipelineTask(obs=None, obsid=268439922,
db="ilt_par_5_prop@iccdb.sron.rug.nl 0 READ", tmVersion="ilt-
par")
```

5. The pipeline task automatically processes data from all four spectrometers. You can select an apid for processing to Level 1 (for now, processing to Level 0 always includes all available apids):

```
obs = pipelineTask(obs=None, obsid=268516902,
apids=["1030"], db="ilt_fm_5_prop@iccdb1.sron.rug.nl 0 READ",
obsMode="HifiPointModeLoadChop")
```

6. The pipeline can be re-run in various ways:

   a. Re-run the pipeline, assuming Level 0 is available:

```
pipelineTask(obs=obs)
```

   b. Re-run for Level 0 too. (Note that all calibration and other products are not replaced):

```
pipelineTask(obs=obs, reprocessAllLevels=1)
```

   c. Or just re-generate Level 0:

```
obs = pipelineTask(obsid=268516902,
obsMode="HifiPointModeLoadChop", uptoLevel0=1)
```

   d. You can then use this to process from Level 0 up to Level 0.5:

```
obs = pipelineTask(obs=obs, uptoLevel0_5=1)
```

7. You can edit the algorithm of the pipeline tasks:

```
obs = pipelineTask(obs=None, obsid=268435583,
db="ds1@iccdb.sron.rug.nl 0 READ", wbsAlgo=myWbsAlgo,
hrsAlgo=myHrsAlgo, genericAlgo=myGenericAlgo)
```

   The algorithms can be found in:

   WBS: {build_root}/lib/herschel/hifi/pipeline/wbs/WbsPipelineAlgo.py
   HRS: {build_root}/lib/herschel/hifi/pipeline/hrs/HrsPipelineAlgo.py
   Generic: {build_root}/lib/herschel/hifi/pipeline/generic/GenericPipelineAlgo.py

8. And provide your own palStore to which the pipeline will write to:

```
obs = pipelineTask(obs=None, obsid=268435583,
db="ds1@iccdb.sron.rug.nl 0 READ", palStore = myStore)
```

9. Clear CachedStoreHandler to avoid a block due to none closed stores. Note, this closes ALL stores available in this cache and may affect other applications running in the session.

```
obs = pipelineTask(obs=None, obsid=268435583,
db="ds1@iccdb.sron.rug.nl 0 READ", uptoLevel0=1,
clearCachedStoreHandler=1)
```

10. Finally, if pipeline task is not behaving as you expect you could try a reset:

```
pipelineTask = PipelineTask()
```

5.3. Running the Individual Pipelines using the HIFI pipeline task

The HIFI pipeline task can be easily used to (re)process an observation context up to levels 0, 0.5, 1 and 2 using the `uptoLevel...` check boxes in the HIFI pipeline task (see Figure 5.2). However, it is also possible to run each component of the pipeline individually: the HRS and WBS pipeline Tasks can be run to remove instrumental effects (up to Level 0.5), and the Generic pipeline Task can then be used to intensity calibrate the data (up to a Level 1 or 2 product).

These pipeline tasks, which are intended for more expert users, are run and set-up in the GUI in much the same way as the HIFI pipeline task. Some points to note:

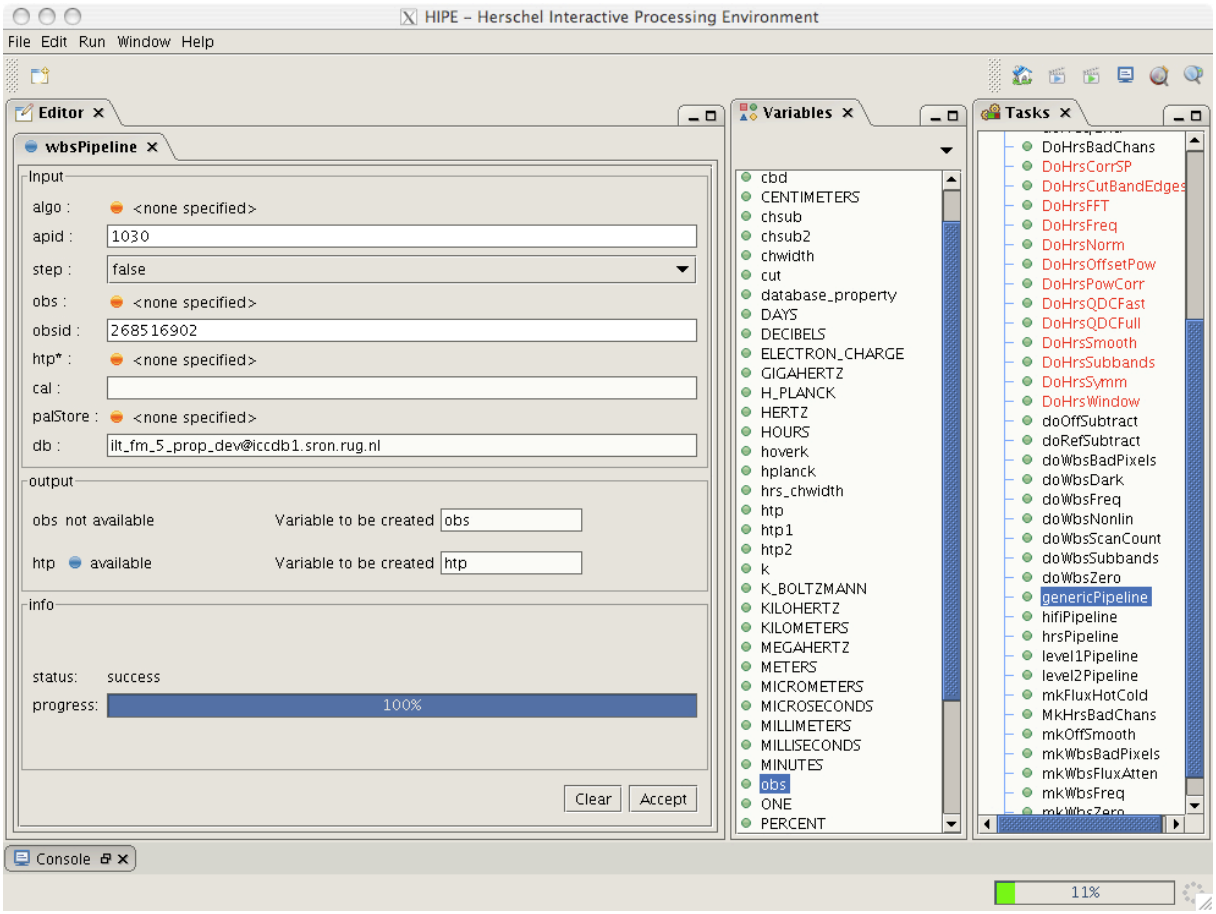
- The individual pipeline tasks can handle both Observation Contexts (ObsContext) and HifiTimeline Products (HTPs), while the HIFI pipeline can handle only Observation Contexts. An Observation Context contains all levels of data, calibration, auxilliary and quality products but an HTP contains only a data set (and meta data). Therefore, processing an HTP is much faster.

- HTP in gives HTP out, ObsContext in gives ObsContext out.

- Spectrometers are identified by apid number 1028 (HRS-H), 1029 (HRS-V), 1030 (WBS-H), or 1031 (HRS-V). Data is processed for only one spectrometer at a time.

- Note that, unlike the hrs- and wbsPipeline Tasks (see Figure below), the genericPipeline Task does not allow to access an observation from a database. Instead an HTP resulting from one of the back-end pipelines or a Level 0.5 Observation Context from the HSA should be passed to it.

In addition, you can (re)process a level 0.5 (1) product up to a level 1 (2) product using the level1Pipeline (level2Pipeline) task. The GUI interface is exactly the same for these pipeline tasks as for the genericPipeline task.



A Level 0 HTP is being reprocessed, only data from the Horizontal polarisation (apid=1030) of the WBS will be used.

Figure 5.5. Running the wbsPipeline task

5.4. Running the Individual Pipeline Tasks

In addition to using the HIFI pipeline task, one can run the underlying pipeline tasks to, for example:

1. Pass an observation context (from running the HIFI pipeline task) to the WBS pipeline task to re-run up to Level 0.5 for only one apid

   ```
   from herschel.hifi.pipeline.wbs.WbsPipelineTask import *

   newobs=wbsPipelineTask(obs=obs, apid=1030)
   ```

2. and then pass that to the Generic pipeline task

   ```
   from herschel.hifi.pipeline.generic.GenericPipelineTask import *

   newobs2=genericPipelineTask(obs=newobs, apid=1030)
   ```

3. Process a HifiTimelineProduct using your own HrsPipelineAlgo

   ```
   from herschel.hifi.pipeline.hrs.HrsPipelineTask import *

   newhtp=hrsPipelineTask(htp=htp, algo=myHrsAlgo)
   ```

4. Load dataframes and housekeeping (create an HifiTimelineProduct)

```
htp          =          wbsPipelineTask(obsid=268516902,
db="ilt_fm_5_prop@iccdb1.sron.rug.nl 0 READ")
```

As these examples illustrate, both ObservationContexts ("obs") and HifiTimelineProducts ("htp") can be passed to these tasks. The parameters used in the Section 5.2 above can also be applied to these tasks.

5.5. Running the Pipeline step by step

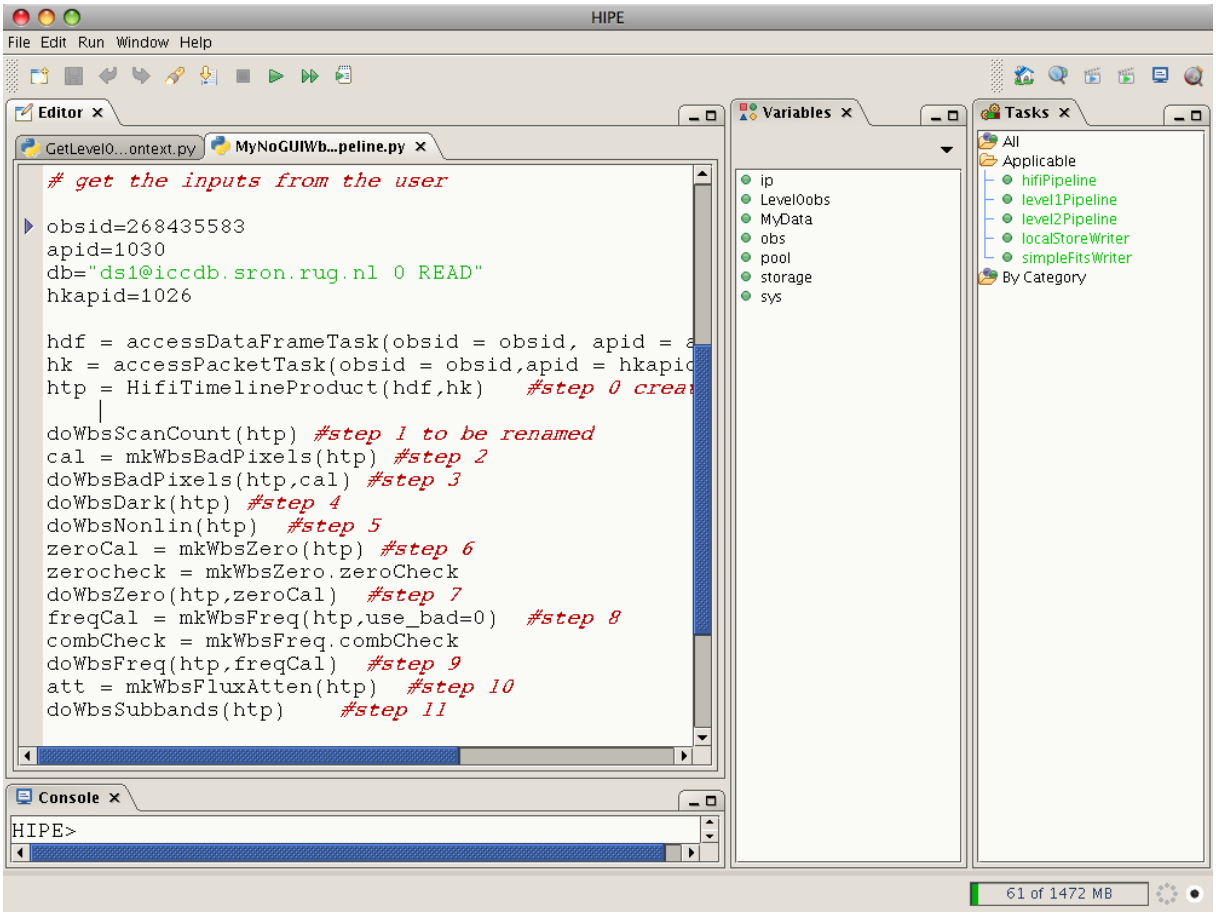
Another way to run the pipeline is step-by-step using the three pipeline branches separately. This is the simplest way to view or modify the pipeline steps, which are contained in the following scripts:

The WBS pipeline is found in `$HCSS_DIR/lib/herschel/hifi/pipeline/wbs/WbsPipeline.py`
The HRS pipeline is found in `$HCSS_DIR/lib/herschel/hifi/pipeline/hrs/HrsPipeline.py`
The Generic (or AOT) pipeline is found in `$HCSS_DIR/lib/herschel/hifi/pipeline/generic/GenericPipeline.py`

As an example, take (again) the simulated WBS-H (apid=1030) HifiPointModeDBS observation with obsid=268435583 from the simulator data database (ds1).

1. Load the WBS pipeline script into your JIDE editor. The obsid, apid and database must be entered manually into the script, see Figure 5.6.

Then step through the WbsPipeline.py script until the end, or play the entire script with the run-all button (two green arrows). The output is a HifiTimeline product, which is stored in a simple Pool called `simple.wbspipeline`



Running the WBS Pipeline Script step-wise

Figure 5.6. Running the WBS Pipeline Script step-wise

2. To run the Generic pipeline, load GenericPipeline.py into your JIDE editor and step through the script.

- The Generic pipeline requires that data have an AOT-like structure. Older obsids, such as gas cell data, do not have this structure and result in the error `herschel.ia.task.SignatureException: params: Null is not allowed`
- The GenericPipeline.py script does not (yet) store the results in a Pool.
- The Generic pipeline script cannot be played with the run-all button but can be run from the command line as follows:

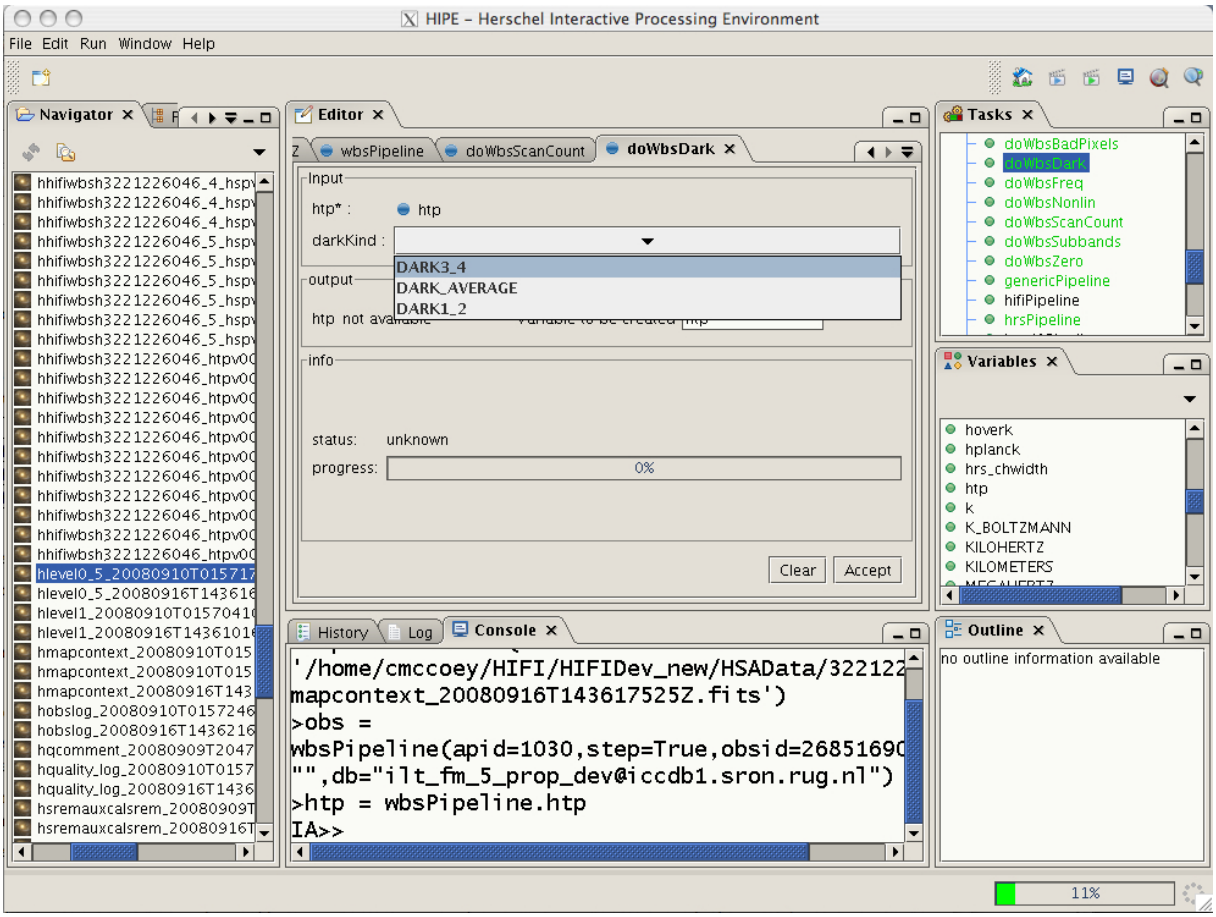
```
from herschel.hifi.pipeline.generic.GenericPipelineAlgo import
runGenericPipeline

newhtp=runGenericPipeline(htp, None)
```

However when the Generic pipeline is run this way, no calibration products are generated.

5.6. Running the Pipeline step by step

The individual steps of the HRS, WBS and Generic Pipelines are also found in the task pane when the HIFI Category is selected. By double-clicking on tasks you may use them to run through the pipeline step by step. As an example, Figure 5.7 shows the GUI for the step in the WBS pipeline that subtracts the darks from the fluxes (DoWbsDark); three methods of dark subtraction can be selected from the

drop-down menu. The HIFI Pipeline Specification document should be consulted for descriptions of each step of the pipeline, as well as the methods used and inputs required.



It is possible to modify the steps taken at stages of the spectrometer pipelines.

Figure 5.7. Running the pipeline step-by-step

Additionally, the wbsPipeline task has an option to be run step-by-step by selecting true from the step drop-down menu, (see Figure 5.2) .

Chapter 6. HowTo run the PACS pipelines within HIPE

Vanessa Doublier-Pritchard, PACS ICC; Katrina Exter, KULeuven

6.1. Introduction

The purpose of this HowTo is to brief users on running the PACS pipelines via the HIPE application, by listing the steps that the pipelines follow. Detailed explanations of what is being done at each stage and how to check your results are given the PACS cookbooks and data reduction guides which are available from the HIPE help main page, and first-time users are *strongly* recommended to read these. The PACS pipelines currently run as a long series of individual tasks, rather than as a single application. The data you will recieve from the Herschel Science Archive (HSA) will have been processed through a pipeline already, and here we list the steps that the pipelines actually do so you could repeat this yourself if you so wished. But, if you want to know what you are doing at each step, you will *have to* read the cookbooks and data reduction guides.

We recommend that you run HIPE with 2G of memory ("hipe -Xmx2000m").

We also recommend that you read the parts of the HowTo documentation that explain how to start and navigate in HIPE, how to select and inspect products, and how to display data. We *do not* repeat those instructions here.

We also recommend that you read the PACS Data Reduction Guide (or similar name), the basic level user documentation for the PACS pipeline. A preliminary version of this guide should be on the HIPE help pages from mid September.

Last edited 28 Feb 2009; rearrangments of spectroscopy in Aug 2009.

6.2. Populate your pool

You will have gotten from the HSA (Herschel Science Archive) a tarfile which you should untar into a directory below your "lstore" directory; HIPE expects this to be at [/Users/me/].hcss/lstore (i.e. off of your home directory). lstore is the default supradirectory in which you place all your "pools", these pools being the location of your PACS data. Normally you will have one pool per tarfile you got from the HSA.

Inside a pool should be everything you will need to process your observation. This includes not just the actual astronomical observation, but also the data products that were used to process the data in the automatic data reduction pipeline, the results of which are included in your pool. These include: spacecraft pointing, time synchronisation data, the satellite orbit, the parameters you entered in HSPOT when you submited the proposal, and the pipeline calibration tables. The automatic pipeline also produces basic spectra and spectral cubes, and generates a quality assessment of the observation. It is these basic cubes that we will look at in this chapter, cubes that have been processed through the pipeline in an automatic fashion.

A pool is also a location into which you can import products that you produce in the course of your data reduction (more of that later). If you wish to share pools, to send someone processed data for example, tar up the whole directory and send them that. The pool's directory name must *not* be changed or HIPE will not be able to find the data therein.

6.3. Start HIPE and get the ObservationContext

What you want to look for in your pool is an "ObservationContext" (the capitalisation and concating is a jython thing). An ObservationContext is a container of products that belong to a specific observation. It provides associations between all the products you need to process that single observation (e.g. pointing products, housekeeping data...). If you are reducing data from stratch you will want to extract from this ObservationContext the Level 0 products.

To get this ObservationContext, first start HIPE. From the main page you go to the Work Bench or the Full Work Bench perspective, by: clicking on the "Work Bench" icon on the HIPE welcome page; clicking on the small blue (Work Bench) or green (Full Work Bench) clapperboard icon at the top right of the HIPE GUI; selecting the menu at the top left Window#Show Perspectives. All of this is explained in the HowTo and we will not repeat everything said there.

It is in the Console section of your work bench that you type commands. If you have already gotten your data from the HSA and unpacked it into a pool located off of your .hcss/lstore directory, you then extract your observation using the command

```
myobs=getObservation(1342182002L, od=<number>)
```

where the first parameter is the OBSID, the observation ID as a number followed by an "l" or "L", and `od` is the observing day of your observation. These information should have been given to you, but also when you search for your observation in the HSA directly the OD and OBSID are listed along with other information. Note that specifying `od` is not always necessary. When you have executed this command, "myobs" will appear in the Variables panel listing.

If you have not aready gotten your data from the HSA and put it in to a pool, then with the following command you can get it and import it into HIPE:

```
myobs=getObservation(1342182002L, useHsa=True)
```

For this to work you must have your HSA username and password written in your ./hcss/user.props file via the following lines:

```
hcss.ia.pal.pool.hsa.haio.login_usr = your username
hcss.ia.pal.pool.hsa.haio.login_pwd = your password
```

(if you are only now writing these in that file, then you will need to restart HIPE for it to take effect.) We recommend you immediately then save myobs to a pool (save to disk)

```
mypool = LocalPool("pool1342182002")
mypool.saveObservation(myobs)
```

where the first command tells HIPE what your new pool will be called (and in this example the new pool will be located in [/Users/katrina/].hcss/lstore/pool1342182002) and the second then puts the ObservationContext (myobs) in that pool (directory).

The full range of parameters for getObservation are

```
myobs=getObservation(obsid [,od=<number>] [,poolName=<string>]
    [,poolLocation=<string>] [,verbose=<boolean>] [,useHsa=<boolean>)
```

where the optional parameters are: `od` (observation day); `poolName` is the name of the pool if you have given it your own, unique name (getObservation by default expects a particular naming convention); `poolLocation`, in case the pool directory is not in /hcss/lstore; and `verbose` for a full reporting.

Since you are reducing the data you will want to start from Level 0. You can selected either (i) *Ramps* or (ii) *Frames* products to work on, these will be called (i) HPSAVGR, HPSAVGB or (ii) HPSFITR or HPSFITB. To do this, on the command line type something like:

```
myramp = myobs.level["level0"].refs["HPSAVGB"].product.refs[0].product
# or
myframe = myobs.level["level0"].refs["HPSFITB"].product.refs[0].product
```

This extracts out the first of the averaged blue ramps (if there is only one you still need to specify refs[0]). To find out how many HPSAVGBs are present, see Fig. 2 from Chap. 1; if you click on the + next to HPSAVGB it will list all (starting from 0) that are present.

An alternative way to get your HPSAVGB..ref[x] product is to click on myobs in the Variables panel to send it to the Editor panel, click on +level0, then on +HPSAVGB to see entries 0 1 2 ... If you only have 1 averaged ramp product then there will only be 0 in here. You can then drag and drop "0" it to the Variables panel. The command that is echoed to the Console will be very similar to the one you typed above, only with a different name on the left (which you will then want to change).

Before beginning you will need to set up the calibration tree. You can either chose that which came with your data or that which is attached to your version of HIPE. The calibration tree contains the information HIPE needs to calibrate your data, e.g. to translate grating position into wavelength, to correct for the spectral response of the pixels, to determine the limits above which flags for instrument movements are set. As long as your HIPE is recent then the caltree that comes with it will be the most recent, and thus most correct, calibration tree. If you wish to recreate the pipeline processed products as done at the Herschel Science Centre you will need to use the calibration tree there used, i.e. that which comes with the data. We recommend you use the calibration tree that comes with HIPE.

```
# from your data
mycaltree=myobs.calibration
# or from HIPE
mycaltree=getCalTree("FM")
# where FM stands for flight model and is anyway the default
```

Currently it is necessary to extract a few other products, in addition to the e.g. HPSAVGB, in order for the pipeline processing steps to be carried out. These are the dmcHead, the pointing product, and the orbit ephemeris. You can get these, to be used later, with

```
pp=myobs.auxiliary.pointing
dmcHead=myobs.level["level0"].refs["HPSDMCR"].product.refs[0].product
orbitephem = myobs.auxiliary.orbitEphemeris
```

Still to be written: how to know what DMCHead product to extract if there is more than one

6.4. PHOT pipeline

Full PACS pipeline documentation explaining the tasks being performed in each module is provided elsewhere. Here we show the steps. Note that what was called "myframe" above is here called "frames" and "mycaltree" is called "calTree".

6.4.1. Level 0 to Level 0.5

Level 0 to Level 0.5 tasks are common to all photometer observing modes:

```
frames = findBlocks(frames, calTree=calTree)
frames = photFlagBadPixels(frames, calTree=calTree)
frames = photFlagSaturation(frames, calTree=calTree)
frames = photConvDigit2Volts(frames, calTree=calTree)
frames = photCorrectCrosstalk(frames, calTree=calTree)
frames = photMMTDeglitching(frames)
frames = convertChopper2Angle(frames, calTree=calTree)
frames = photAddInstantPointing(frames, pp)
frames = cleanPlateauFrames(frames, calTree=calTree)
```

6.4.2. Level 0.5 to Level 2

6.4.2.1. Point Source pipeline

The Point Source data reduction steps are:

- Single command: runPhotometerPointSource.py
- Step-by-step tasks:

```
# Level 0.5 > Level 1  ################################################
frames = photMakeDithPos(frames)
frames = photMakeRasPosCount(frames)
frames = photAvgPlateau(frames)
frames = photAssignRaDec(frames, calTree=calTree)
frames = photDiffChop(frames)
frames = photAvgDith(frames)
frames = photDiffNod(frames)
frames = photCombineNod(frames)
print -"photRespFlatfieldCorrection"
frames = photRespFlatfieldCorrection(frames, calTree=calTree)
frames = photDriftCorrection(frames)

# Level 1 > Level 2  ################################################
image = photShiftDith(frames, copy=1)
```

6.4.2.2. Small Extended Source

The Small Extended Source data reduction steps are:

- Single command: runPhotometerSmallExtendedSource.py
- Step-by-step tasks:

```
# Level 0.5 > Level 1  ################################################
frames = photMakeRasPosCount(frames)
frames = photAvgPlateau(frames)
frames = photAssignRaDec(frames, calTree=calTree)
frames = photDiffChop(frames)
frames = photAvgNod(frames)
frames = photDiffNodSmall(frames)
print -"photRespFlatfieldCorrection"
frames = photRespFlatfieldCorrection(frames, calTree=calTree)

# Level 1 > Level 2  ################################################
image = photProject(frames,calTree=calTree)
```

6.4.2.3. Scan Map -simple-

The Scan Map, default setup, data reduction steps are:

- Single command: runPhotometerScanMap.py
- Step-bystep tasks:

```
# Level 0.5 > Level 1  ################################################
frames = photMakeDithPos(frames)
frames = photMakeRasPosCount(frames)
frames = photAvgPlateau(frames)
```

```
frames = photAssignRaDec(frames, calTree=calTree)
frames = photDiffChop(frames)
frames = photAvgDith(frames)
frames = photDiffNod(frames)
frames = photCombineNod(frames)
print -"photRespFlatfieldCorrection"
frames = photRespFlatfieldCorrection(frames, calTree=calTree)
frames = photDriftCorrection(frames)
```

Level 1 > Level 2

```
image = photShiftDith(frames,copy=1)
```

6.4.2.4. Scan Map

The Scan Map, any setup, data reduction steps are:

- Single command: runPhotometerScanMap.py
- Step-by-step tasks:

Level 0.5 > Level 1

```
frames = photFluxCal(frames)
```

Level 1 > Level 2

```
frames = photAssignRaDec(frames, calTree=calTree)
frames = photHighpassFilter(frames, 200)
#Rem: Input paramters (scale =1 means skypix=dectector pixel)
#crota2 =0.0 of output map
scale          = 1
crota2         = 0.0
tod = makeTodArray(frames, scale, crota2, -"test.tod", -".")
filterLength  = 0  maxRelError   = 1e6  maxIterations = 500
if (runNaiveMapper == None):
    runNaiveMapper = Boolean.FALSE
    map = runMadMap(tod, calTree, filterLength, maxRelError,
    maxIterations, runNaiveMapper)
```

6.4.2.5. Chopped Raster

The Chopped Raster data reduction steps are:

- Single command: runPhotometerRaster.py
- Step-by-step tasks:

Level 0.5 > Level 1 ##

```
frames = photMakeDithPos(frames)
frames = photMakeRasPosCount(frames)
frames = photAvgPlateau(frames)
frames = photAssignRaDec(frames, calTree=calTree)
frames = photDiffChop(frames)
frames = photAvgDith(frames)
frames = photDiffNod(frames)
frames = photCombineNod(frames)
print -"photRespFlatfieldCorrection"
frames = photRespFlatfieldCorrection(frames, calTree=calTree)
frames = photDriftCorrection(frames)
```

Level 1 > Level 2 ##

```
image = photShiftDith(frames,copy=1)
```

6.5. SPEC pipeline

Full PACS pipeline documentation explaining the tasks being performed in each module is provided in the PACS user-level data reduction guide which from mid Sept will hopefully become available, at least for spectroscopy. Here we only list the pipeline steps for the chop-nod AOT, we do not include the same level of detail of the data reduction guide. The data reduction from level 0 to 0.5 are the same for all types of observation.

6.5.1. Level 0 to 0.5

We are starting from a *Ramps* product (myramp) and begin with

```
myramp = specFlagSaturationRamps(myramp,calTree=mycaltree)
myframe = fitRamps(myramp)
myframe = specConvDigit2VoltsPerSecFrames(myframe, calTree=mycaltree)
myframe = detectCalibrationBlock(myframe)
myframe = addUtc(myframe) not yet working
myframe = specAddInstantPointing(myframe, pp, calTree=mycaltree)
myframe = convertChopper2Angle(myframe, caltree=mycaltree)
myframe = specAssignRaDec(myframe, caltee=mycaltree)
myframe = waveCalc(myframe, calTree=mycaltree)
myframe = specCorrectHerschelVelocity(myframe, orbitephem, pp)
myframe = findBlocks(myframe, calTree=mycaltree)
myframe = specExtendStatus(myframe, calTree=mycaltree)
myframe = specFlagBadPixelsFrames(myframe, calTree=mycaltree)
myframe = cleanPlateauFrames(myframe, dmcHead=dmchead, calTree=mycaltrree)
myframe = flagGratMoveFrames(myframe, dmcHead=dmchead, calTree=mycaltree)
# and, if you began from a Frames product (myframe)
myframe = specFlagSaturationFrames(myframe,calTree=mycaltree)
```

6.5.2. Level 0.5 to 2

These stages are AOT specific, and at present we only give the details for a chop-nod point source AOT.

```
myframe = specFlagGlitchFramesQTest(myframe)
# or
myframe = specFlagGlitchFramesQTestChopped(myframe)
# and then
myframe = specCorrectSignalNonLinearities(myframe, calTree=mycalTree)
myframe = convertSignal2StandardCap(myframe,calTree=calTree)
RespandDark = specDiffCs(myframe,calTree=mycalTree)
myframe = specDiffChop(myframe)
myframe = rsrfCal(myframe,calTree=calTree)
myframe = specRespCal(myframe, csResponseAndDark=RespandDark, calTree=mycalTree)
myframe = specAddNod(myframe)
mycube = specFrames2PacsCube(myframe)
# end of Level 1
waveGrid = wavelengthGrid(mycube,calTree=mycalTree,oversample=3,upsample=3)
mycube = specFlagOutliers(mycube)
rebinnedCube = specWaveRebin(mycube, waveGrid)
combinedCube = specProject(rebinnedCube)
# end of Level 2
```

Chapter 7. How to perform SPIRE pipeline processing in HIPE

Herschel SPIRE Editorial Board

version 1.2, 13-September-2009

7.1. SPIRE pipeline processing

This HowTo gives a step by step cookbook of how to run the SPIRE photometer pipelines using HIPE.

7.1.1. SPIRE photometer pipeline processing

7.1.1.1. Preperation for running the SPIRE photometer pipeline within HIPE.

The user can process SPIRE photometer data for each of the various AOTs using the standard pipeline scripts that are bundled in within HIPE. The available AOTs for SPIRE photometry are as follows:

Table 7.1. SPIRE Photometry AOTs

Instrument Mode	HSpot Observation Mode	Description
POF2	Point Source Photometry	Seven-Point Jiggle Map
POF3	Small Map	Small Map - 64-point jiggle Map
POF5	Large Map	Large Scan Map Without Chopping
PARALLEL	Parallel Mode	SPIRE/PACS Parallel Mode

For the purposes of this example, we will obtain and process a POF9 observation (SPIRE/PACS Parallel Mode).

Download the required observation (for our example case, ObsID: 3221226284).This can be retrieved via FTP, or more conveniently using the HSA Retrieval mechanism, which will bring the observation into your HIPE session (see HowTo on accessing the Herschel Science Archive).

We shall create a pool called '3221226284' to store our downloaded data for the PARALLEL mode pipeline.

In the latter case the default name of the downloaded observation product is

```
obsid_<observation number>
# in our case, obsid_3221226284
```

Using the 'Navigator' window, navigate to the directory: ***{HIPE_directory}/scripts/*** within the build tree and choose the appropriate pipeline file (those with the `pipeline.py` extension) as shown in the figure below for the case of the POF3 pipeline script:

Double click on the name of the relevant pipeline script in the Navigator window, and the contents of the pipeline file will appear within the Editor window. The user should note that this is the official script which contains many commands related to the Standard Product Generation (SPG) infrastructure. This pipeline is not meant for stepping through but rather for batch processing.

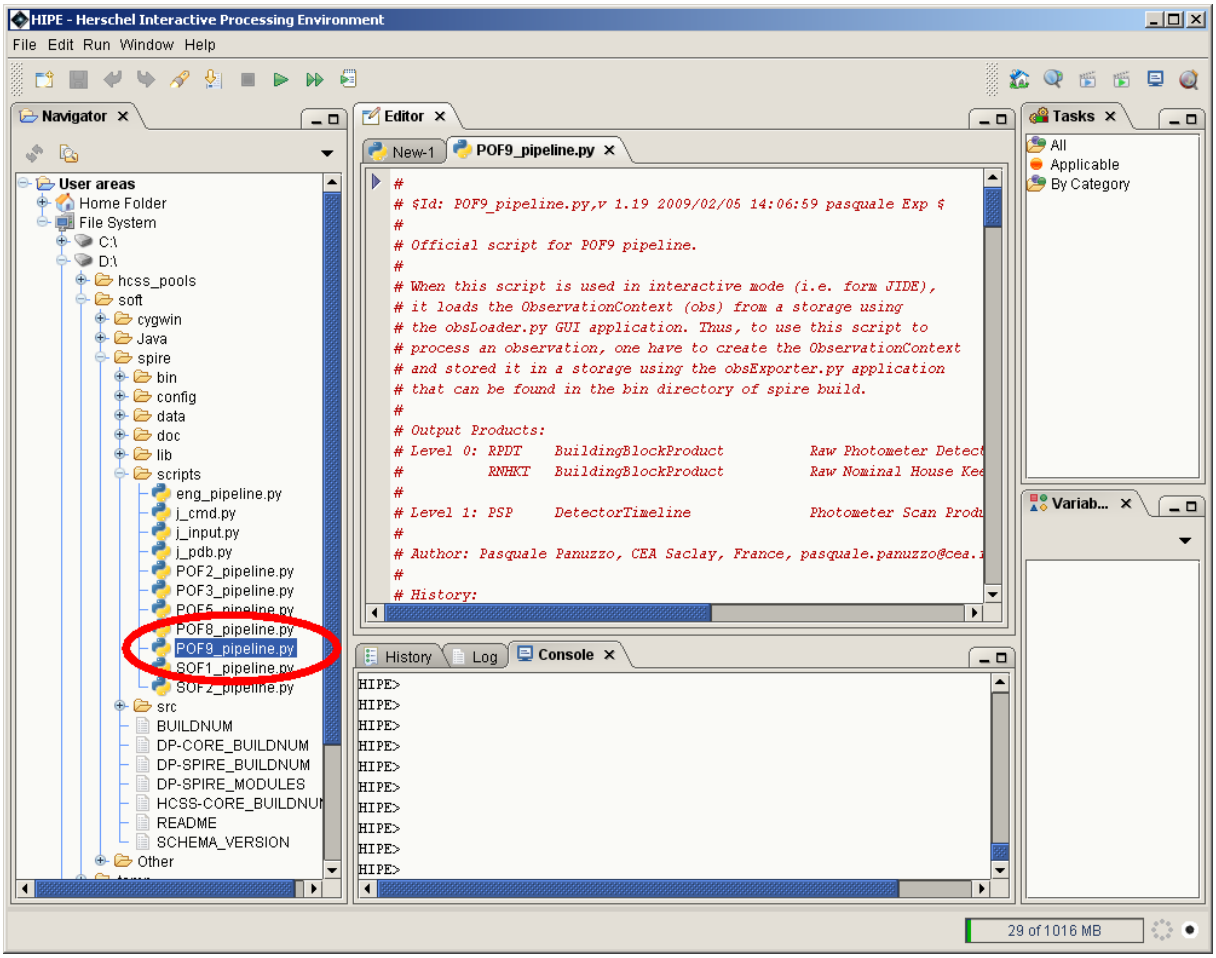


Figure 7.1. The Navigator window within the HIPE GUI

There are two ways to run the pipeline though HIPE:

1) Either running through the pipeline script step-by-step by repeatedly clicking the Play button:

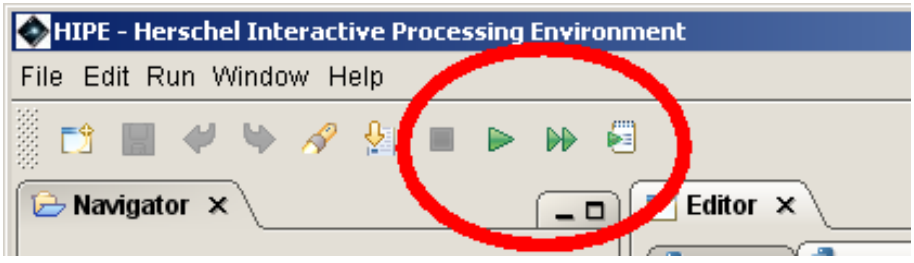


Figure 7.2. The Play button on the HIPE GUI

2) or you can run the entire pipeline straight through by first highlighting the contents of the entire script within the Editor window and then clicking the Play button. Alternatively, you can click the Fast Forward button to run the entire script. Once you have started running the pipeline, you obtain the following window to track the progress of the processing:

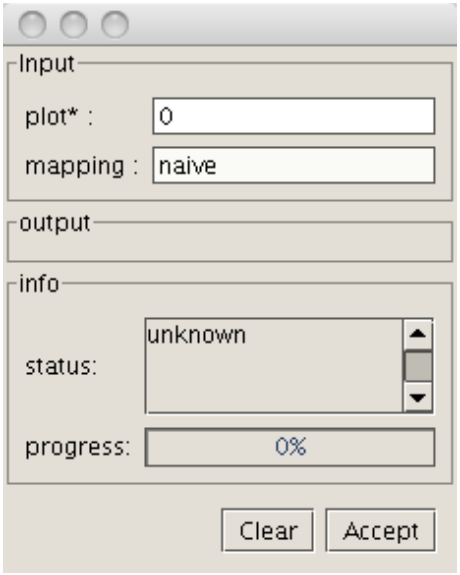


Figure 7.3. Plot dialog.

If you wish to obtain plots, change the 0 to 1 in the 'plot' dialog, otherwise just hit the 'Accept' button. The window will then close automatically.

Next, you will see the `Observation_Loader` window - for HIPE to access and process the data via the pipeline, this requires a Storage ID of the data pool (in this case, obsid_3221226084) and an Observation ID (again in this case, 3221226084) for the data to be processed:

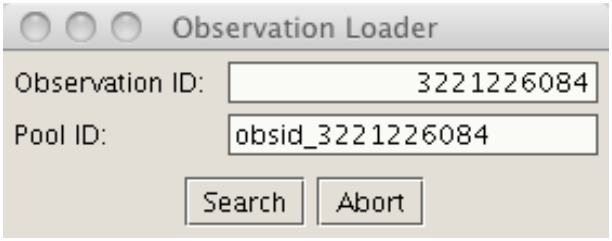


Figure 7.4. Observation Loader.

Then hit Search and the pipeline processing of data will start automatically. Running the pipeline will process data automatically from the initial product Level 0 to the final Level 2 products without further user interaction. However, the user may wish to tweak some of the pipeline processing parameters. In the next Section we will run a customized interactive script to run a POF9 pipeline, with the dataset 3221226084 - if you use a different observation to this, the plots will of course differ.

In HIPE version 1.2, the SPIRE pipelines are in fact integrated within HIPE, which should make the execution of pipeline processing much more straightforward, without having to deal with individual scripts. To do this,

- load in your HIPE section a SPIRE photometry observation;
- select the observation context in the Variables view of HIPE,
- go in the Task view, look in the "Applicable" list, and you'll find a task called spire****Pipeline (where **** can be PointSource, LargeMap etc),
- double click on it, and a GUI will appear;
- click on "accept" and it will run the pipeline script for you.

7.1.1.2. Running the SPIRE photometer pipeline interactively.

The pipeline for **Level 0.5 to Level 1 processing** involves the following sequence of processing modules. The pipeline works on a Photometer Detector Timeline (PDT) and requires the Nominal Housekeeping Timeline (NHKT). Additional auxilliary products are required for the telescope pointing information (see the flowchart below)

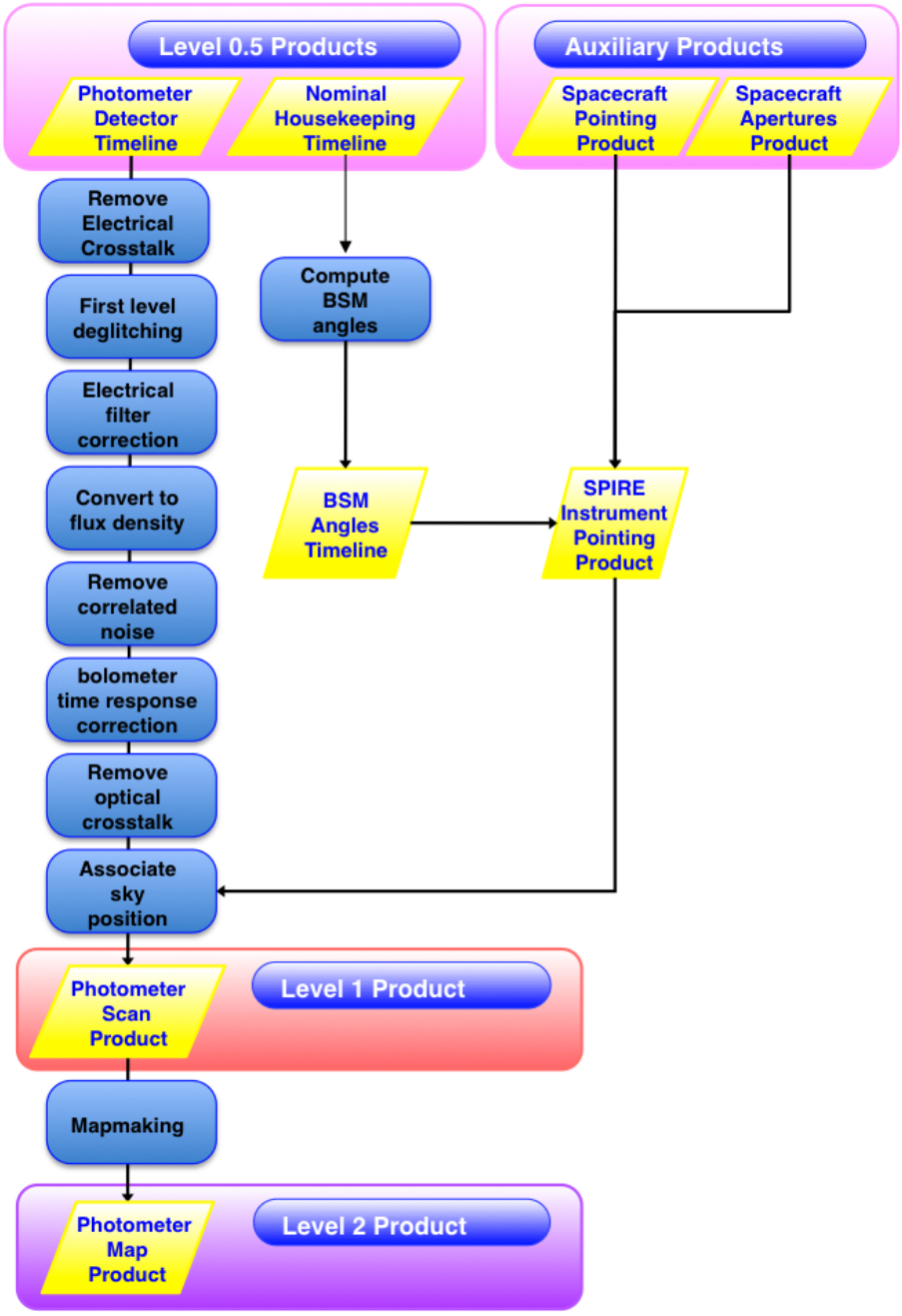


Figure 7.5. The SPIRE POF9 pipeline.

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

```
for bbid in bbids:
 block=level0_5.get(obsid,bbid)
 pdt=block.pdt
 pdt=elecCross(pdt,table=obs.calibration.phot.elecCross)
 pdtList.append(sink.save(pdt))
```

Now,for this observation, we know that detector timeline #5 contains a glitch in detector "PMWA13" at sample 135:

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

We can start to take steps to correct this glitch. First we get the voltage of detector "PMWA13". The getVoltage() method is defined for DetectorTimeline objects:

```
voltage=pdt.getVoltage("PMWA13")
```

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

```
time=pdt.sampleTime
```

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

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

Get the name of the unit of the voltage:

```
uni=pdt.getVoltageUnit("PMWA13").toString()
```

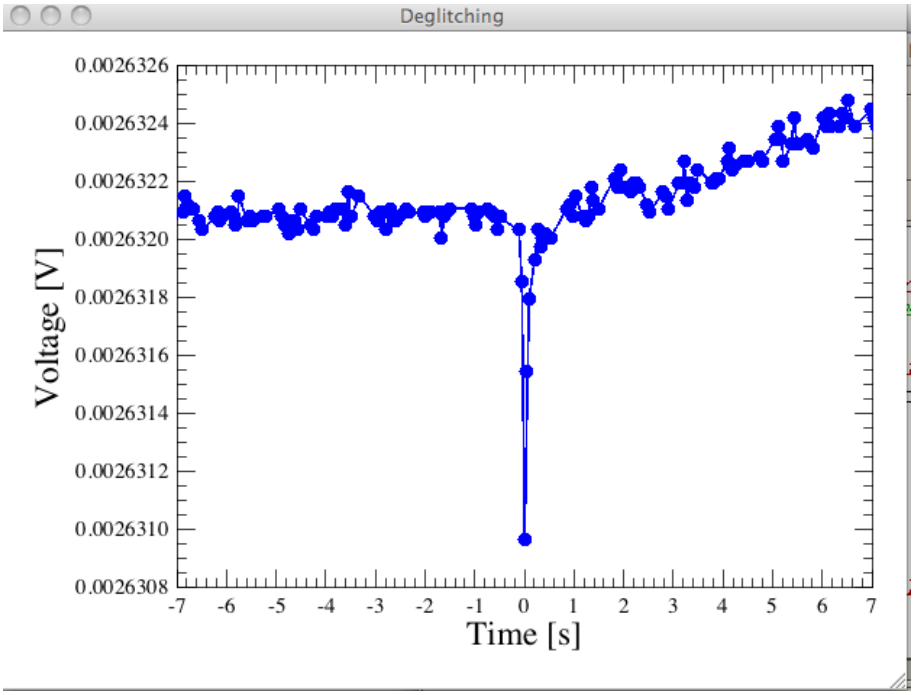


Figure 7.6. Plotting voltage against time.

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

```
plot1=PlotXY(time,voltage,color=Color.blue,xrange=[-7,7],\
   xtitle="Time [s]",ytitle="Voltage ["+uni+"]",name="Deglitching")
plot1[0].style.stroke=1
plot1[0].style.line=2
plot1[0].style.symbol=14
```

To correct, we run deglitching on all scan lines:

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

Now we get the same timeline after deglitching:

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

Again we get the voltage of detector PMWA13:

```
volt_deg=pdt_deg.getVoltage("PMWA13")
```

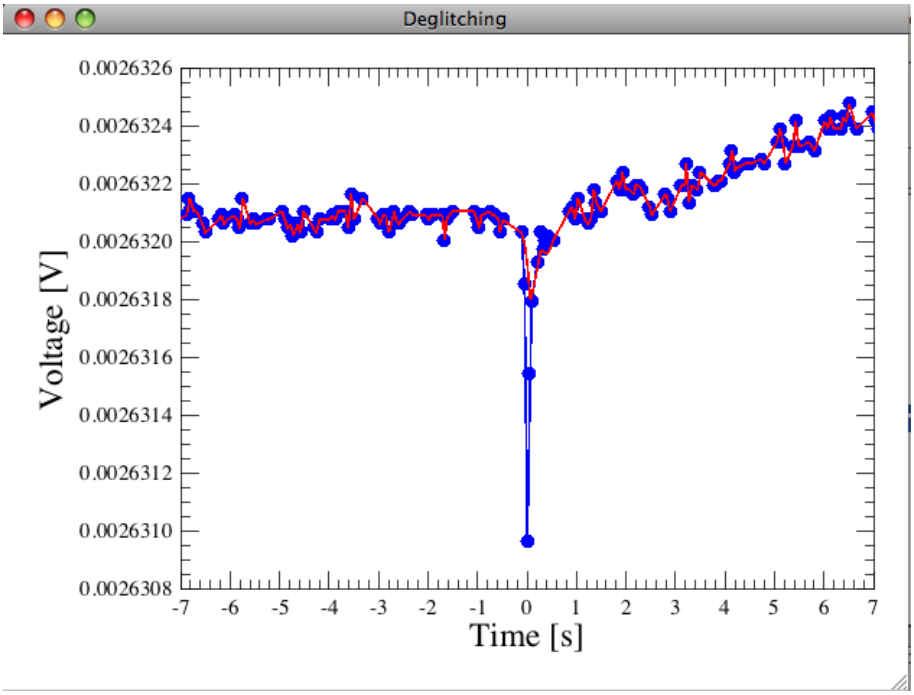


Figure 7.7. And we can overplot on the old timeline. We are currently optimising the parameters of the deglitching task to improve the glitch subtraction - glitch removal in the final version of the pipeline should be much improved over what is shown in this figure!

Overplot on the old timeline:

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

Now we apply the Electrical Filter Response Correction

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

Now we run Flux Conversion:

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

And let's plot the signal seen by the detector "PSWE10" of the first scan line. We will compare it with the result of the temperature drift correction:

```
pdt=pdtList[0].product
signal=pdt.getSignal("PSWE10")
time=pdt.sampleTime-t0
```

And obtain the name of the unit of the signal:

```
uni=pdt.getSignalUnit("PSWE10").toString()
```

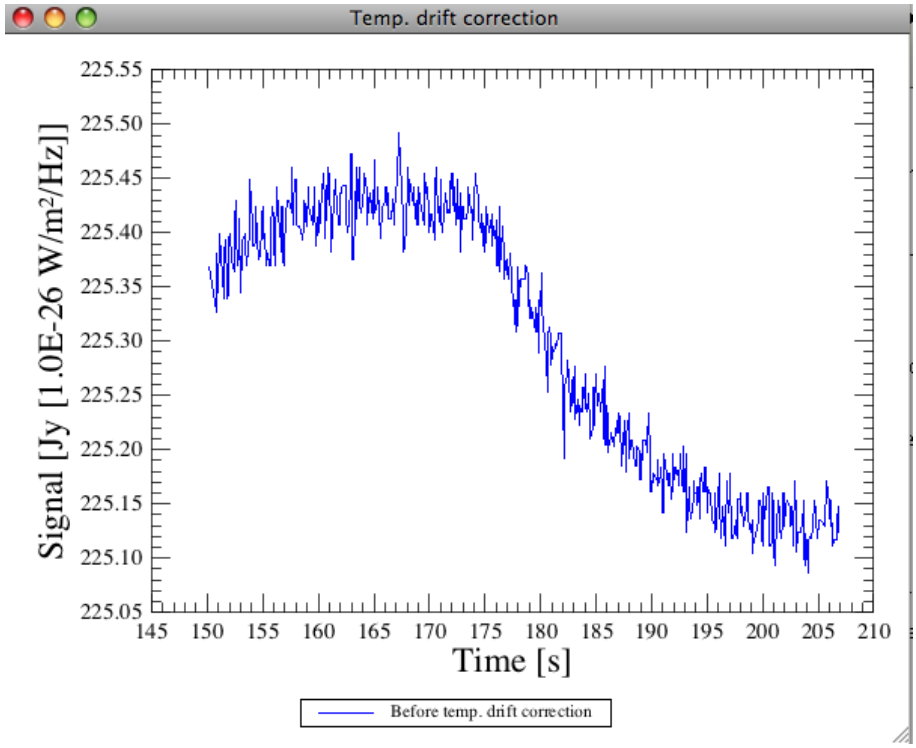


Figure 7.8. Plotting signal vs. time.

And plot the signal versus time:

```
plot2=PlotXY(time,signal,color=Color.blue,\
 xtitle="Time [s]",ytitle="Signal ["+uni+"]",name="Temp. drift correction")
plot2[0].name="Before temp. drift correction"
plot2.legend.visible=1
```

Apply correction for temperature drift

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

Get the corrected PDT:

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

Get the signal of the same detector:

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

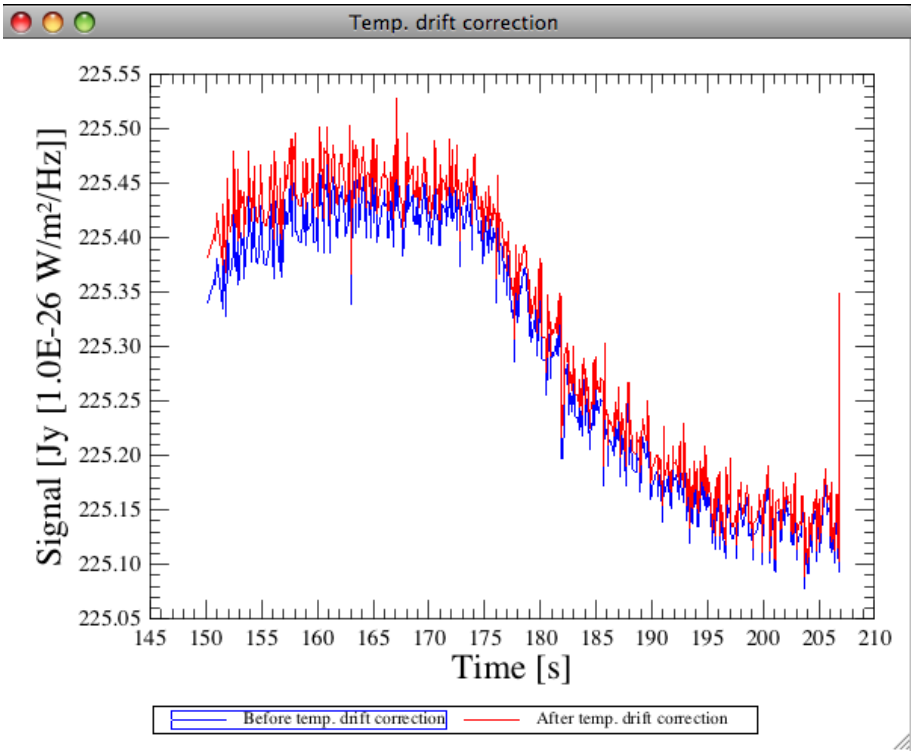


Figure 7.9. After Temperature Drift Correction.

And finally overplot it over the original signal v time plot before the temperature drift correction:

```
plot2[1]=LayerXY(time,signal_corr,color=Color.red,name="After temp. drift
correction")
```

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

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

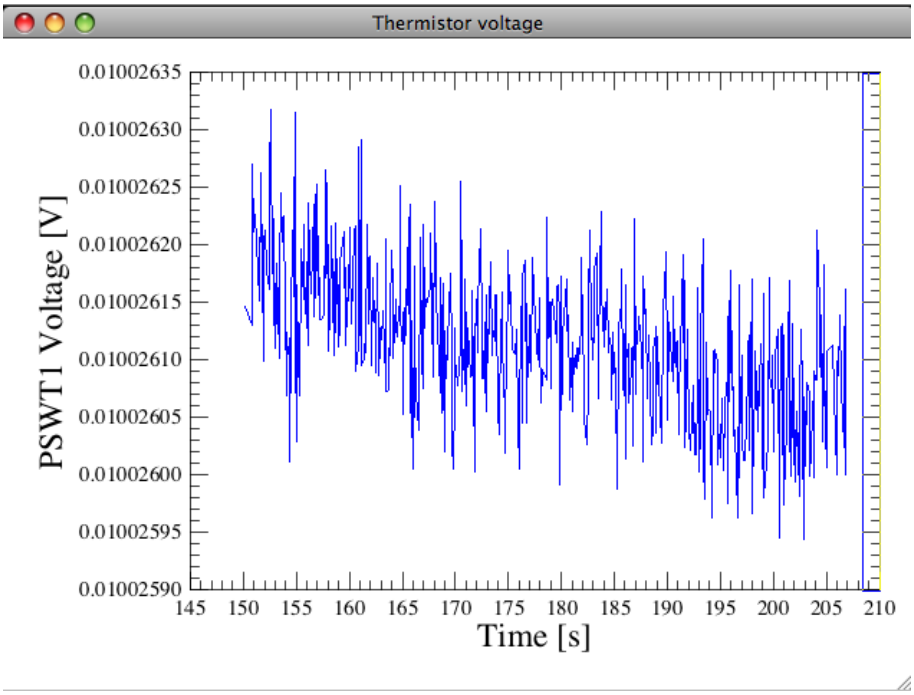


Figure 7.10. PSWT1 voltage.

Apply the bolometer response correction:

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

Apply the Optical Crosstalk Correction:

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

Create a Spire Pointing Product:

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

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

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

In this loop we compute the pointing:

```
for i in range(nscans):
 block=level0_5.get(obsid,bbids[i])
 nhkt  = block.nhkt
# calculate BSM angles
bat=calcBsmAngles(nhkt,bsmPos=obs.calibration.phot.bsmPos)
#
```

```
# add the Bsm Angles Timeline to the SpirePointingProduct
spp.bat=bat
# associate sky positions to flux samples
pdt=pdtList[i].product
ppt=associateSkyPosition(pdt,spp=spp)
scanCon.refs.add(sink.save(ppt))
```

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

Run MADmap map making for the three bands:

```
mapPsw=madScanMapper(scanCon, array="PSW")
mapPmw=madScanMapper(scanCon, array="PMW")
mapPlw=madScanMapper(scanCon, array="PLW")
```

Save maps in the sink and attach them in the ObservationContext

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

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

Saving the data maps for each photometer array

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

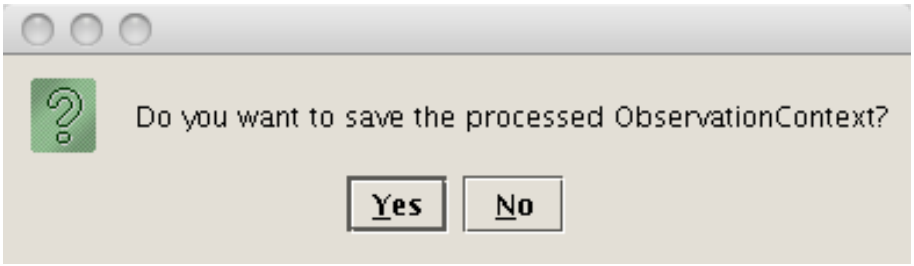


Figure 7.11. Observation context dialog.

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

Saving the data maps for each photometer array

In order to browse the processed data, within the 'Variables' window, select 'obs' from the list of the available variables - this is the variable containing the final observation context. Doing this will bring up the data summary information in the Editor window:

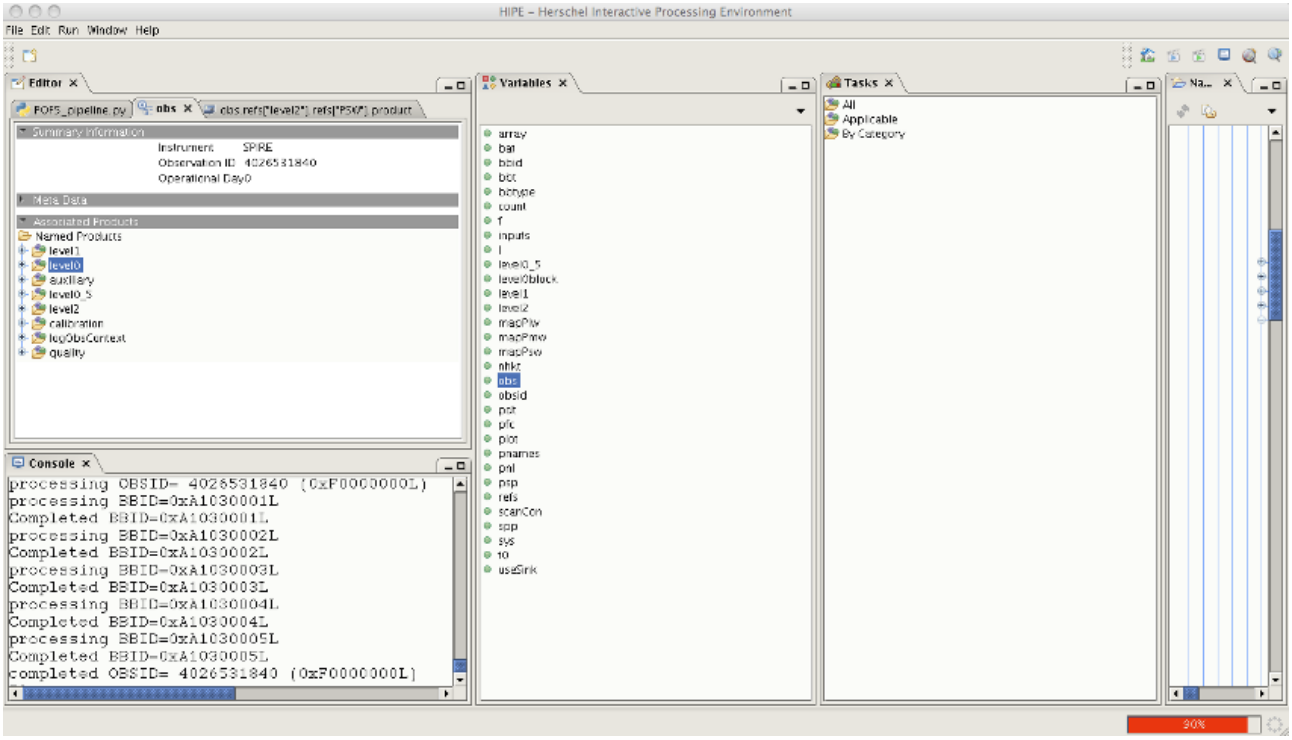


Figure 7.12. Data summary information.

Furthermore, different levels of data processing can be accessed and inspected from the associated products window. For the point source photometry pipeline (POF2), the final products are Level 1 products - namely extracted fluxes. For the remaining mapping pipelines, the final pipeline processing products will be in the form of maps, naturally. Typical example of a final pipeline processing product is shown below, where we have accessed the level 2 product maps from the POF9 scan map pipelines, the PSW, PLW and PMW map products:

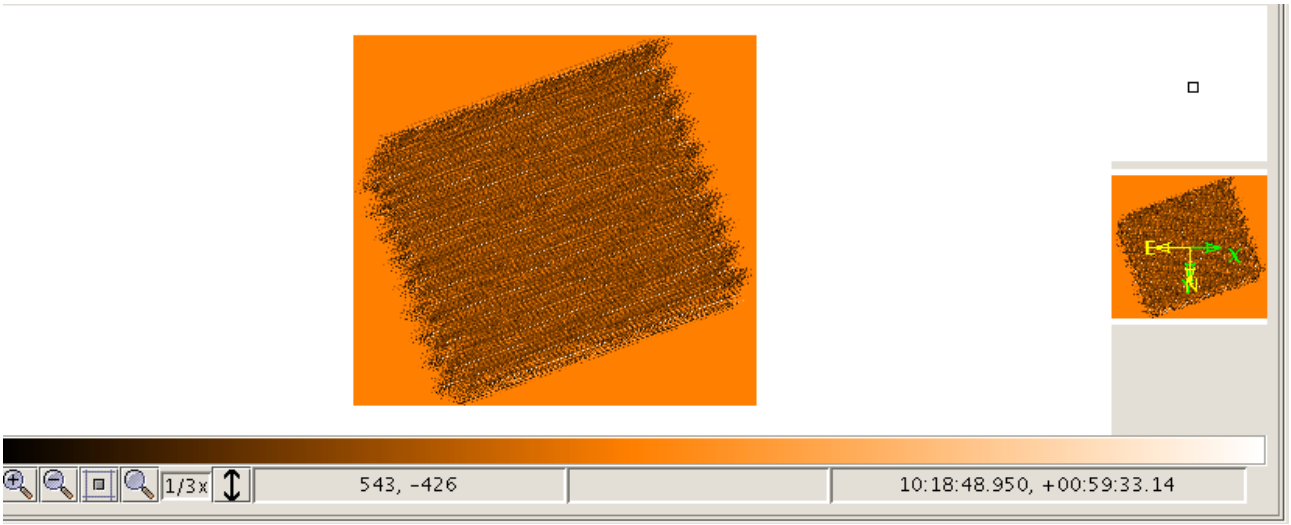


Figure 7.13. Level 2 PSW product from POF9 pipeline.

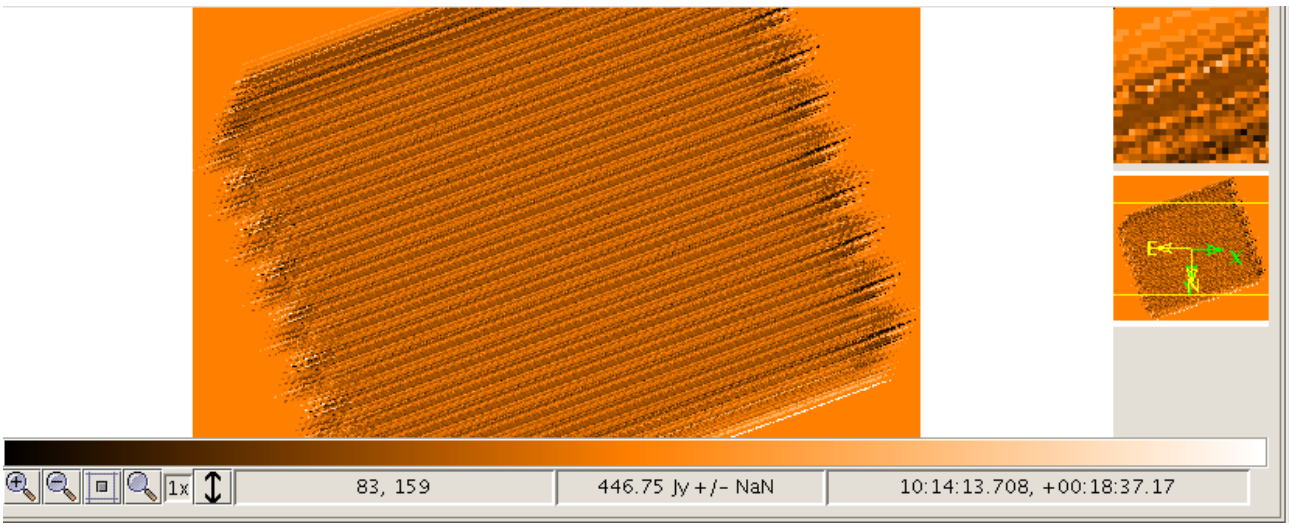


Figure 7.14. Level 2 PMW product from POF9 pipeline.

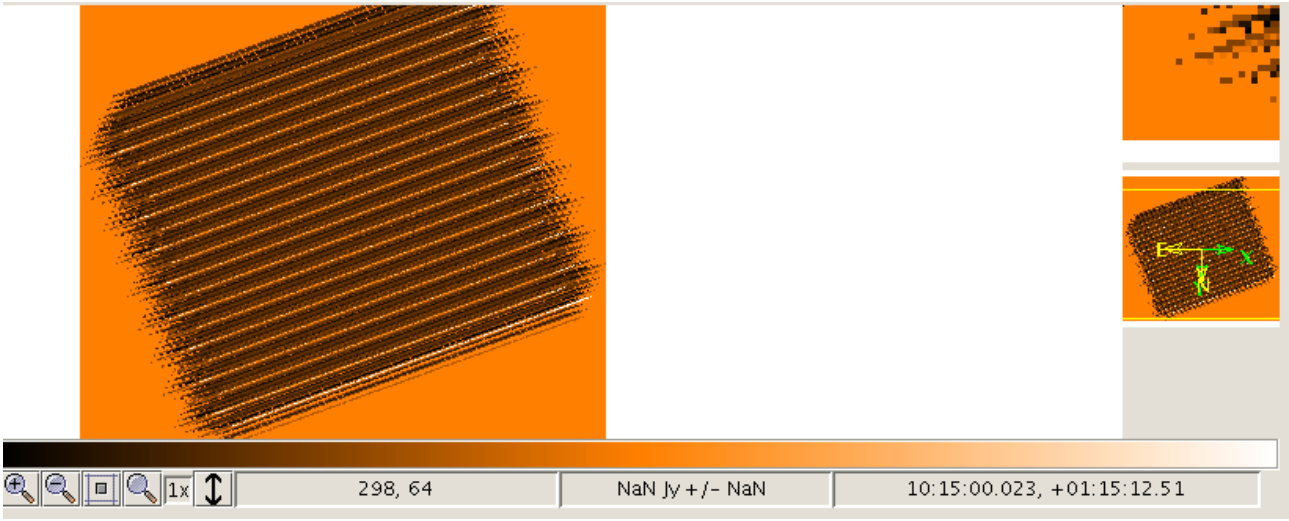


Figure 7.15. Level 2 PLW product from POF9 pipeline.

7.1.2. SPIRE spectrometer pipeline processing.

7.1.2.1. Preparation to running the SPIRE spectrometer pipeline within HIPE.

The process for preparing to run a SPIRE spectrometer pipeline within HIPE is a very similar process to that for preparing to run a photometry pipeline - the user must obtain their data and set up a pool for data storage in the same manner as for photometry - see section 7.1.1 for instructions on how to do this.

The user can process SPIRE spectrometer data for each of the various AOTs using the standard pipeline scripts that are bundled in within HIPE. The user should note that the official scripts contains many commands related to the Standard Product Generation (SPG) infrastructure. This pipeline is not meant for stepping through but rather for batch processing. The available AOTs for SPIRE spectrometry are as follows:

Table 7.2. SPIRE Spectrometry AOTs

Instrument Mode	HSpot Observation Mode	Description
SOF1	Point-source spectrometry	

Instrument Mode	HSpot Observation Mode	Description
		Point Source Spectrum (Continuous Scan)
SOF2	Raster Mapping spectrometry	Fully Sampled Spectral Map within FOV (Continuous Scan)

As mentioned in Section 7.1.1., as of HIPE version 1.2, the SPIRE pipelines are in fact integrated within HIPE, which should make the execution of pipeline processing much more straightforward, without having to deal with individual scripts. To do this,

- load in your HIPE section a SPIRE spectrometry observation;
- select the observation context in the Variables view of HIPE,
- go in the Task view, look in the "Applicable" list, and you'll find a task called spire****Pipeline (where **** can be SinglePointing, Raster etc),
- double click on it, and a GUI will appear;
- click on "accept" and it will run the pipeline script for you.

7.1.2.2. Running the SPIRE spectrometer pipeline interactively.

For the purposes of this example, we will obtain and process a observation (point-source spectrometry), ObsID=0x300117FE within your pool. (This OBSID number is in Hex - generally you will search the HSA in decimal notation for the OBSID.) That observation was an observation where a 202 μm laser was shone on SSWD3. We base the steps for this spectroscopy HOWTO upon the script used for the May 2009 Data Reduction workshop. One can attempt processing sample SPEC data by accessing data from the May 2009 data analsysis workshop by accessing:

- ftp://ftp.sciops.esa.int/
- user: hipe_user
- pass: beta_tester
- Directory: v1.0_May09/SPIRE_SPEC/DATA/

Run the demo script to the point where interferograms are generated.

Run all commands up to and including:

```
sdi=createIfgm(sdt=sdt, smect=smect, hkt=nhkt,
        calSmecZpd=obs.calibration.spec.smecZpd,
        calSpecChanTimeOff = obs.calibration.spec.chanTimeOff,
        calSpecSmecStepFactor=obs.calibration.spec.smecStepFactor,
        interpolType= -"spline")
```

After the commands run to completion, inspect the resultant interferograms. In particular note the difference between the interferograms on SSWD3 and SSWA2 (or any of the SLW pixels).

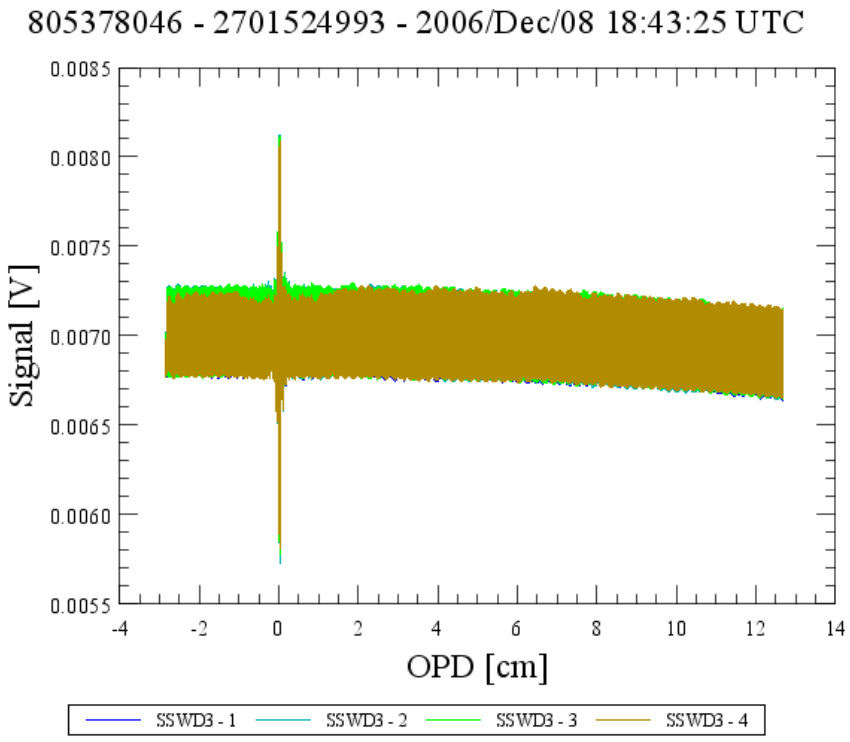


Figure 7.16. Interferograms for detector SSWD3

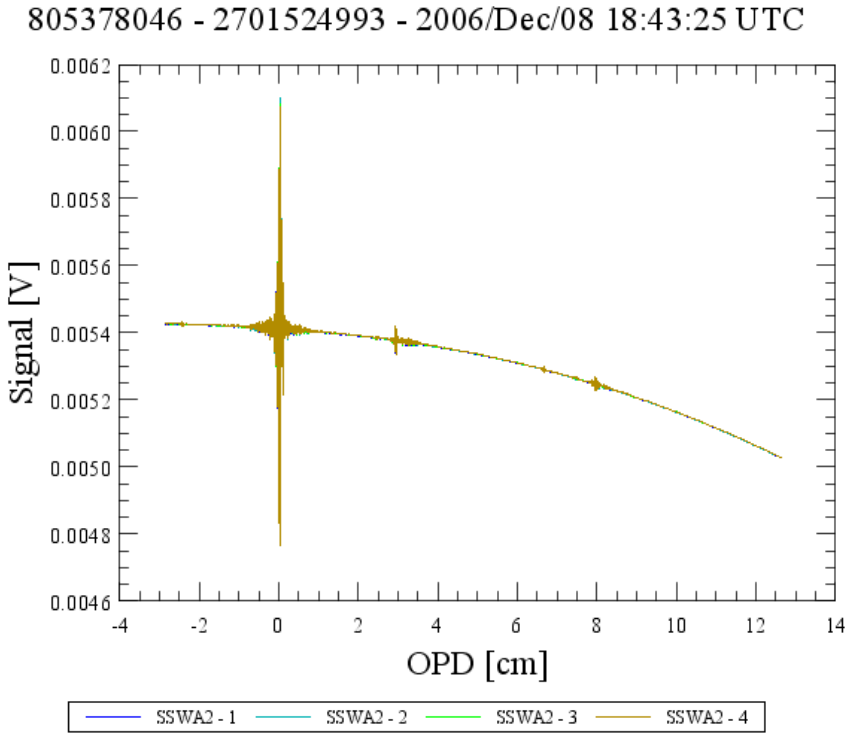


Figure 7.17. Interferograms for detector SSWA2. The baseline is clearly visible.

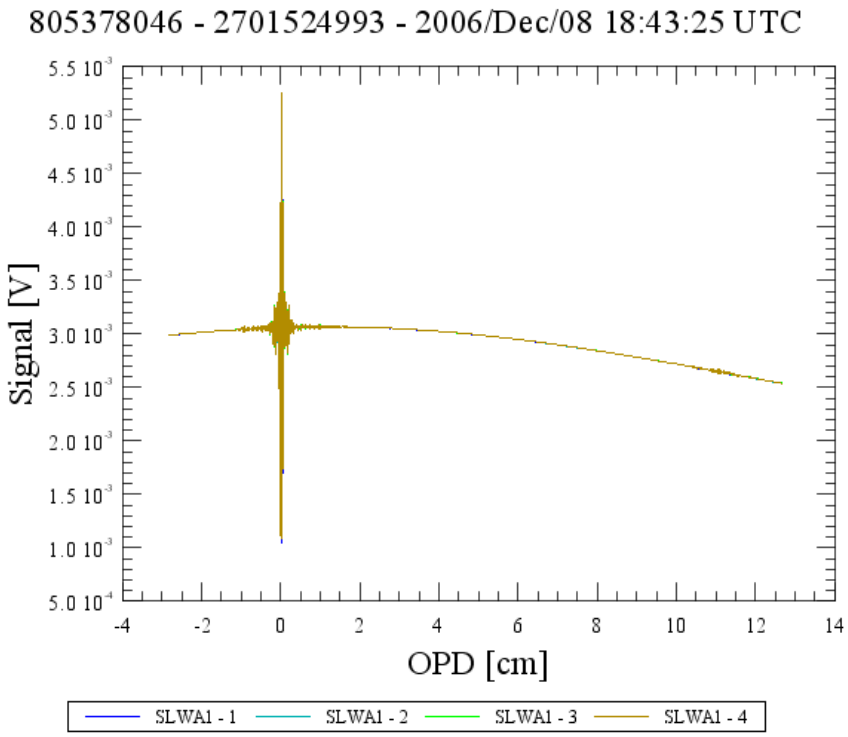


Figure 7.18. Interferograms for detector SLWA1. The baseline is clearly visible.

As shown in the above figures, the laser signature is clearly visible on SSWD3, while the position-dependent baseline is clearly visible for SSWA2 and SLWA1. Next, remove the OPD-dependent offset. One can modify the degree argument to change the order of the fitted polynomial (4 is the default).

```
sdi=baselineCorrection(sdi=sdi, type= -"polynomial", degree = 4)
```

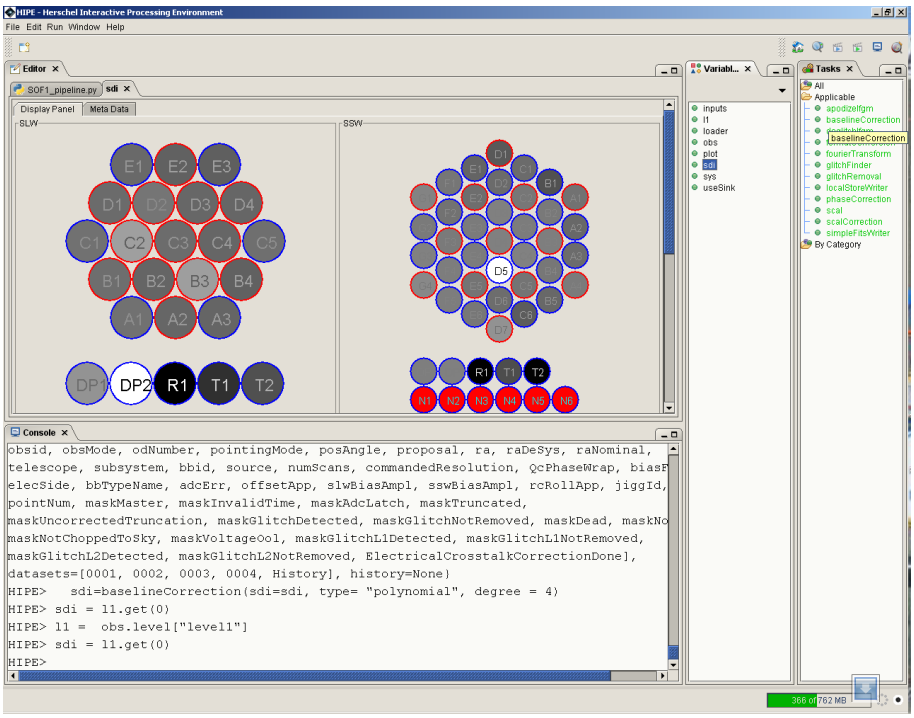


Figure 7.19. Task view for baseline correction.

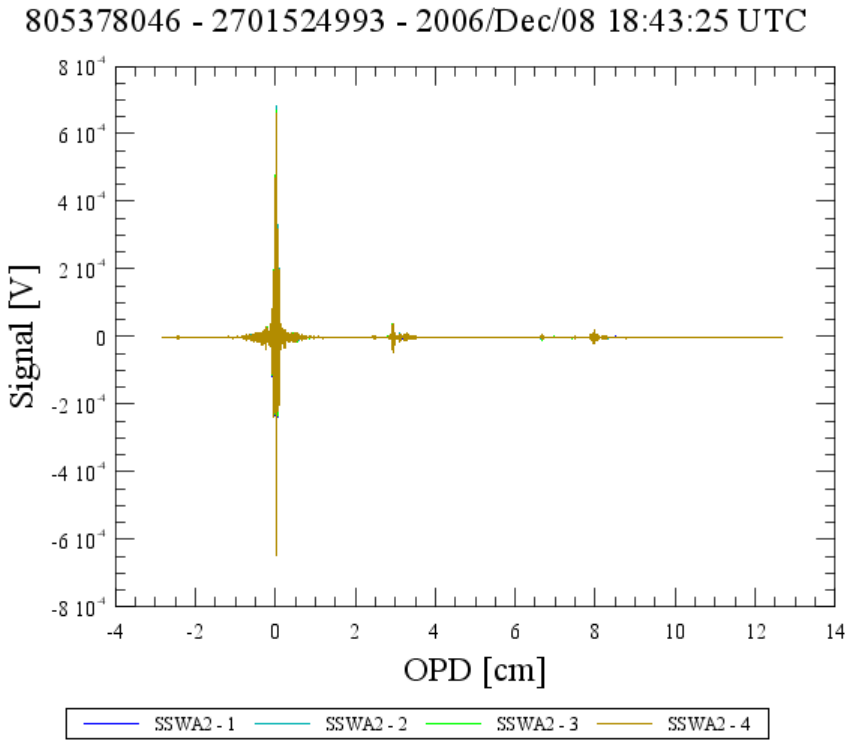


Figure 7.20. Interferograms for detector SSWA2 after baselineCorrection().

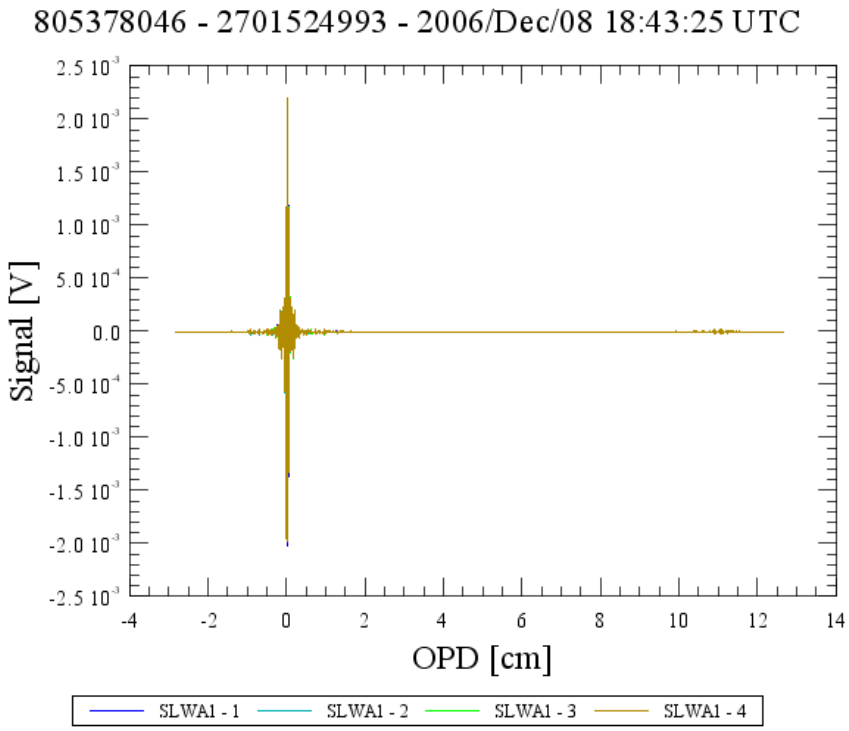


Figure 7.21. Interferograms for pixel SSWA1 after baselineCorrection().

Here again, note the difference in the interferograms before and after this step (In particular, note the differences for the outer edge pixels, compare Figure 7.17 and Figure 7.18 with Figure 7.20 and Figure 7.21). Next, we prepare the interferograms for phase-correction. First we extract the portion of the interferograms symmetric about OPD=0 and apodize these interferograms.

```
presdi = apodizeIfgms(sdi=sdi, apodType="ds", apodFunctionName="aNB_15")
```

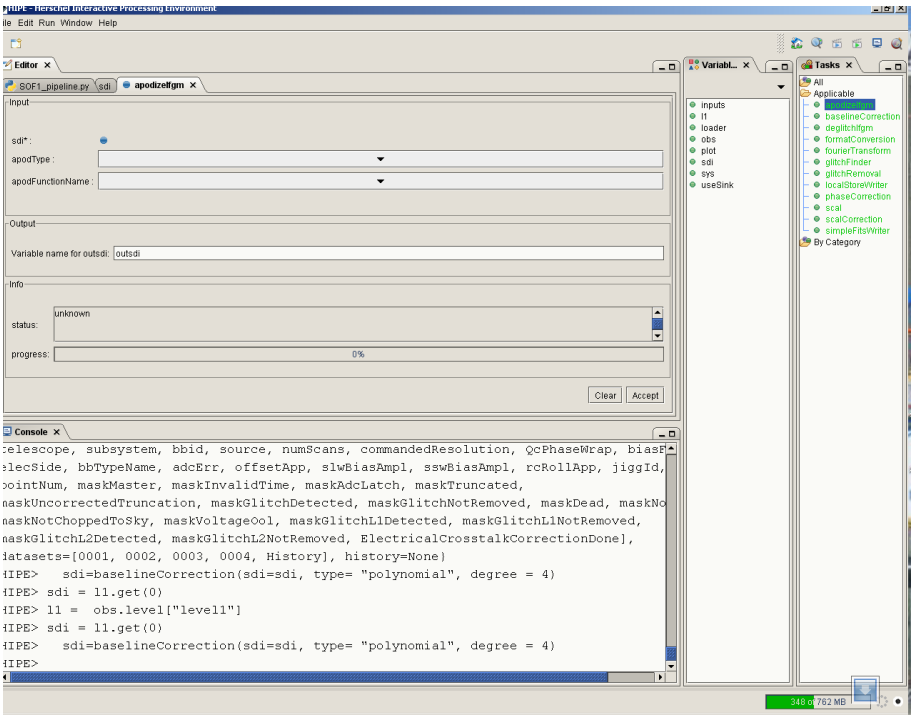


Figure 7.22. Task view for selecting apodizlgfm.

The apodType argument means that we are only going to apodize the symmetric portion. The apodFunctionName is the name of the apodization function that we select. Here this function is from the Norton-Beer family. All apodization functions have an adverse effect on spectral resolution; this one will decrease the resolution by a factor of 1.5 (that is where the 15 comes from.).

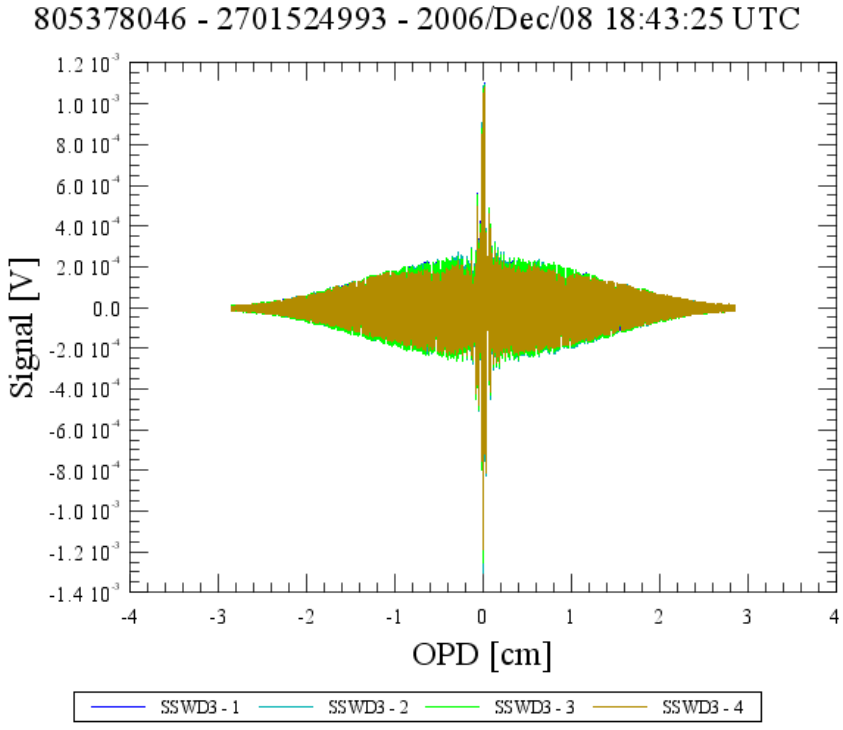


Figure 7.23. Apodization of the double-sided portion of an SSWD3 interferogram. Note that this interferogram contains only the signals from OPD positions symmetric about ZPD (OPD=0).

The next pipeline step is to transform the double-sided interferograms. We do this because we want to evaluate the phase (the phase being the arc tangent of the Imaginary part over the Real part of each spectra.)

```
dsds = fourierTransform(sdi=presdi, ftType="ds", IA=True)
```

Here, the side argument just tells the function to take the transform of the double sided interferograms and return a product whose signals will contain real and imaginary components. Using PixelViewer(), you can inspect these spectra and see the Real and Imaginary components.

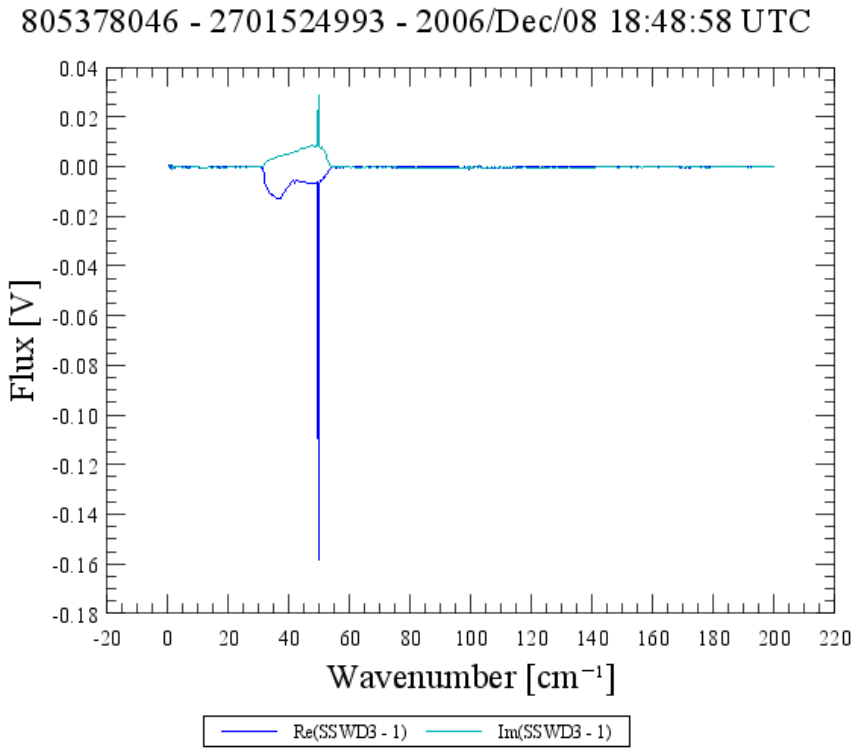


Figure 7.24. Spectra from the double-sided portion of an SSWD3 interferogram. Note the presence of real and imaginary components.

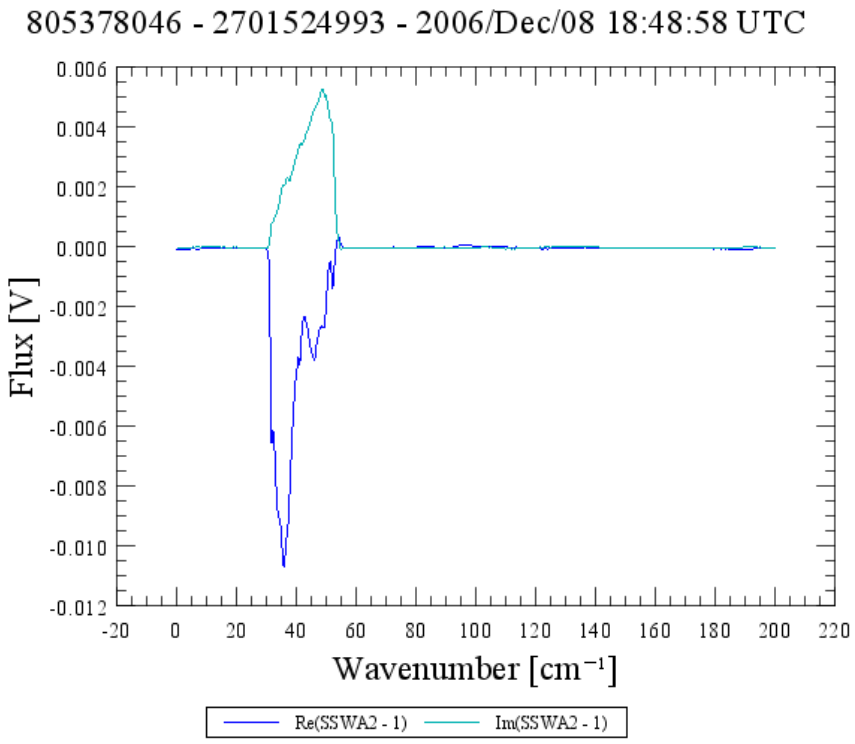


Figure 7.25. Spectra from the double-sided portion of an SSWA2 interferogram. Note the presence of real and imaginary components.

We then pass these spectra and the original interferograms to the PhaseCorrectionTask() module:

```
sdi = phaseCorrection(sdi=sdi, dsds, polyDegree=4,
pcfSize=127, obs.calibration.spec.bandEdge)
```

This module will make a fit to the in-band phase for each interferogram, and then correct the interferograms by convolution with the inverse transform of the fitted phase.

The two free parameters shown here, polyDegree and pcfSize, refer to the fitting and the convolution, respectively. polyDegree is the order of the polynomial used in the fitting of the phase, and pcfSize refers to the length of the convolution function.

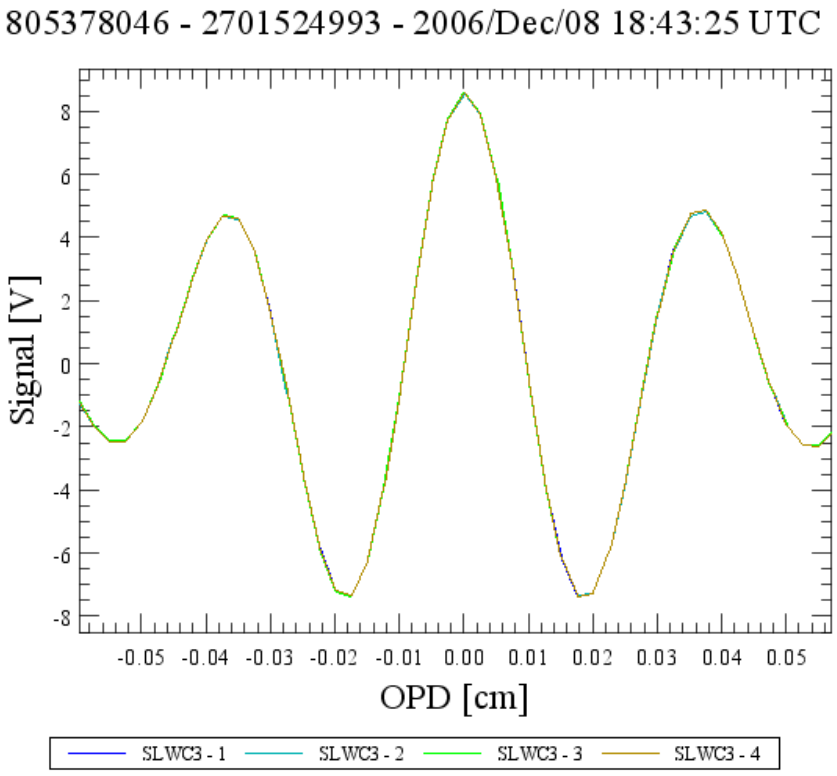


Figure 7.26. SLWC3 - after phase correction.

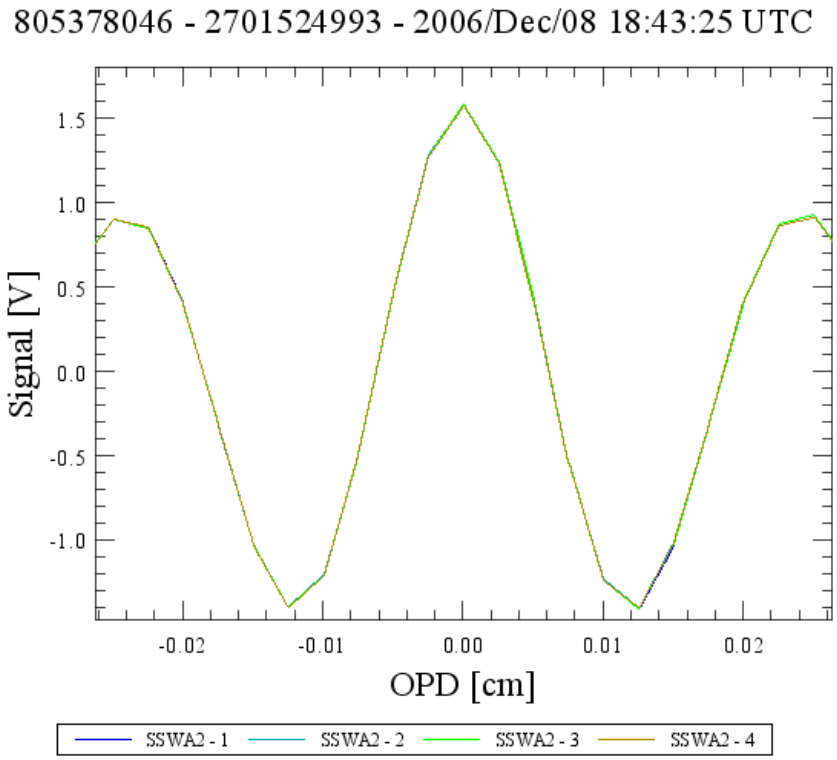


Figure 7.27. SSWA2 - after phase correction.

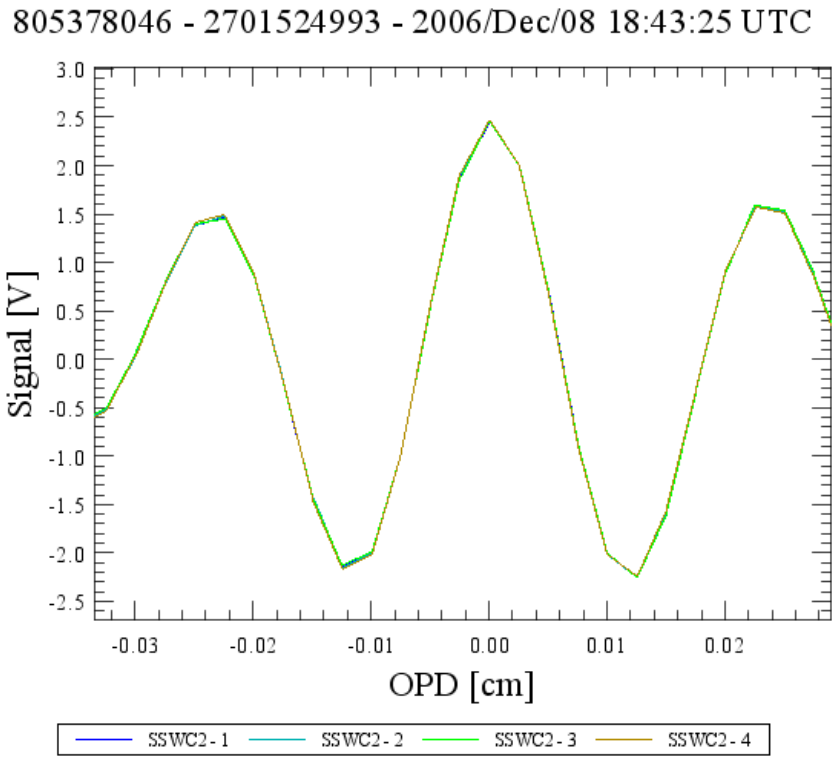


Figure 7.28. SSWC2 - after phase correction.

Inspect the corrected interferograms. You can zoom right in on the portion around ZPD (OPD=0). There you should see that the signals are now pretty much symmetric.

Now that phase-correction is complete, we can (optionally) apodize the interferograms. Here, we want the apodization to apply to the entire interferogram so we set apodType=”ss”.

One thing you might want to do is to create a copy of the interferogram product. Create the copy by un-commenting the lines:

```
unapodSdi = SpectrometerDetectorInterferogram(sdi)
```

Now, run the apodization:

```
sdi = apodizeIfgms(sdi=sdi,
apodType="ss", apodFunctionName="aNB_17")
```

Here we are using the Norton-Beer function that reduces resolution by a factor of 1.7.

We transform the interferograms to create high-resolution spectra. We tell the fourierTransformTask() module to do this by setting the ftType argument to “ss”.

```
ssds = fourierTransform(sdi=sdi, ftType="ss")
```

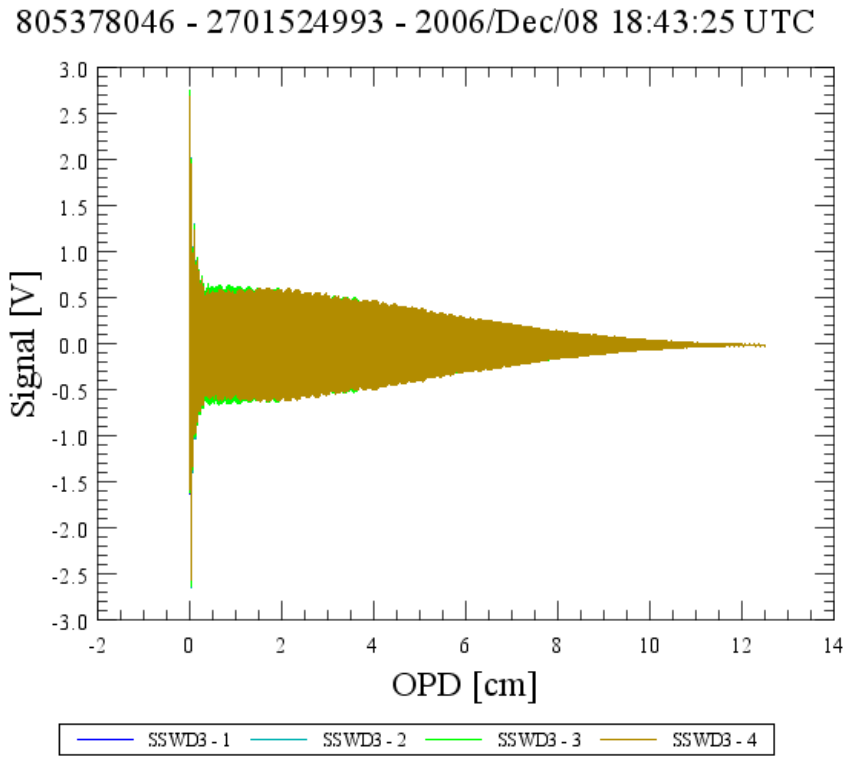


Figure 7.29. High resolution interferograms from SSWD3 after apodization.

Inspect the results. Again, look at SSWD3. Note too the difference in the spectra for this pixel between the apodized and unapodized (see how the ripples are decreased at the cost of broadening the line.)

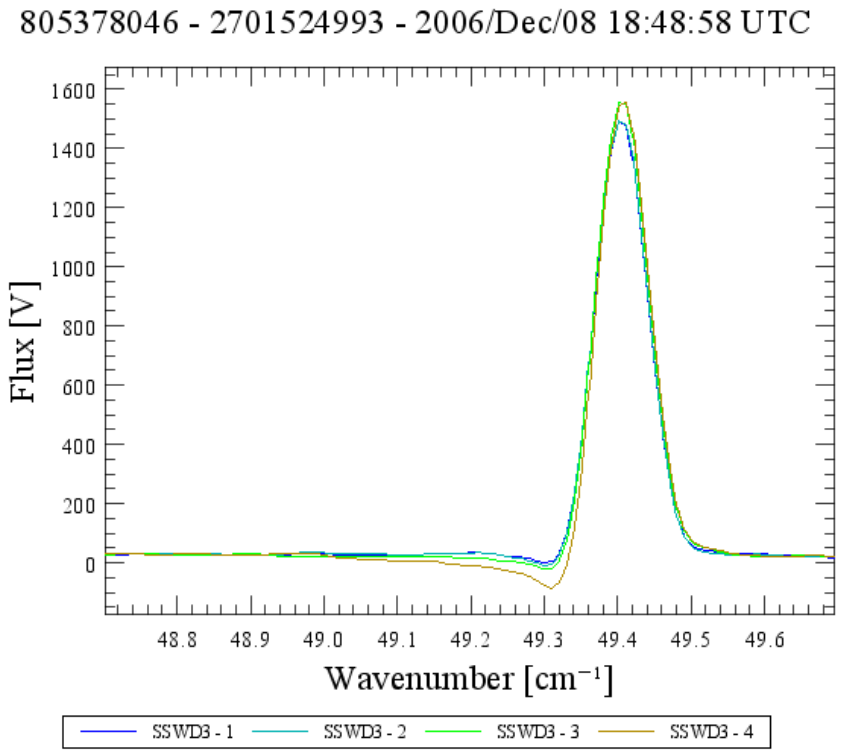


Figure 7.30. High resolution spectra for SSWD3. The Norton-Beer apodization function has been applied. A close up of the laser line region is shown here.

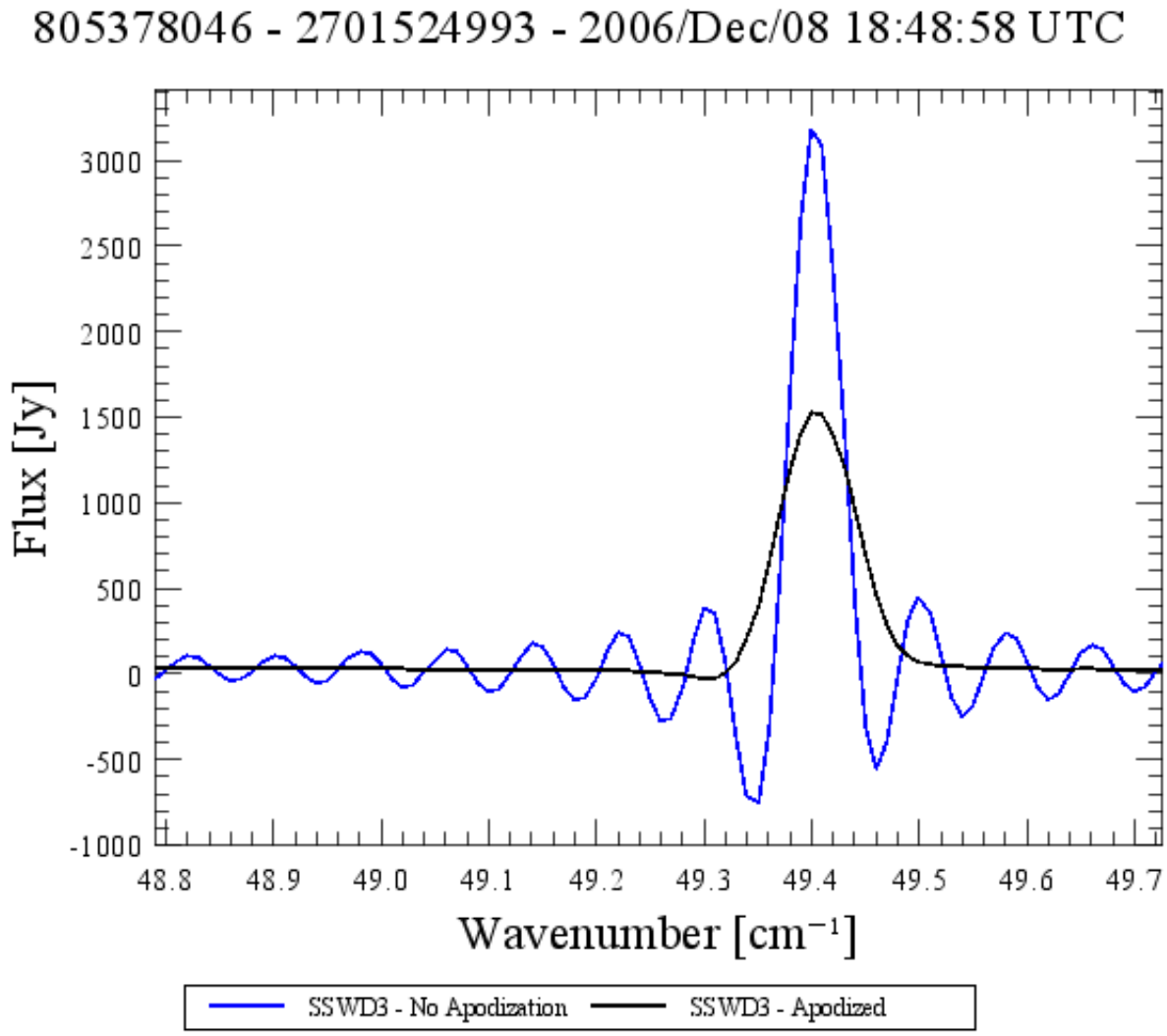


Figure 7.31. High resolution spectra for SSWD3. No apodization function has been applied. The ripples from the Sinc ILS are clearly visible.

You may have noticed that the data products to this point contain four scans each. In theory, these data are redundant as each is essentially the same observation – each is a scan of the spectrometer mechanism while the detectors view the same target. Theory not always being equal to practice, it is typical to perform more than one scan per spectrometer observation in order to increase the resultant signal-to-noise ratio.

All this leads in to the final step of the spectrometer pipeline; that of averaging the spectra that we just created.

The output product from this step contains a single spectrum per detector and that spectrum is the average of the each input spectra per detector per wavenumber bin. Example spectra from this product is shown in Figures 7.30 and 7.31.

7.1.3. Additional reading

We have include processing of only the photometry and spectroscopy pipeline scripts as illustrative examples - however, the processing of other photometer and spectroscopy AOTs will follow the same basic prescriptions. Further, additional information regarding the structure of data at the various levels

of processing, post-running of the respective pipeline scripts for each of the different photometry pipelines can be found in the SPIRE Pipeline Description document

Chapter 8. How to Save and Restore Data (including Herschel Archive data, ASCII and FITS)

Herschel Editorial Board

8.1. Introduction

Saving HCSS data and exporting it to a format which can be read by other tools outside HCSS is a very important task. It is necessary to have a correct understanding of the data structures in an object-oriented environment like the HCSS, because the data may come in different types of classes and hierarchies. The following drawing shows the general overview of the available high level data in HCSS:

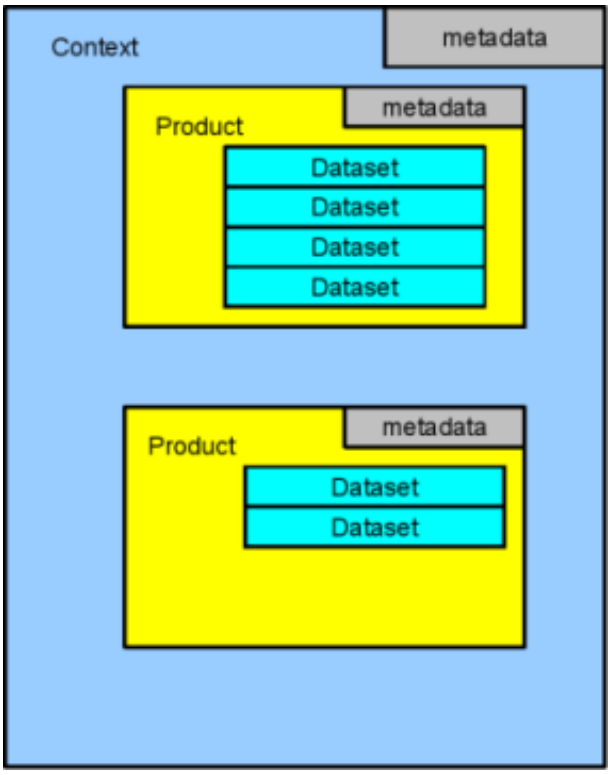


Figure 8.1. HCSS high level data hierarchy

In this scheme, the highest level data is the Context, which is a product that stores references to other products. So, the Context shown above in reality does not physically keep the two Products in it but instead it only keeps their references, also known as URNs.

The next level in the data hierarchy are Products, which may contain datasets of different types: ArrayDatasets, TableDatasets etc. Some examples of Products are the SimpleImage, which holds the images...

Note

How to save and restore variables of different kinds in a DP session is explained in Section 10.2 of the DP User's Manual.

In short:

```
save("myfile.sav","x,y,z") # x, y, z are variables that are
saved in a disk
                           # file called -"myfile.sav"
save("myfile.sav") # saves all variables in a DP session on disk
file
restore("myfile.sav")  # restores all variables to the session.
# WARNING: This file can get very large
# for many sets of data being accessed
# and saved from a DP session.
```

8.2. How to save and restore data from the command line

In order to illustrate the steps to save and restore data we need to create some HCSS high level data, like products and datasets.

First, let's create a product. An example of an HCSS product is the `SimpleImage` which is the product used to hold images, together with the optional mask, flag, exposure and errors images. It also includes necessary metadata in form of keywords (similar to FITS header keywords).

```
myImage = SimpleImage(description="An image",image = Double2d(50,100),\
        error=Double2d(50,100),exposure=Double2d(50,100))
```

Warning

The above code will generate an image with the value 50 assigned to the `NAXIS2` keyword and 100 assigned to `NAXIS1`. In other words, the image size will be 50 pixels along the *y* axis and 100 pixels along the *x* axis. The coordinate values will be displayed in this order (*y*, *x*) in the Image Viewer. For an explanation of why the *y* size comes *before* the *x* size, see section *A note on array ordering* in the *User's Manual*.

8.3. How to save and restore products using a Local Store

Note

Information in this section is covered in more detail in the PAL chapter of the "DP Basic User's Manual" available from within the Help environment of HIPE.

The easiest way to save and restore products in a HIPE session is to place them into a "local store". The "local store" is a simple folder structure which contains pools of products in the form of FITS files with corresponding metadata. Pool folders usually reside in the ".hcss/lstore" directory under the user's own home directory.

The "PAL storage manager" view in HIPE is shown in Figure 8.2. This allows the user to easily create pools, open a product storage and register a pool in it. More than one pool (directory of data) can be placed in a storage, if wanted, making many products from different observations or instruments available from the same storage area. The ID given to the storage is the one needed to directly access the products from that storage.

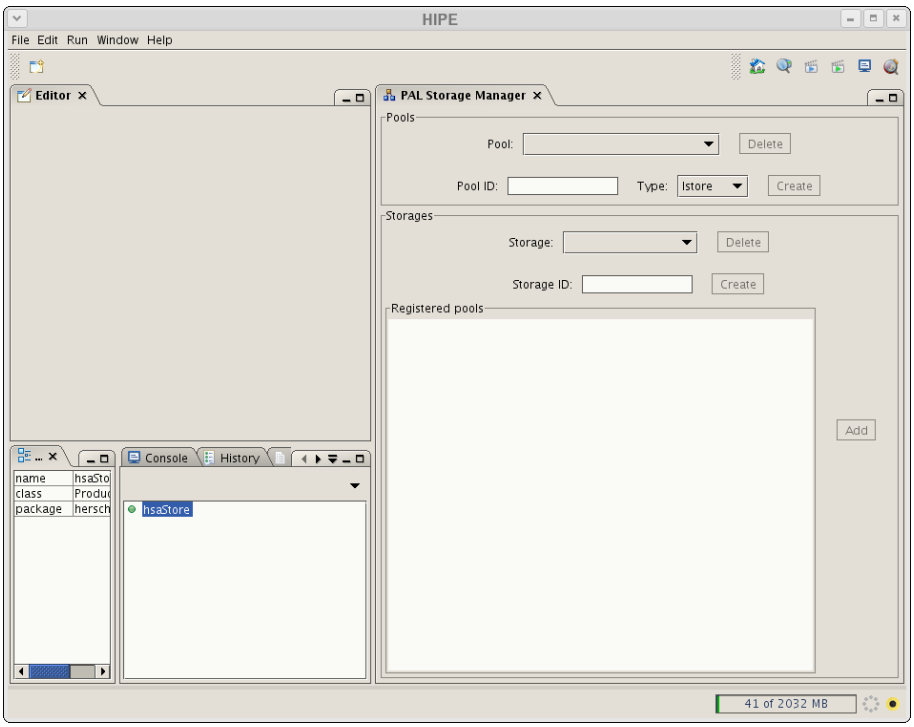


Figure 8.2. View for creating a pool for storing products

On the top panel are shown the list of user's pools under ".hcss/lstore" directory. New pools can be created fill-in Pool ID field and clicking the "Create" button. The action performed by the button is echoed in the HIPE Console (see Figure 8.3).

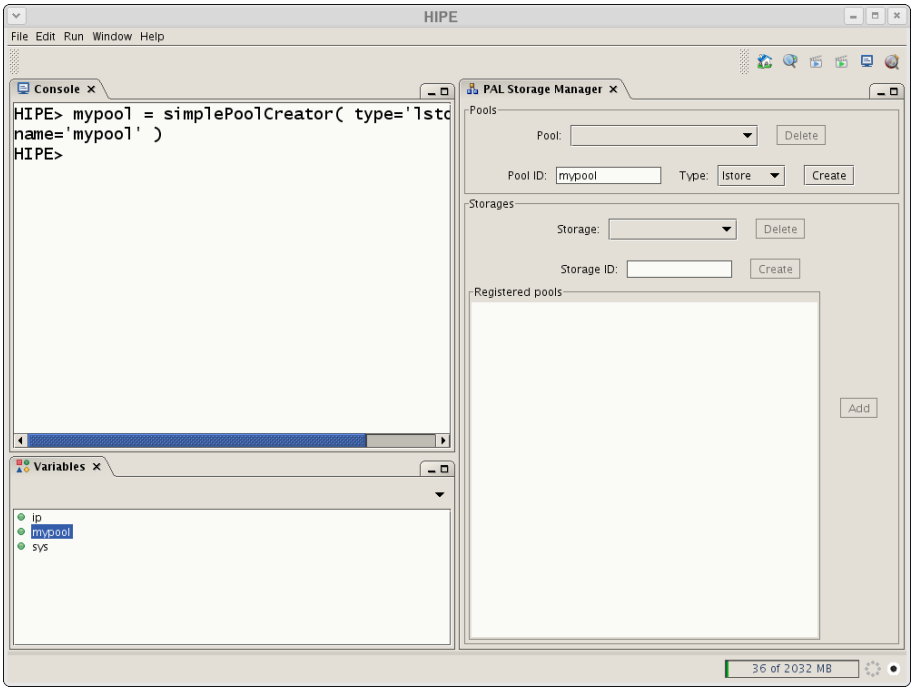


Figure 8.3. Listing available pools

Under panel Storages appear a number of pre-defined storages. New ones can be created (Figure 8.4). Moreover, pools can be registered on the storage by selecting the Storage and pressing "Add": A pop-up window will appear with the list of the available pools (Figure 8.5).

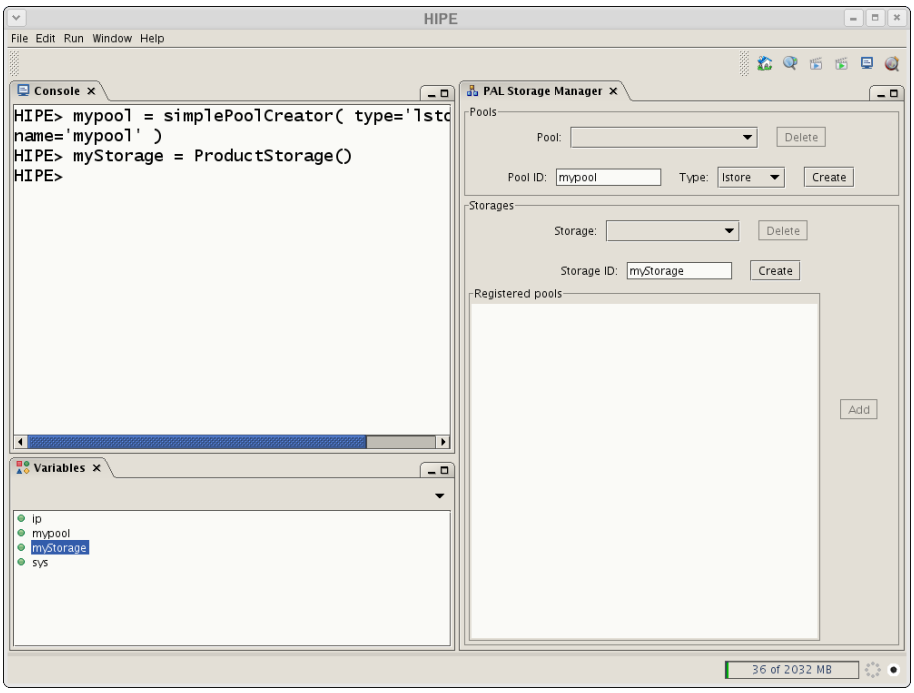


Figure 8.4. Adding new storages

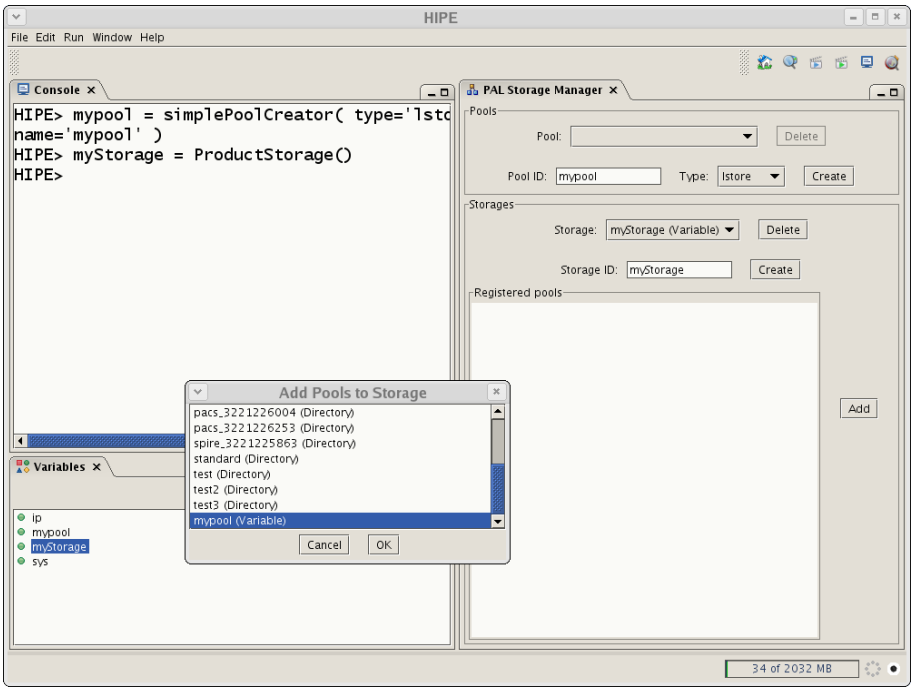


Figure 8.5. Adding already existing storages to the session

Saving data

In order to save a product into the local store we need to open a product storage, register a pool in it and then save the product into the pool. The first three items are discussed int he previous section. The following example also illustrates how this can be done on the command-line. The last line actually saves the data in the store.

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

```
store.save(myImage)
```

Now the product "myImage" is saved in the local store in the directory on local disk called ${HOME}/.hcss/lstore/myTestPool.

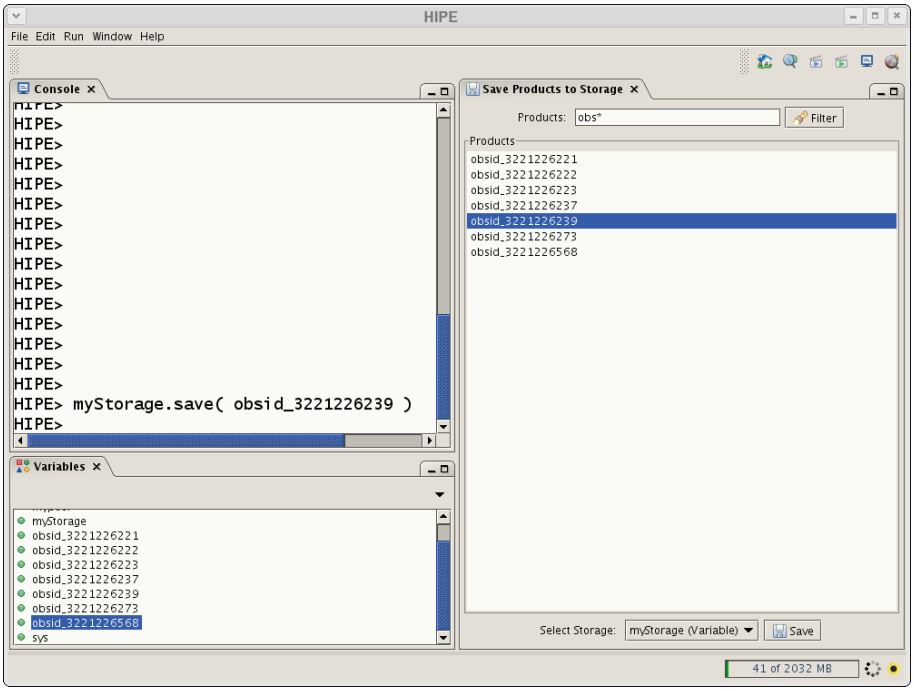


Figure 8.6. Save products to Storage view

Saving in HIPE can also simply be done using the "Save Products to Storage" view (see Figure 8.6), which allows to easily save products into a pool by simply selecting those in your DP seesion and the storage to use. Saving them in the storage occurs with the "Save" button at the bottom of the view.

Note that you can only save products, which means that if you want to save a Dataset of any kind - `TableDataset`, `Spectrum1d` or `2d` Dataset etc. you need to wrap them in a product as is shown in the following example

```
# create a TableDataset with two columns index and xvalue
table = TableDataset(description = -"A table")
table["index"] = Column(data=Int1d.range(100))
table["xvalue"] = Column(data=Double1d(100).apply(RandomUniform()))
```

Next we need to put in a product:

```
tProduct = Product(description="A table")
tProduct["myTable"] = table
store.save(tProduct)
```

Placing things into products allows for the proper header information to be included. Products can be wrapped within products (e.g., several images in a single product such as an observation) and each level has its own metadata/header information.

Restoring data using the productBrowser

Restoring the data back into HCSS is more complicated as it is necessary to know the product URN in the local store in order to retrieve it. However, a simple way to do this is using the productBrowser(). We can also use the Data Access view (*see also HowTo on accessing data*):

```
store = ProductStorage()
```

```
myPool = LocalStoreFactory.getStore("myTestPool")
store.register(myPool)
result = browseProduct(store)
```

You can then query the available products in "myPool", select the one you need, add it to the basket and then exit the `productBrowser`. The steps are shown in the following screenshot:

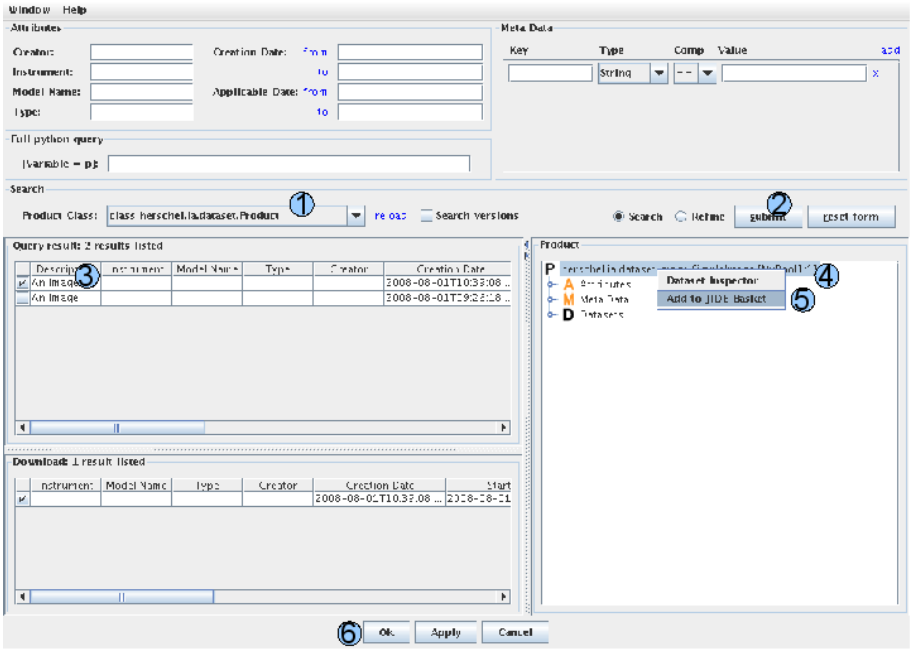


Figure 8.7. Restoring product from the local store using the `productBrowser`

These are the steps explained:

1. Select the product class to be of type Product
2. Click on "Submit" button to execute the search
3. Results for the products in myPool of type Product are shown in "Query result" view. Select the one you want.
4. The selected product structure appears in the "Product" view: Attributes, Metadata and Datasets are shown for this particular product.
5. Click with the right-hand mouse button on the product line (with the large "P" in front) opens a menu with "Dataset inspector" and "Add to JIDE Basket". Select the second item. The selected product will appear in the "Downloads" view.
6. Click "OK" to close the productBrowser().

At the end the reference to the product will be stored in the result variable and you can restore the SimpleImage following this example:

```
print result
# [urn:MyPool1:herschel.ia.dataset.image.SimpleImage:0]

image = result[0].product
```

If there is more than one result then we can refer to it with an index ([0] in the previous example).

The same way we can retrieve products which contain datasets (`TableDataset` or `ArrayDataset`) instead of `SimpleImage`.

Restoring data using command line queries

We can search the local store for products with a given attributes. For example, querying the local store pool "myPool" for products with description matching "An image":

```
query=MetaQuery(Product,"p","p.description=='An image'")
results2=store.select(query)
print results2
# [urn:MyPool1:herschel.ia.dataset.image.SimpleImage:0]

image = results2[0].product
```

The same as above, if there are more than one result then we can refer to it with the index.

8.4. How to Save Images and Tables as FITS files

It is possible to save and read using command-line input or task dialogs in HIPE. For all task dialogs it should be noted that the dialog appears in an Editor view window. To run the task via the dialog always hit the "Accept" button to bottom right in the dialog box.

8.4.1. Saving with a Task Dialog

The simplest way to save data is using the `simpleFitsWriter` task. It is required that the data (image/table/set of spectra) are wrapped up as a product. An example product is an observation itself. But we can wrap any dataset into a product so that the appropriate metadata (header information) and history is available. Clicking on the name of any product will show this task available in the Tasks under the Applicable Tasks folder. A double-click on the task (shown in green) brings up the simple dialog shown in Figure 8.8.

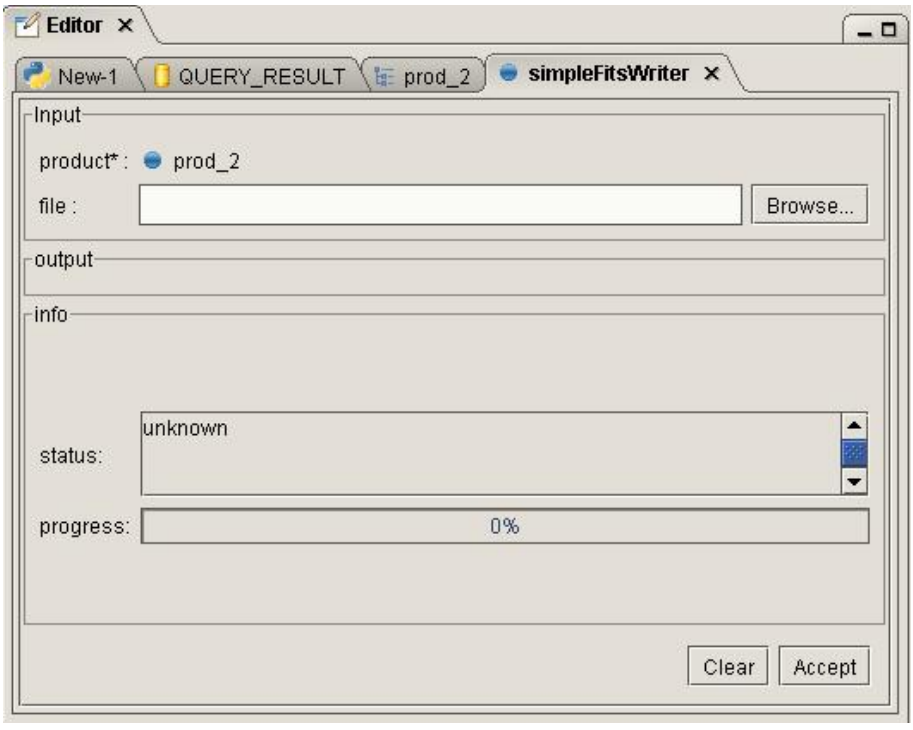


Figure 8.8. FITS save task dialog.

The only option the user needs to fill in is the name of the output FITS file. The default directory is the one that hipe was started from, so the full path name is usually required. Hitting the "Accept" button runs the task.

8.4.2. Saving Using Command-line Inputs

The FITS reader and writer in HCSS is in the FitsArchive package. First, let's store the product "myImage" from our previous example in a FITS file.

```
fits=FitsArchive()
fits.save("testFits-file1.fits",myImage)
```

The file "testFits-file1.fits" will be saved in the folder from where you started up HIPE. Otherwise, the full directory path should be supplied. It is a multi-extension FITS file with all the content of the SimpleImage product. Here is the structure of the saved FITS file:

```
 No. Type     EXTNAME      BITPIX Dimensions(columns)

   0  PRIMARY                 32     0
   1  IMAGE    image         -64     100 100
   2  IMAGE    flag           16     100 100
   3  IMAGE    error         -64     100 100
   4  IMAGE    exposure      -64     100 100
```

8.4.3. How to Save TableDatasets as FITS Files

Once we have the TableDataset wrapped in a Product we can save it like all other products. We can use the same FITS writing task from HIPE as noted above, or we can use a command-line method. For example:

```
     fits=FitsArchive()
     myTable = TableDataset() # create an empty table
     myTable["X values"] = Column(Double1d([2,3.4,4])) # create fake column
     myTable["Y values"] = Column(Double1d([2,4.5,4.8])) # create 2nd column
     tProduct = Product(description="This is a table") # create the product
     tProduct["firstTable"] = myTable  # add in the table and give it a label
     fits.save("testFits-file2.fits",tProduct)
```

The resulting structure of the saved FITS file is:

```
  No. Type    EXTNAME      BITPIX   Dimensions(columns)

  0  PRIMARY                 32     0
  1  BINTABLE table           8     2(3)

Column Name                      Format     Dims       Units     TLMIN  TLMAX
     1 X values                  1D
     2 Y values                  1D
```

We can see that the column names, which we named as "X values" and "Y values" are in the file.

8.4.4. How to Read FITS Files

The `simpleFitsReader` task allows FITS files to be read in. Two types of FITS readers are available -- for HCSS FITS and Standard FITS. You can let the software choose the appropriate one or choose a specific reader (see Figure 8.9).

To run the command from the HIPE dialog, go to the "Tasks" view -- select the "All" tasks folder and scroll down to `simpleFitsReader`. A double-click on the name brings up the dialog. Once a name is input and the FITS reader chosen, click the "Accept" button to run the task and read in the FITS file.

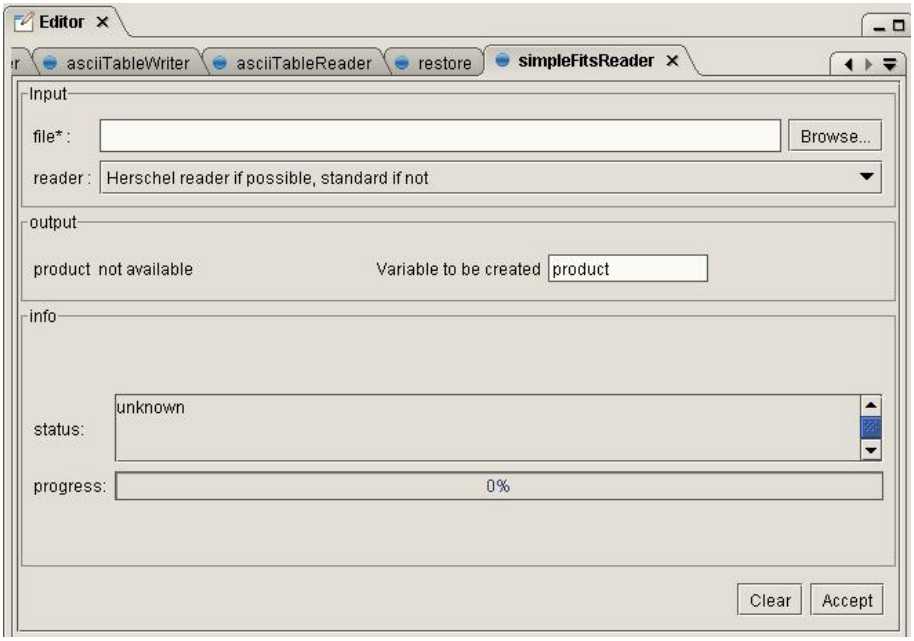


Figure 8.9. FITS read task dialog.

8.5. How to Create and Read ASCII Table Files

In this case it is not necessary to put the TableDataset in a Product and we can directly save the Dataset to an ASCII file. As for FITS writing, we can do this from within a HIPE task dialog or from the command-line.

8.5.1. Using HIPE Task Dialogs to Create and Read ASCII Tables

If we click on a variable that is a TableDataset -- such as "myTable" in the example above -- in the Variables view of HIPE, then we see that an Applicable Task in the Tasks view window is `asciiTableWriter`. Double-clicking on this task brings up a dialog for creating an ASCII table. The simplest way of formulating an ASCII table is to take the defaults and simply fill in a name for the output table. But more sophisticated options are available (see Figure 8.10).

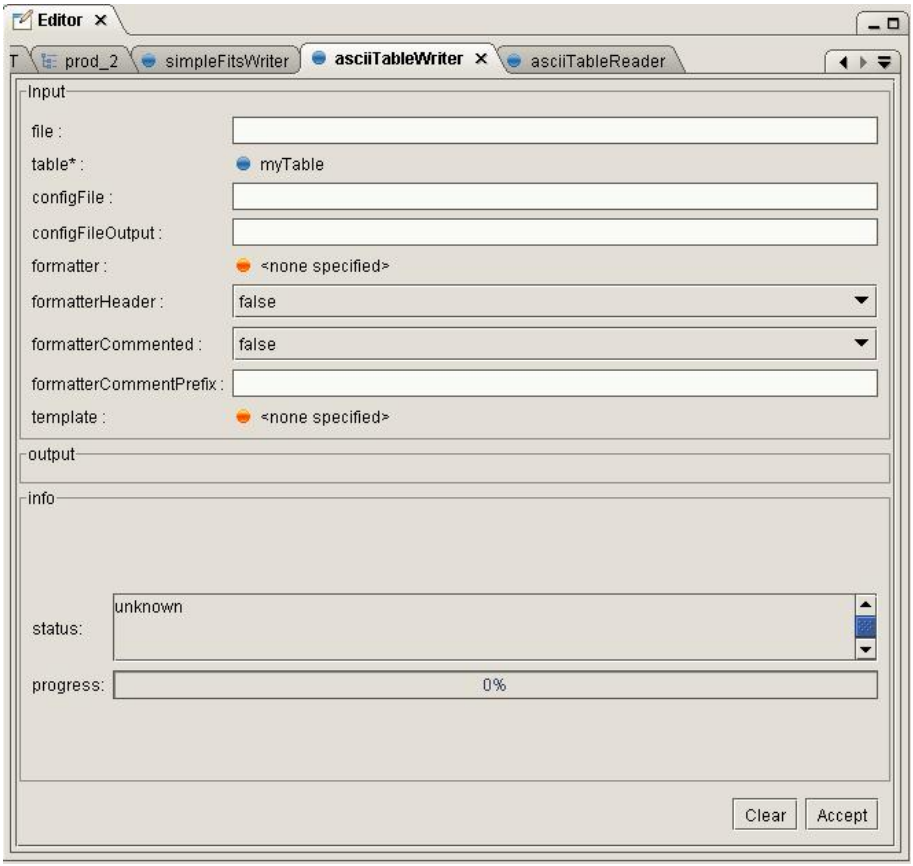


Figure 8.10. FITS save task dialog.

The other possible inputs for the task are the following (this information is also available by hovering the mouse over the parameters shown in the dialog).

```
* file = output file name.
* table = TableDataset to write.
* configFile  = configuration file where the formatter
(AsciiFormatter), parser (AsciiParser) and table template
(TableTemplate) must be specified. When configFile parameter is specified,
any parameter related to parser or to table template are not allowed.
* configFileOutput = if a config file is specified, an output configuration
file will be created.
* formatter (default AsciiTableTool formatter) = AsciiFormatter object.
* formatterHeader (default AsciiFormatter header allowed) = Specifies
if header information to be provided (true/false).
* formatterCommented (default AsciiFormatter comments allowed) = Specifies
if there are comments when writing a file (true/false).
* formatterCommentPrefix (default AsciiFormatter comments prefix value) =
Specifies what the prefix is for identifying all comments.
* template (INPUT, default value: extracted from the first file rows) =
TableTemplate object for specifying the data structure (see DP Basic User's
Manual for more details).
```

Clicking on "Accept" at the bottom of the task dialog window runs the task and creates an ASCII table.

Reading an ASCII table into HIPE can be done using the `asciiTableReader`. Go to the "Tasks" view and open the folder "All". Double-click on the word `asciiTableReader`. This provides a dialog. For standard CVS tables the only thing that needs to be filled in is the file name of the ASCII table to be read in. More

```
* file = input file containing ASCII table.
* table = TableDataset object name for loaded table.
* configFile = configuration file where the formatter (AsciiFormatter),
```

```
 parser (AsciiParser) and table template (TableTemplate) must be specified.
 When configFile parameter is specified, any parameter related to parser or
 to table template are not allowed.
 * configFileOutput = if a file is specified, an output configuration
 file will be created.
 * parser (default AsciiTableTool parser) = AsciiParser object.
 * parserIgnore (default AsciiParser ignore value)
 = String expression to ignore when parsing a file.
 * parserSkip (default AsciiParser skipping rows value) = Number of rows to
 skip when reading a file.
 * parserTrim (default AsciiParser trim rows value) = Specifies if the parser
 must trim each row when reading a file (true/false).
 * parserGuess (default value AsciiParser.GUESS_NONE) = specifies if
 the parser should guess column types. Files should not contain HCSS header
 (use skip=AsciiReader.HCSS_HEADER for skipping HCSS header or comment these
 lines)

   Valid options:
       o AsciiParser.GUESS_NONE: (default) file must contain template
or template must be provided (no guess)
       o AsciiParser.GUESS_TRY: guess types based on the first 100 records
       o AsciiParser.GUESS_ALL: guess types based on all records
       o AsciiParser.ALL_STRING: each record is a string (no guess required)
       o AsciiParser.ALL_BOOLEAN: each record is a boolean (no guess required)
       o AsciiParser.ALL_BYTE: each record is a byte (no guess required)
       o AsciiParser.ALL_INTEGER: each record is an integer (no guess required)
       o AsciiParser.ALL_LONG: each record is a long (no guess required)
       o AsciiParser.ALL_FLOAT: each record is a float (no guess required)
       o AsciiParser.ALL_DOUBLE: each record is a double (no guess required)
       o AsciiParser.ALL_COMPLEX: each record is a complex (no guess required)
 * parserDelim (INPUT, default value: comma) = Specifies the field delimiter.
 If it is one character, a csvParser is selected. If it is an expression,
 a RegExpParser (regular expression) is selected.
 * template (INPUT, default value: extracted from the first file rows) =
 TableTemplate object for specifying the data structure. See TableTemplate.
```

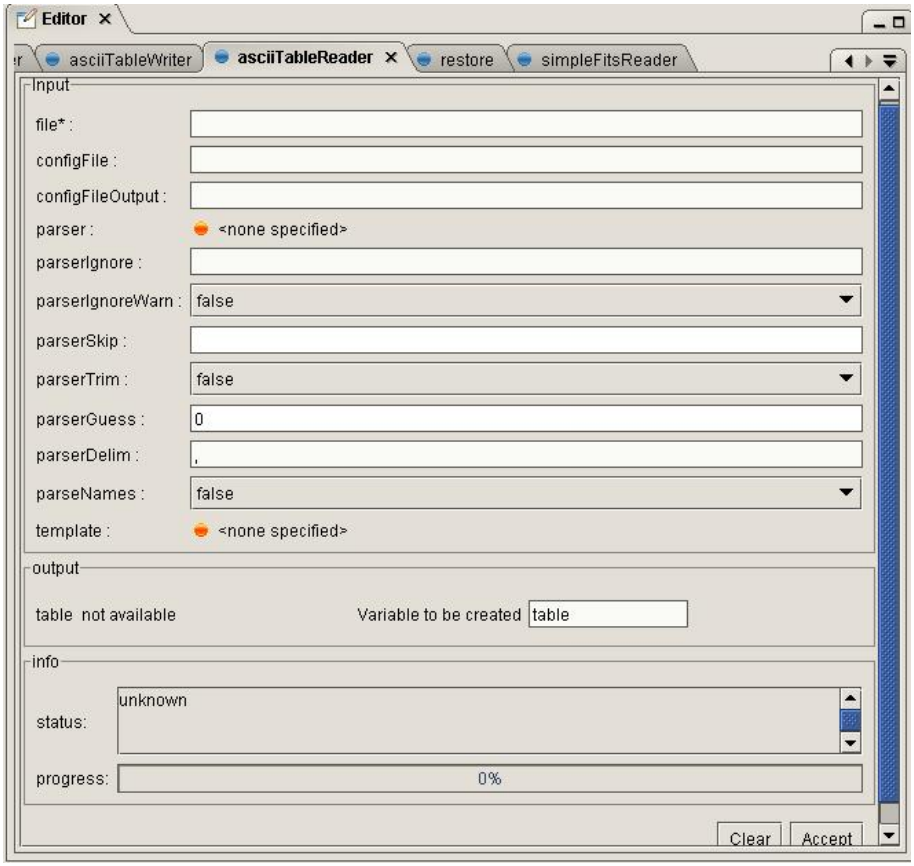


Figure 8.11. FITS read task dialog.

8.5.2. Using Command-line Input to Create and Read ASCII Tables

We can also do simple ASCII table creation from command-line inputs.

```
ascii = AsciiTableTool()
ascii.save("testAscii-file1.txt",table)
```

To read the table in again we need to "load" it using the same tool.

```
ascii = AsciiTableTool()
table=ascii.load("testAscii-file1.txt")
```

Loading uses the same parser/formatter (see below for how this may be changed) as is applied for saving.

By default the table is saved as a coma-separated-value file with 4 header lines, for example

```
X values,Y values      # column names
Double,Double          # column data types
,                      # column data units
,                      # description of the column
2.0,2.0  # the data start from this line
3.4,4.5
4.0,4.8
```

The default output delimiter can be changed to another symbol, like is shown in the following example:

```
ascii.formatter = CsvFormatter(delimiter = -'*')
ascii.save("testAscii-file2.txt",table)
```

Or the columns at output can have a fixed width using the FixedWidthFormatter with an indication of the column widths given.

```
ascii.formatter = FixedWidthFormatter(sizes=[8,12])
ascii.save("testAscii-file3.txt",table)
```

More information on creating and reading tables is available in the DP Basic User's Manual.

Chapter 9. How to plot in HIPE

9.1. Introduction

This chapter provides several examples that show you how to create and customize plots from the command line and from the HIPE graphical interface.

For an extensive description of the plot package, including its structure and philosophy, see the *User's Manual*: Chapter 6.

9.2. Simple plots from the command line

In order to illustrate the steps to produce simple plots we need an input x and y variables:

```
x = Double1d.range(11)
y = x*x
```

1. Simple plot:

```
from herschel.ia.gui.plot import *

plot = PlotXY()
plot.autoBoxAxes=1
layer = LayerXY(x,y)
plot.addLayer(layer)
```

2. Overplot a second x and y dataset

```
x1 = 10.0*Double1d.range(11)/10.0 -- 5.0
y1 = x1**3.0
```

Note that we do not need to repeat all plotting commands from the above example, we simply add a new layer

```
layer2 = LayerXY(x1,y1)
plot.addLayer(layer2)
```

And we note that the axis ranges are expanded correspondingly and that the new layer is with a different colour.

3. Change the plot title and subtitle

```
plot.title.text="Example plot"
plot.subtitle.text="two layers"
```

or if you don't want to have plot title and subtitle you can switch them off

```
plot.title.setVisible(0)
plot.subtitle.setVisible(0)
```

4. Change the axis labels:

```
plot.xaxis.title.text="X-values"
plot.yaxis.title.text="Y-values"
```

5. Change the axis ranges

```
plot.xaxis.setRange([-2.0,2.0])
plot.yaxis.setRange([-10.0,10.0])
```

or go back to the auto range

```
plot.xaxis.setAutoRange(1)
plot.yaxis.setAutoRange(1)
```

6. Change the tick marks spacing and then the number of minor tick marks

```
plot.xaxis.getTick().setInterval(3.0)
plot.yaxis.getTick().setInterval(30.0)
```

and to have 5 minor tick intervals between the major tick marks (which means 4 minor ticks)

```
plot.xaxis.getTick().setMinorNumber(4)
plot.yaxis.getTick().setMinorNumber(4)
```

7. Draw grid lines

```
plot.xaxis.getTick().setGridLines(1)
plot.yaxis.getTick().setGridLines(1)
```

Note that the grid lines are drawn at the major tick marks.

8. Change the axis from linear to log

```
plot.xaxis.setType(Axis.LOG)
plot.xaxis.setType(Axis.LINEAR)
```

Warning

The axis ranges need to be positive otherwise values are ignored in the LOG plot. When returning the plot back to LINEAR, all points are made plotted again even if some had been dropped in the LOG plot.

9. Change the line style for a given layer

```
layer.setLine(Style.NONE)
```

The line styles for setLine() can be

- Style.NONE - symbols only
- Style.MARKED - symbols connected with lines
- Style.SOLID - solid line, no symbols
- Style.DASHED - dashed lines
- Style.MARK_DASHED - symbols connected with dashed lines

  Note that in the MARKED styles the default plotting symbol is used

10. Change the plotting symbol and its size. In order to have an effect you need to change the line style first to be one of NONE or MARKED styles

```
layer.setLine(Style.NONE)
layer.setSymbol(Style.FSQUARE)
layer.setSymbolSize(10)
```

The symbols can be:

Table 9.1. Symbols codes

DOT = 1	a dot	VCROSS = 2	a "+" sign
DCROSS = 3	an "x" sign	VDCROSS = 4	a "+" + "x" sign
CIRCLE = 5	an empty circle	TRIANGLE = 6	an empty triangle
UTRIANGLE = 7	an empty upside-down triangle	SQUARE = 8	an empty square
SQUARE_CROSS=9	an empty square + "x"	DIAMOND = 10	an empty diamond
DIAMOND_CROSS=11	a diamond + "+"	OCTAGON=12	an empty octagon
STAR = 13	an empty star	FCIRCLE=14	a filled circle
FTRIANGLE=15	a filled triangle	FSQUARE = 16	a filled square
FDIAMOND=17	a filled diamond	FOCTAGON=18	a filled octagon
UARROW = 19	an up arrow	DARROW = 20	a down arrow
RARROW=21	a right arrow	LARROW = 22	a left arrow
DARROW_LARGE=23	a large down arrow	UARROW_TRIANGLE = 24	a large up triangular arrow
DARROW_TRIANGLE = 25	a large down triangular arrow		

Note

You can use either the code or the numeric value for the symbol, that is, setSymbol(Style.FSQUARE) is equivalent to setSymbol(16).

11. Change the colour of the symbols and lines for a given layer

```
layer.setColor(java.awt.Color.RED)
```

12. Show or remove the legend for the layers

```
plot.setLegendVisible(1)
```

and we can also remove itt

```
plot.setLegendVisible(0)
```

13. We can also change the legend name for a given layer

```
layer.setName("Test 1")
```

and we can also remove the legend for a particular layer if we don't want it to appear on the plot

```
layer.setInLegend(0)
```

14. Histogram mode. You need to be in MARKED or SOLID line style for this mode to work:

```
layer.setLine(Style.MARKED)
layer.style.setChartType(Style.HISTOGRAM)
```

The chart type can be HISTOGRAM - the data point is in the middle of the histogram horizontal bar, HISTOGRAM_EDGE - the data point is on the edge of the histogram horizontal, LINECHART - the data points are connected with lines.

15. Add error bars to x and/or y values. First we need to create arrays with errors

```
xerr = SQRT(x)
yerr = SQRT(y)

layer.setErrorX(xerr,xerr)
layer.setErrorY(yerr,yerr)
```

Note that the upper (the first argument to setError() method) and the lower (the second argument to setError() method) error limits can be different.

16. Add an annotation

```
layer.setAnnotation(0,Annotation(6.5,-10,"Test",color=java.awt.Color.GREEN))
```

17. You can use math and special symbols for text labels in your plot. It is possible to use TeX-like formatting of strings. In particular, entering math mode using a `$` symbol it is possible to insert Greek characters, e.g. using `\\alpha` or `\\beta`. Superscripts are preceded by the `^` symbol and subscripts by the _ symbol. For example the following can be used to set the title of the *x* axis:

```
plot.xaxis.title.text="$A_{1.3}^{b-3/2}$"
plot.xaxis.title.text="$\\alpha_{1.3}^{\\beta-3/2}$"
```

Note that it is necessary to use "\\" to escape the "\" symbol *from the command line*. A single backslash should be used in the *Property Panel* window instead.

Warning

Not all special symbols are available. If the symbol is not available it will be treated as normal text by the interpreter. For example, `$\\Alpha$` will be rendered as *\Alpha*.

The available special symbols are the following:

- All the lower-case Greek letters.
- The following upper-case Greek letters: `\Gamma, \Delta, \Theta, \Lambda, \Xi, \Pi, \Sigma, \Upsilon, \Phi, \Psi, \Omega`.
- The `\angstrom` and `\micro` symbols.

To insert other symbols you can use the Unicode escape sequence `\uxxxx`, where `xxxx` is the hexadecimal code of the symbol. For example, `\u2299` corresponds to the *circle dot operator*, which can also be used as symbol for the Sun.

For a list of Unicode sequences see for example http://www.utf8-chartable.de/.

18. Change the plot window size. You can resize the window with the mouse or you can specify the desired window size once you have added layers to the plot

```
plot.setWidth(400)
plot.setHeight(300)
```

19. Save plot in a file

```
plot.saveAsJPG("myfile.jpg") # JPEG format
plot.saveAsEPS("myfile.eps") # Encapsulated PS
plot.saveAsPNG("myfile.png") # PNG format
```

9.3. Interacting with plots using plot properties GUI

Once you have done steps 1., 2. and/or 3. from the above then most of the following interactions with the plot properties, i.e. steps from 4. on from the previous section, can be done via the plot properties GUI. To open up the plot properties GUI you need to click with the right-hand mouse button and choose "Properties..." entry in the menu. This will bring up the following window:

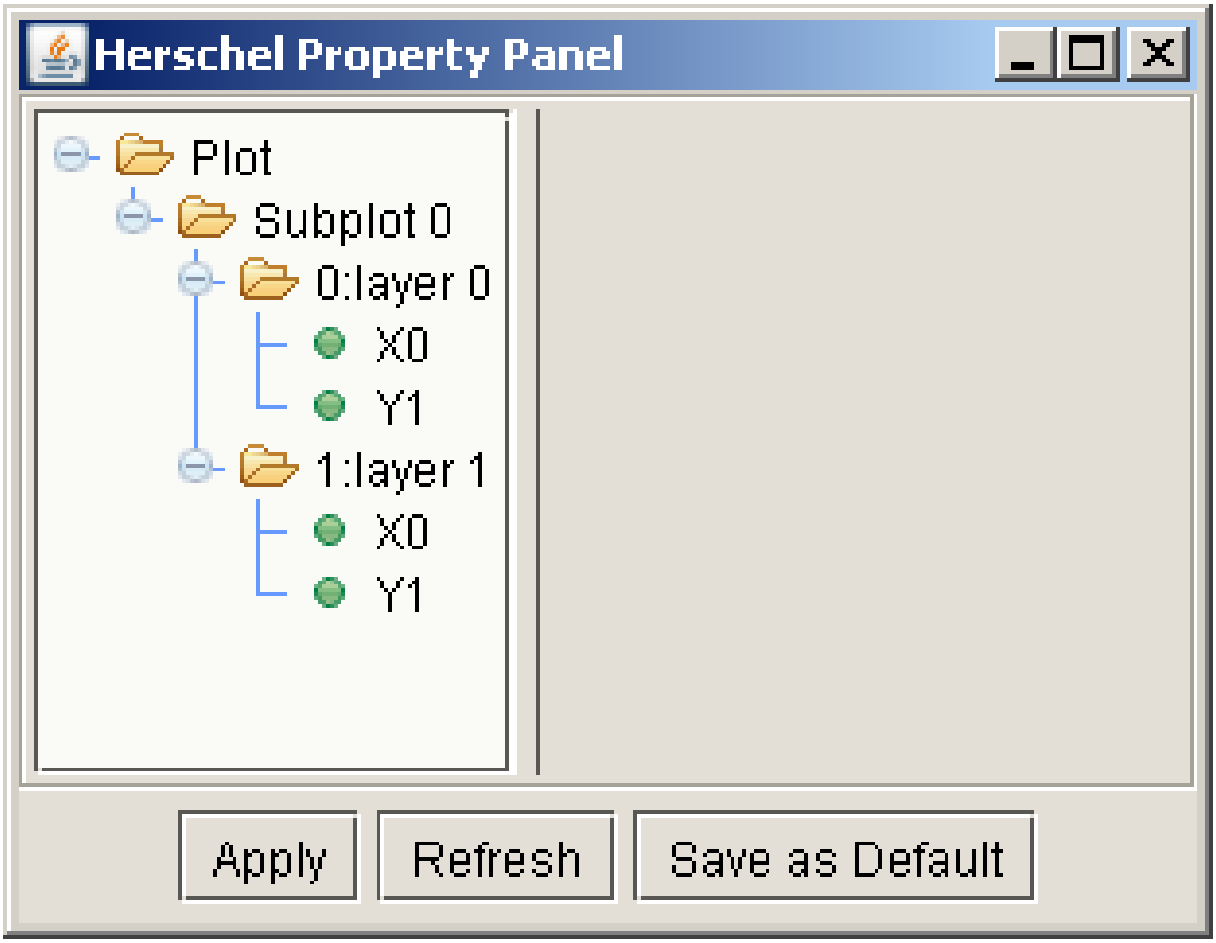


Figure 9.1. Plot properties initial windows.

On the left frame of the window we can see the hierarchical structure of the plot we created following steps 1., 2. and 3. from the above example: we have the root element our plot and we have two layers added on top of it. Each layer has also sub-objects for the x and y axes. Navigating to each of these elements we can change the properties for each of the items.

1. First, let's enter the Plot properties:

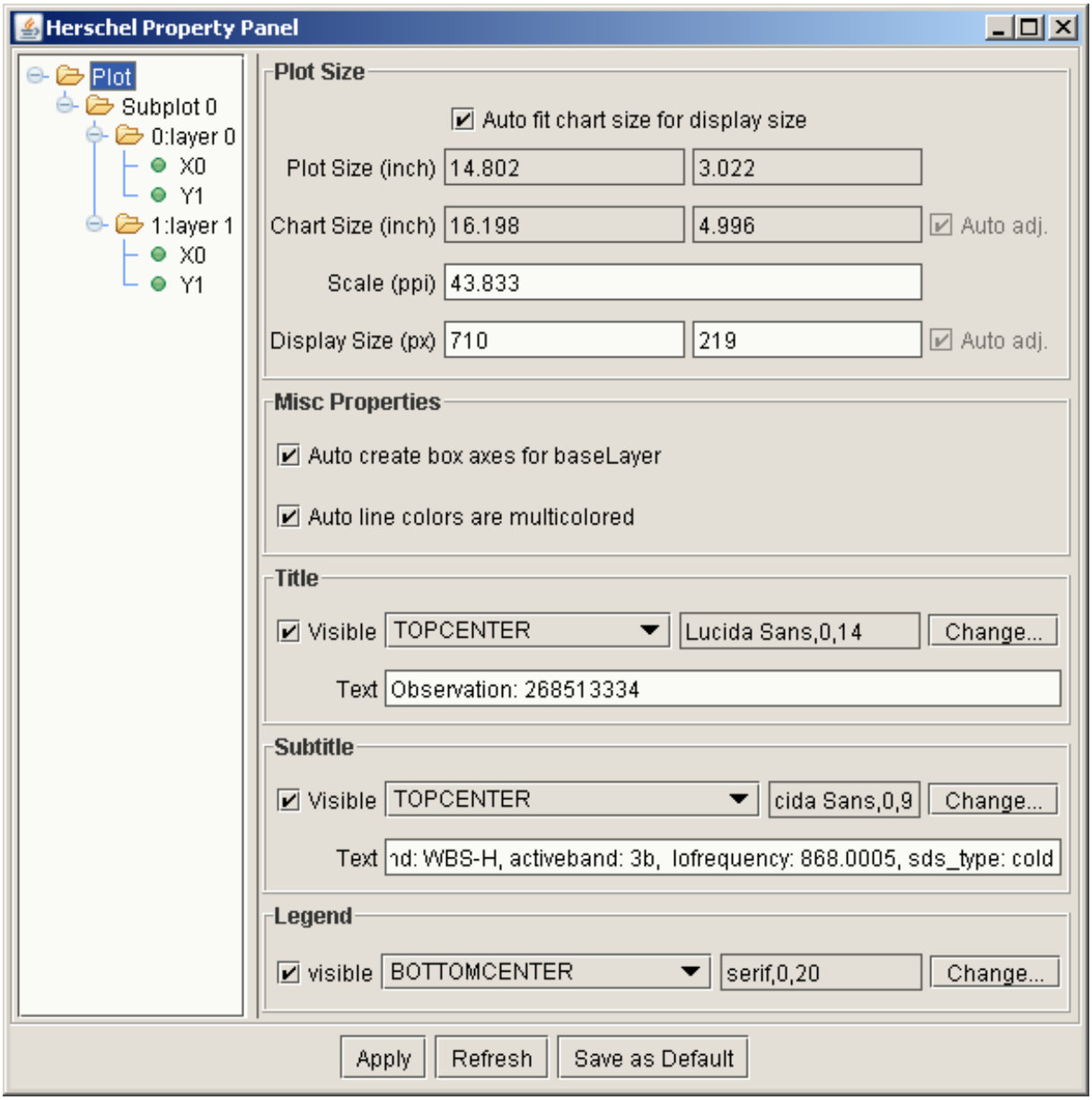


Figure 9.2. Plot properties items

The different base plot entries are self-explanatory and they can be changed interactively and applied. Also, if you plan to reuse some of them for all subsequent plots you may save them as default.

2. Layers properties

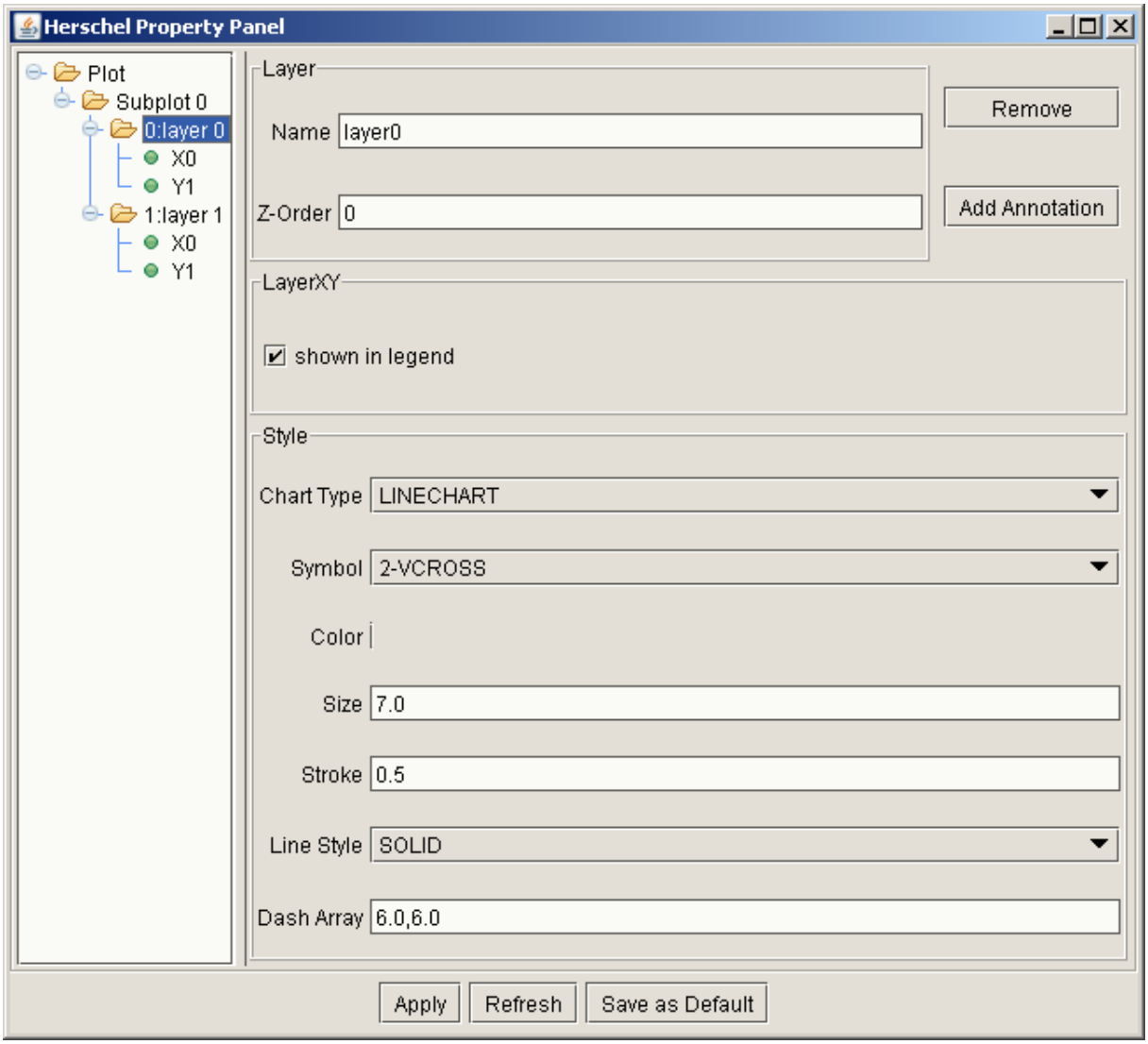


Figure 9.3. Layer's properties

These are the property entries at a layer level, they are the same for all layers. From this panel you can also add annotations to the plot. Note that this annotation is attached to the corresponding layer so any change in layer colour will affect the annotation as well.

3. Axes properties

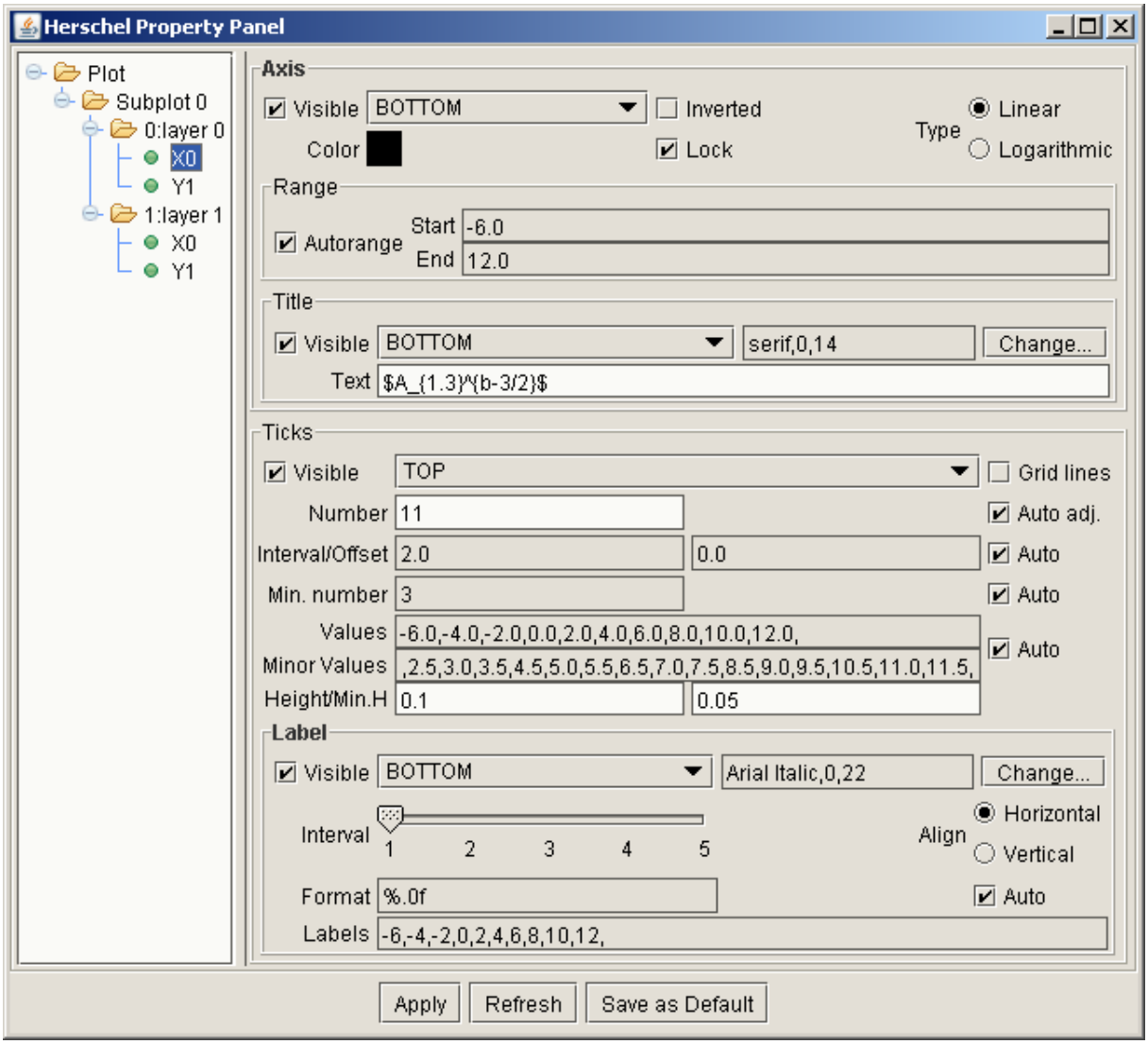


Figure 9.4. X-axis properties

Here you have almost complete control over the axis properties: range, title, tick marks and minor tick marks, axis label. And different radio buttons allow you to turn on/off auto features.

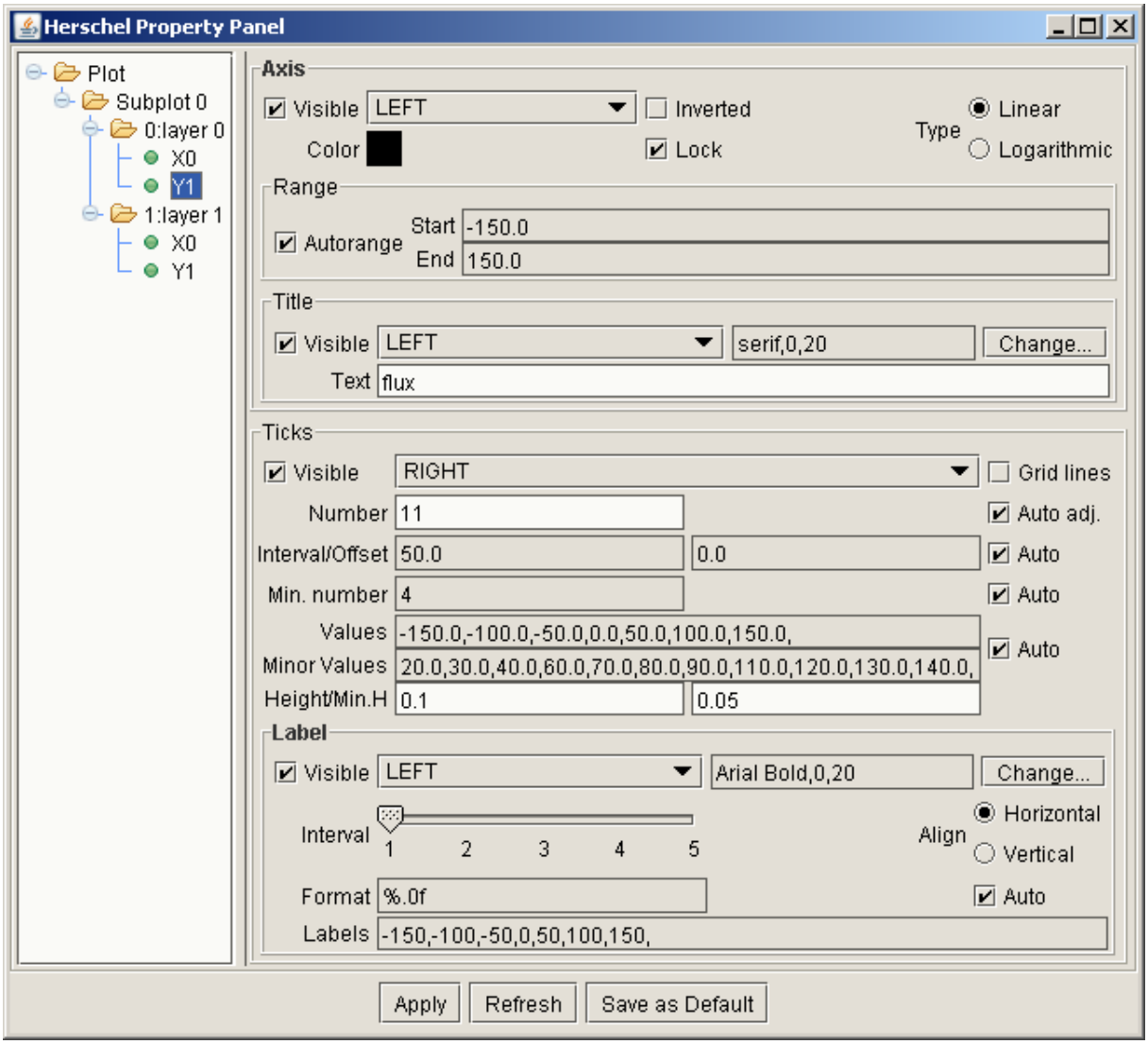


Figure 9.5. Y-axis properties

4. Printing of a plot. In the menu which pops up when you click with the right-hand side mouse button you have "Print..." menu which allows you to send the plot directly to a printer (if you have configured one for your system).

5. Saving the plot. In the menu which pops up when you click with the right-hand side mouse button you have "Save as..." menu which allows you to save the plot in different image formats: Encapsulated PostScript file (EPS), JPG or PNG files.

Note

For plots, layers, annotations or axis titles when using the TeX notation you should not escape the "\" symbol, that is you can directly use α in the text field of the GUI.

9.4. Advanced plotting

Here we introduce some more advanced plotting. Most of these are explained in greater detail and with examples in the *User's Manual*: Chapter 6.

1. Multiple plots per window.

When we add layers to the plot we can specify their position on a grid as in the example below which places 4 layers onto a 2x2 grid (running indeces from 0,0 to 1,1).

```
plot = PlotXY()
layer = LayerXY(x,y)
layer1 = LayerXY(x1,y1)
layer1x = LayerXY(x1,y1/5.0)
layer1y = LayerXY(x1/5.0,y1)
plot.addLayer(layer,0,0) # top left
plot.addLayer(layer,0,1) # top right
plot.addLayer(layer,1,0) # bottom left
plot.addLayer(layer,1,1) # bottom right
```

Now, if we open the plot properties GUI we have all four layers and we can change each one of them if necessary. We can interact with each layer and change its properties following the command line methods too.

2. Create a plot in batch mode.

   This is useful when you have many layers to add to the plot and you want to avoid to have the plot window redrawn and reajusted each time a new layer is added. From the above example:

```
plot = PlotXY()
plot.setBatch(1)
layer = LayerXY(x,y)
layer1 = LayerXY(x1,y1)
layer1x = LayerXY(x1,y1/5.0)
layer1y = LayerXY(x1/5.0,y1)
plot.addLayer(layer,0,0)
plot.addLayer(layer,0,1)
plot.addLayer(layer,1,0)
plot.addLayer(layer,1,1)
plot.setBatch(0)
```

9.5. Plotting table datasets with the TablePlotter

You can plot table datasets with the *TablePlotter* tool. You can find more information about TablePlotter in the *User's Manual* (Section 4.15.1) and in this *HowTo's Manual* (Chapter 10).

Chapter 10. HowTo Inspect and Plot Dataset Tables in HIPE

Herschel Editorial Board

10.1. Introduction

This HowTo is a description of how to create and inspect a simple TableDataset in HIPE. It will walk you through the necessary steps to create a dummy TableDataset, if you don't already have one--using the command line window. We will show you how to manually inspect the values in the table and, then use TablePlotter to plot data within the table, NOTE: Currently it is not yet possible to run TablePlotter within HIPE, but this option will soon become available.

A `TableDataset` is made up of a number of columns. Each column contains an `ArrayDataset` (data), a description and a quantity value associated with the `ArrayDataset`. Each `ArrayDataset` can have up to 5 dimensions and can be of varying types.

Constructed on 2008/06/23 19:14...

10.2. Steps to creating and viewing a simple TableDataset with the HIPE GUI

These are the steps to follow to create, view, and plot graphs of a TableDataset within HIPE.

1. Step 1: Open HIPE's "Welcome" window and click on Workbench Icon

2. Next we assume here that you do not have a TableDataset loaded into your session. If you already have one loaded into HIPE, then skip to the next item. Otherwise read on. Type the following commands into command-line window containing the "IA>>" prompt (bottom center in the default view). In the example given here, we will create a `TableDataset` with 3 columns each containing a 1D dataset, one being a sequence of numbers from 1 to 100, the second being the sine value of each of the numbers in the first column, and the final column containing the values in the first column multiplied by 100. The column names are `x`, `sin` and `y` respectively.

```
from herschel.share.unit import *
x = Double1d.range(100)
t = TableDataset(description="This is a table")  # ❶
t["x"] = Column(data=x, unit=Duration.SECONDS)  # ❷
t["sin"] = Column(data=SIN(x),description="sin(x)")  # ❸
t["y"] = Column(data=x*100,description="x*100")
```

   ❶ This sets up the table dataset with an associated description
   ❷ This creates our first column which has the data, `x` and its associated units, which in this case is a time duration of SECONDS.
   ❸ Here we have applied the `SIN` function from the numeric package, and we have also added a description for the second column.

   Notice that when you create the variable x and the TableDataset t, they appear in the "Variable" window in the top right.

3. Next we wish to view the table we have created. Move your cursor over the item "t" in the Variables window and right mouse click on it. Choose the OPEN WITH option in the drop-down menu and select Dataset Viewer. At his point a view of the table will appear in the Editor window and you can

scroll down and view the table, and expand it if necessary using the cursor and left-mouse clicking at the boundries of the window to re-size it.

4. Now we wish to view the table in the TablePlotter task. Again right-mouse click on the item "t" in the variable list and select OPEN WITH item "TablePlotter". This will bring-up the TablePlotter GUI in its own window. A complete guide to the TablePlotter is found in the Herschel DP Basic User's Manual. We list below a brief guide to TablePlotter.

10.3. Guide to TablePlotter Controls and their functions

The TablePlotter provides the following control buttons to view and analyze data.

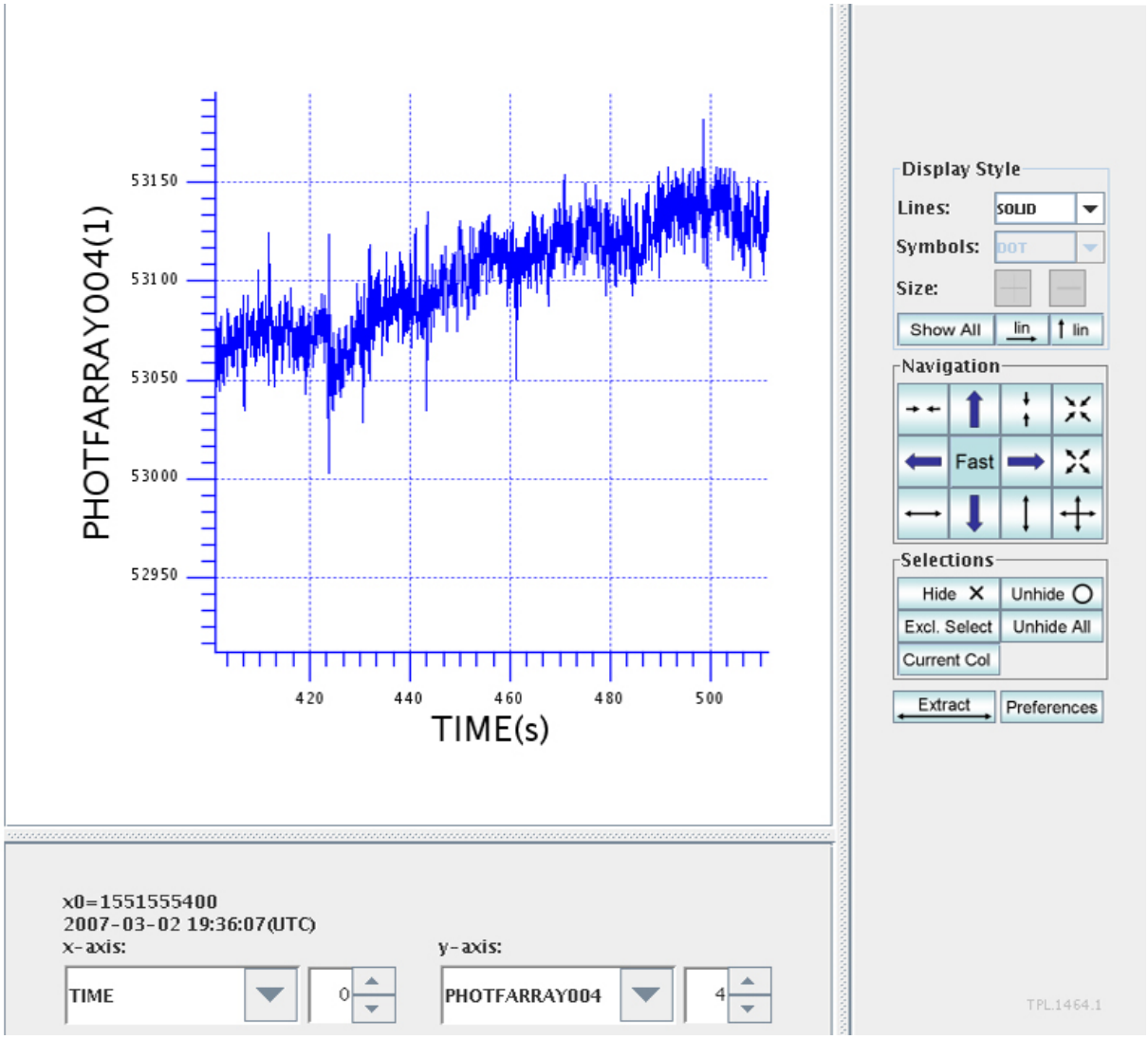


Figure 10.1. Example of the TablePlotterGui.

- **X and Y- Axis Selection:**

  Under the graphics display area, two sets of Combo Box buttons and spinner buttons allow users to select X and Y-axis data. The first column of the TableDataset is associated with X-axis by default. The second column is initially associated with the Y-axis.

  Users can choose a column by name in the Combo Box and by number in the spinner.

Fast forward/backward selection of columns in the spinner can be achieved by holding the left mouse button down and moving the mouse up or down to select.

- **Display Style:**

  The control buttons in this section allow to change the axis style (linear or log), line style (solid or dashed and more), and symbol style.

  The default axis scaling is linear. The toggle buttons 

 / 

 and 

 / 

 allow to switch between linear and logarithmic scales in the X / Y axes respectively.

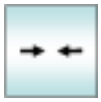
  The pull-down menus of Lines and Symbols allow to select line style and symbol style. The selection of symbol styles is only available when the line styles are either *MARKED*, *MARK_DASHED* or *NONE*. To increase or decrease the symbol size, click either  or  .

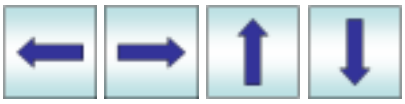
  Another toggle button 

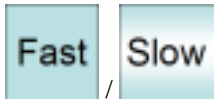
 / 

 determines whether all data points or only the selected ones are shown (see detail in Selections below).

- **Navigation:**

  Two buttons are provided to simultaneously zoom in  and zoom out  in both X and Y directions. The buttons  and  zoom out individually in either X or Y direction .

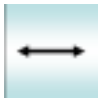
  Four buttons with arrows  pan the view onto the graph in X or Y direction.

  The size of each zooming or panning step is controlled by a toggle button 

 and the exact factors of both fast and slow modes can be adjusted in the 

 menu (for details see the *Preferences* section below).

  To Zoom-in on specific areas of the graph, press the left mouse button, and hold-and-drag a rectangle around the area of interest with the mouse pointer.

  There are three *Free Scaling* buttons. The  button will adjust the scales of both axes such that all visible datapoints are optimally distributed within the display, while  and  will do the same but for either X or Y axis alone.

- **Selections:**

The selection feature of TablePlotter allows to hide or select a particular portion of the data points. In combination with Show All / Sel Only in the Display Style section, and Multi Column Mode, this feature can be used to display only selected data to get fast automatic scaling when scanning through many columns of data. The main purpose, however, is the extraction of specific data points into new datasets. A typical purpose could be for instance to remove electronic glitches from detector data, or to extract a specific piece of signal from a sequence of instrument configurations.

The following buttons Hide ✕ Unhide ○ Excl. Select, hide, un-hide or exclusively select all data points within a rectangular area in the plot. This area is selected after pushing one of those three buttons by holding and dragging the mouse pointer in the same way as for zooming in.

Clicking the button Unhide All will re-select all hidden data points.

If the Current Col toggle button is visible, the TablePlotter is in single column mode. In this mode hiding or selecting operations will only apply to the current column. Clicking on this button will toggle into all columns mode and the button will change to All Cols Now all the columns are affected and selections are done based on the selected intervals on the X-axis only. The Y-coordinate will be ignored in this mode.

In Show All mode the hidden data points will be marked with red symbols. See Figure 10.2 below. Clicking on Show All toggles to Sel Only mode, where all hidden data points disappear from the graph.

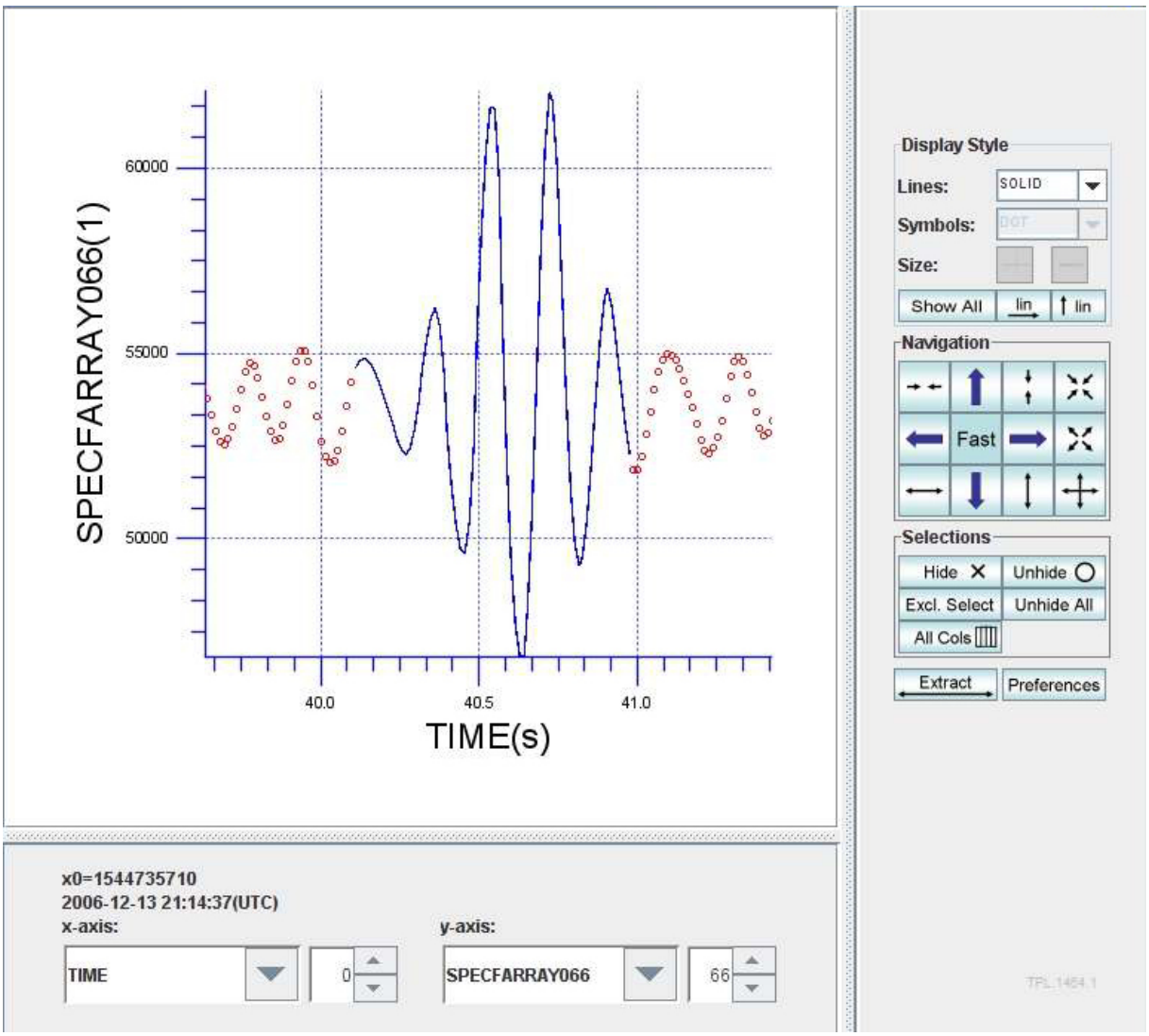


Figure 10.2. The plot with selected and hidden data points.

- **Dataset Extraction:**

  To extract a subset of the data after performing the necessary selection operations, press the 

button. The selected data will be extracted into a new dataset that will be fed back to DataInspector, where it will appear in the leftmost panel under "Datasets".

  If 

is selected, only the selected data points in the currently displayed column will be extracted.

  If 

is selected, the selected data points in all the columns become available for extraction. After clicking 

, a column selection window will pop up to allow users to **Add** individual columns or **Add All** columns to a list. Users can also **Remove** individual columns or **Remove All**. **Up** and **Down** buttons allow to change the order of columns in the new dataset.

  Hitting the **Close** button will complete the extraction and an option is provided to change the default name of the new dataset.

- *Preferences:*

  Finally the Table Plotter proivides a Preferences menu with two options. The first one is Set properties... where preferred zooming and panning factors for Fast and Slow modes can be set.

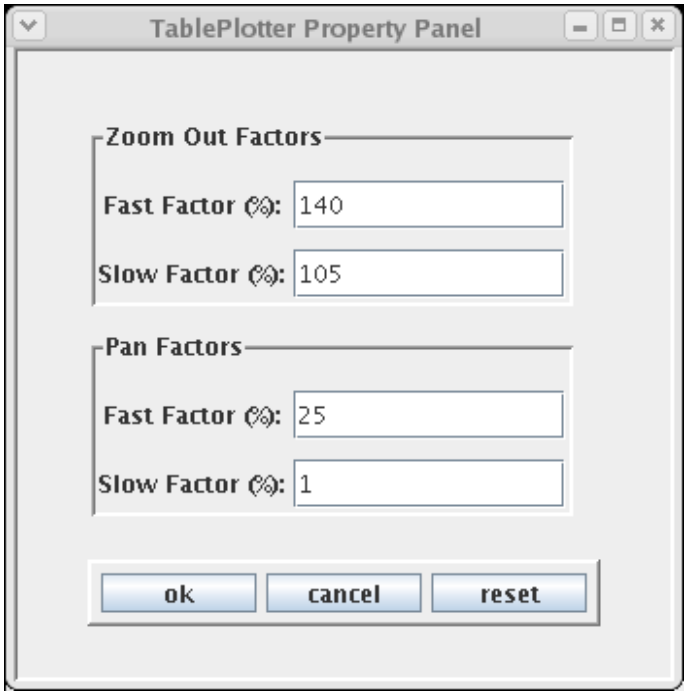


Figure 10.3. Preferences: Set Properties

The second one controls the display of Complex Data. TablePlotter allows only one graph to be displayed at a time. Here the user has three choices: plot modulus only, plot real part only, or plot imaginary part only.

The selected preferences are stored in a properties file and will be "remembered" in the next call to Table Plotter.

Chapter 11. HowTo Display Spectra

11.1. Introduction

HIFI spectra can be visualised in several ways, at various levels of sophistication and user-friendliness. At the lowest level, individual X and Y axis values can be extracted and units applied and the basic plotting facilities of the PlotXY package can be used (see "HowTo Plot" and the "HIFI DP User's Manual" for more details). However, a simpler way for most users of HIPE is to use the "Spectrum Explorer" package.

11.2. Obtaining a Spectrum from an `ObservationContext`

Most users will obtain spectra from downloading observations from the Herschel Science Archive (HSA). The main product form of an observation is referred to as an `ObservationContext`. An `ObservationContext` contains all the components of an observation, including all calibrations needed for repeating pipeline processing data. The full observation download from the HSA includes all levels of processed data from level 0 (raw data) to level 2 (final pipelined product).

An observation context can contain many spectral products (e.g., from all 4 spectrometers of HIFI) at each of the different levels of processing.

We can consider a HIFI `ObservationContext` called "prod" which has been downloaded from the HSA. A double-click on the variable name "prod" in the "Variables" view of HIPE provides an outline view of its contents in the "Outline" view (see Figure 11.1). This shows the containers of spectra at the different processed levels (also quality and calibration information associated with the observation). If we now open the level 2 folder and click on the product (highlighted in Figure 11.1), we get a view in a new "Editor" window like the one shown in Figure 11.2) -- after expanding out the folder labeled 1030.

At present, the HIFI spectrometers are identified by the values 1028 = HRS H polarization, 1029 = HRS V polarization, 1030 = WBS H polarization and 1031 = WBS V polarization. In this case the 1030 folder under level 2 is the final, processed data product from Standard Processing at the Herschel Science Centre of WBS H polarization data. The folder labeled "1" contains the first (and in this case only) final product. A double-click on product(load) -- highlighted in Figure 11.2) -- provides access to a listing of the metadata and the final dataset (scroll to bottom of "Editor" window) which is marked with a green dot beside it.

We can display this final spectrum via a viewer called `Spectrum Explorer`. As with all viewers in HIPE -- click with right mouse button on the dataset word, choose "Open With..." from the menu that appears and then click on the words "Spectrum Explorer". This will display a new "Editor" window with a blank spectrum display (initially). To fill in the spectrum the user needs to click on the four boxes (label above shows 1, 2, 3 and 4) to the left under the plot or simply clicking on the box under "ALL". This fills in the 4 sub-bands of 1GHz wide CCDs that make up the full backend spectrum from a WBS spectrometer (see Figure 11.3). An appropriate legend is automatically created.

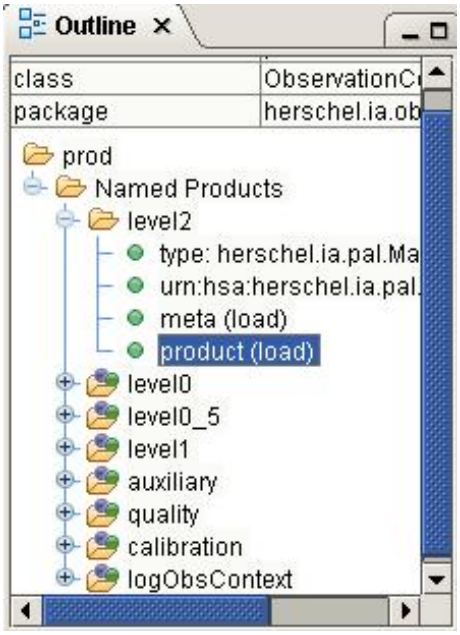


Figure 11.1. Accessing the level 2 (final) processed product from an observation.

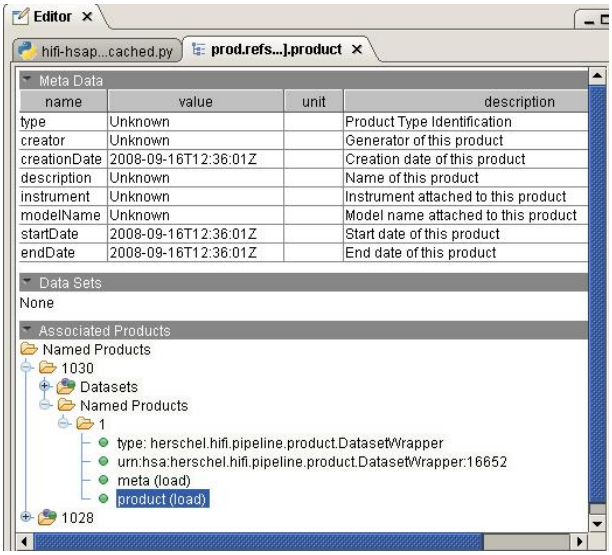


Figure 11.2. Editor view showing access to the final WBS H polarization spectrum from within the full observation tree.

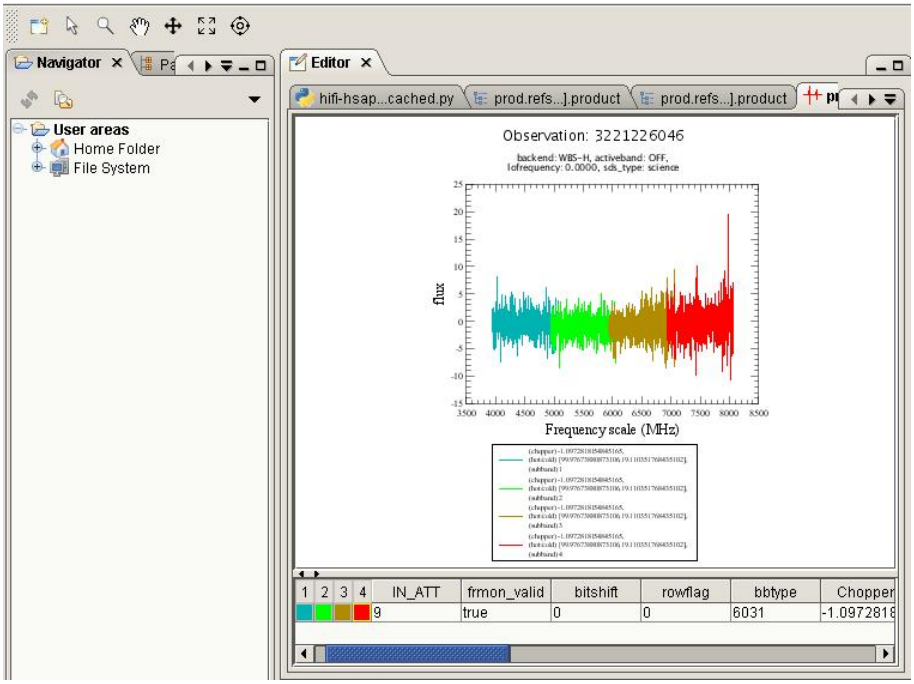


Figure 11.3. Display of a test data (no source) produced by the HIFI pipeline using the `SpectrumExplorer`.

An example of a SPIRE spectrum is provided in Figure 11.6. Here we can input (say) a FITS file using the "simpleFitsReader" available in the Tasks view (see Figure 11.4). It is also possible to input a FITS file by simply double-clicking on it in the directory display of the Navigator view of HIPE. The variable created ("product1" in this case) can be seen in the Variables menu. Click on this and the Outline is shown and it can also be opened by double-clicking. This provides access to the metadata and datasets (see Figure 11.5). Click on the spectrum dataset (with the green dot next to it) and the SpectrumExplorer is started with a display of the spectrum (click the "1" symbol to bottom left). This will then show a spectrum similar to that shown in Figure 11.6.

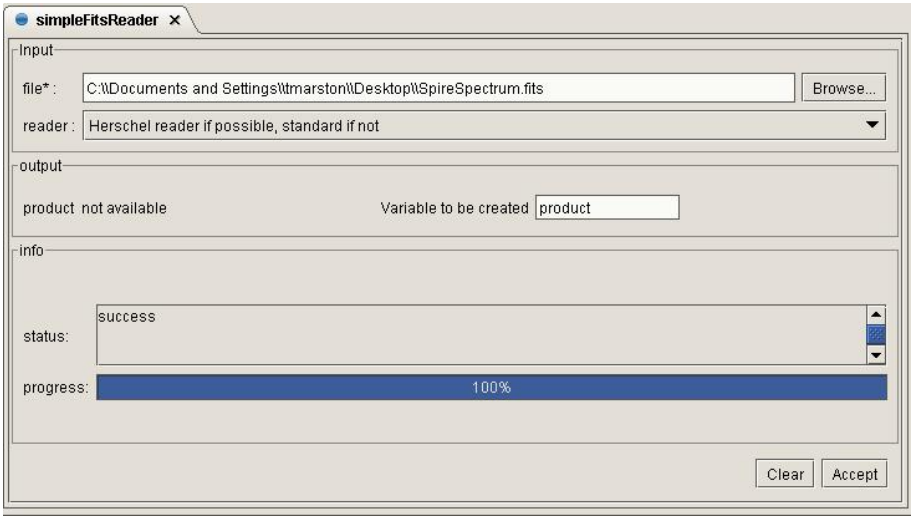


Figure 11.4. Using the simpleFitsReader task for reading in a FITS file with a SPIRE spectrum.

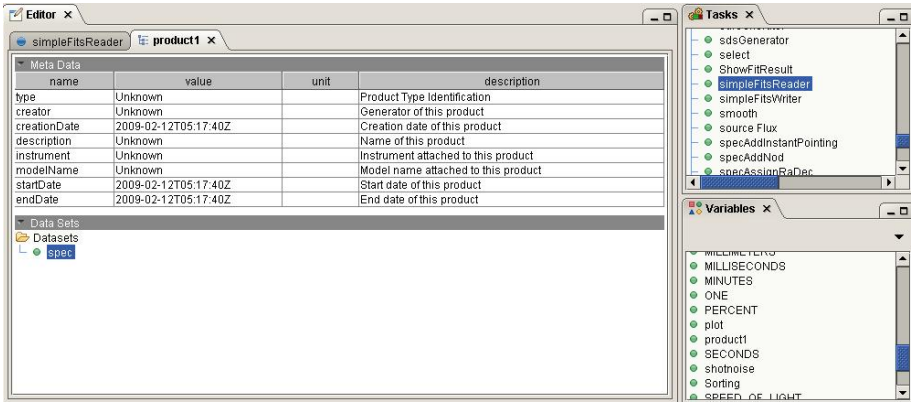


Figure 11.5. Display of the metadata and datasets in the FITS file when the variable is double-clicked.

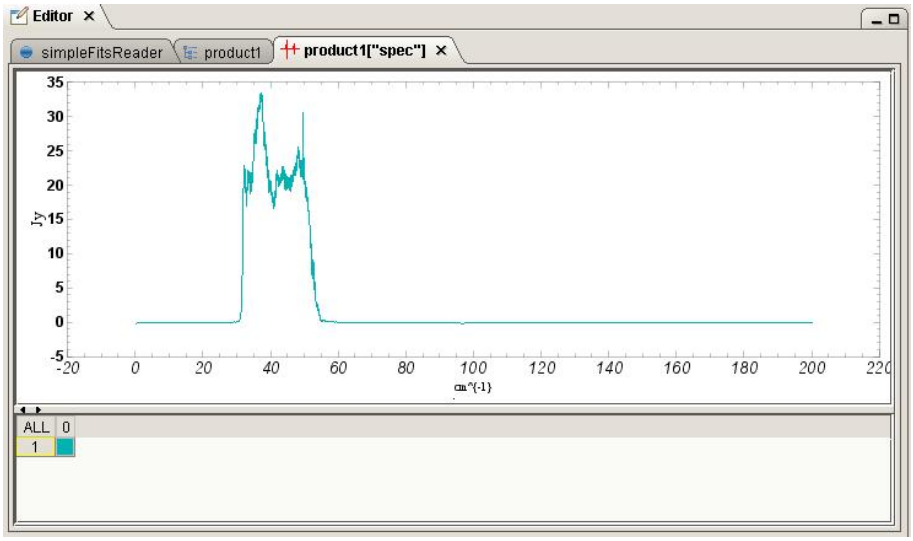


Figure 11.6. Display of the SPIRE spectrum using the `SpectrumExplorer`.

11.3. The `SpectrumExplorer` Package

The `SpectrumExplorer` package is based on the PlotXY package, but allows the user to visualize `SpectrumDatasets` in a friendlier, interactive way.

In the example from the previous section, the different colors indicate different WBS sub-bands. Individual sub-bands or individual scans can be plotted by clicking on the appropriate boxes in the bottom panel and removed by double-clicking. Any plot parameter (plot range, titles, colors etc.) can be modified using the right mouse bottom in the same way as for the PlotXY package (see "HowTo Plot Data"). For example, a right-click on the plotted spectrum allows changes in the axes and plot properties (e.g., labels fonts etc.). It is also possible to save and print the plot from the right-click menu.

In addition the plot can be modified interactively after clicking the appropriate action button which `Spectrum Explorer` places in the top left of the HIPE display (see top left of Figure 11.3). Hovering the mouse over the icons allows provides the user with a tooltip for what the icon allows you to do. From left to right:

- button 1: highlight/select a spectrum (or WBS sub-band) by moving the mouse over it and click the right mouse button to change its color, description, or remove it.

- button 2: change the horizontal and vertical plot ranges by drawing a rectangular box using the left mouse button. Also, one can scroll the spectrum along the horizontal and vertical axes by clicking

on an axis with the left mouse button and then moving the mouse or using the mouse wheel. The mouse wheel can also be used to (un)zoom the spectrum.

- button 3: pan through the spectrum by clicking the left mouse button and moving the mouse.
- button 4: click on a spectrum (or WBS sub-band) and drag it to right or down to another or a new panel which is automatically generated on release (however, dragging to the left or top of the first panel is not possible).
- button 5: click on this button to auto-range the displayed spectra (after zoom).
- button 6: only show the active plot panel, and change the axis ratio in order to fit the screen.

This allows images such as the Figure 11.7 to be constructed from the displayed data.

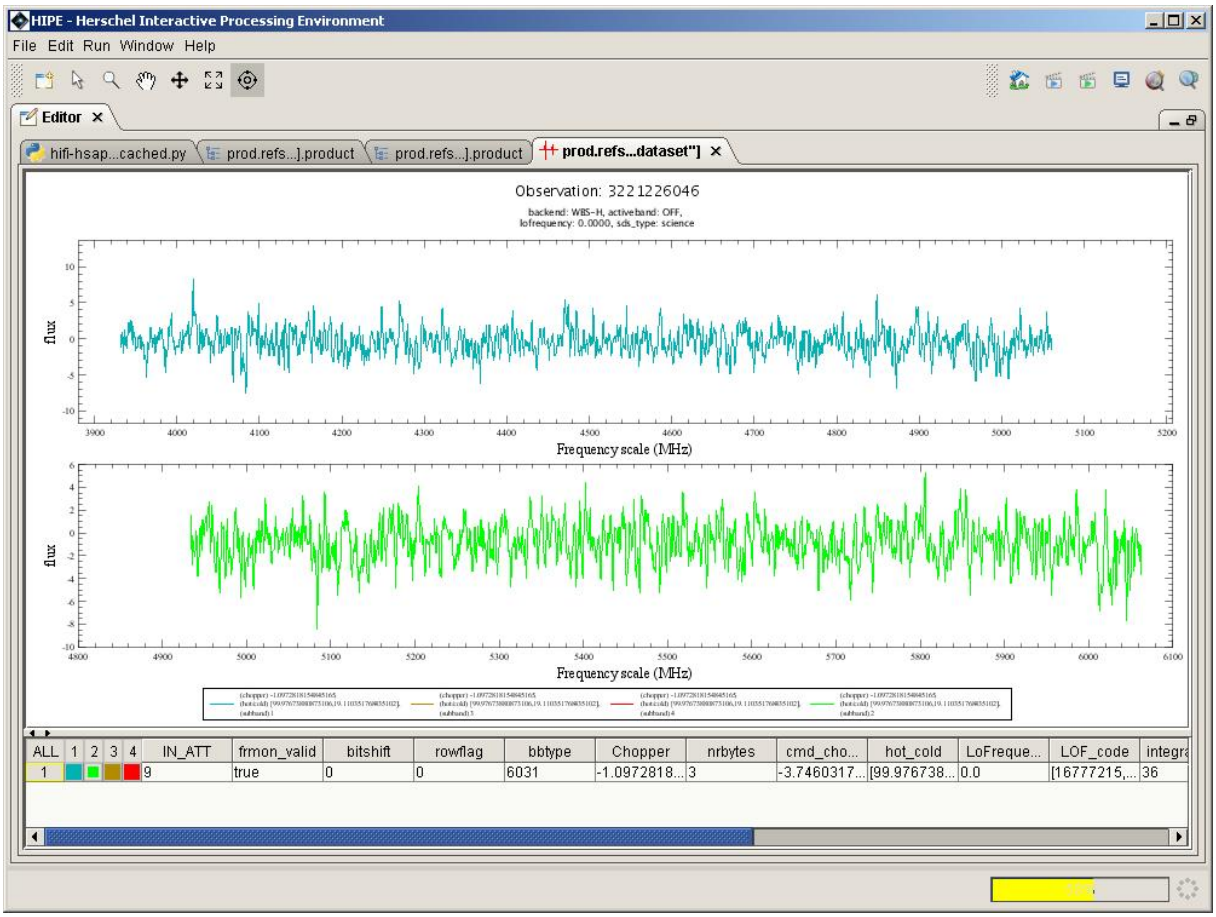


Figure 11.7. Two sub-bands extracted interactively (button 4) using `SpectrumExplorer` shown with the full resolution of the screen (button 6).

11.4. Future developments

Finally, the `SpectrumExplorer` package is still under development. Future developments include:

1. clicking on product will plot all SpectrumDatasets included (likely a HIPE functionality, not JIDE)
2. apply a filter to the meta data and only plot the applicable spectra (e.g. a certain chopper position)
3. applying functions to greyed-out/flagged spectral regions
4. saving modified `SpectrumDatasets` back into the session, e.g. with interactively masked points removed.

5. overplotting multiple `SpectrumDatasets`.

Chapter 12. Spectral Arithmetic and Mathematical Operations

12.1. Introduction

The spectrum arithmetic toolbox allows us to combine Herschel spectrum data. Operations are performed either on subclasses of spectrum datasets (`Spectrum1d, Spectrum2d`), on cubes (`SimpleCube, SlicedCube`), or on products containing such data structures (e.g., HifiTimelineProduct).

Operations on Spectra include Selection and Arithmetic Operations.

This chapter explains how to work with spectra so that basic spectral arithmetic can be done on a 1D spectrum dataset. It also indicates how to handle datasets composed of multiple 1D spectra. When working with these larger sets of 1D spectra it is also possible to select spectra based on information held in the data or metadata of the individual spectra before applying the arithmetic transformations.

12.2. Starting point -- using a dataset of a number of HIFI spectra.

It is assumed that an observation product containing spectral data is available and active within your HIPE session. For this HowTo, we will have an active variable called "prod" which is a HIFI observation downloaded from the HSA (see HowTo Access Data). This contains several levels of data processing. We will be dealing with level1 data -- double-click on the highlighted "product(load)" in Figure 12.1. The results appear in a new Editor window and include some metadata on the product plus (scrolling down) a set of associated products (see Figure 12.2). Clicking on the highlighted "summary" will provide a list of what datasets are contained for apid=1030 (the WBS spectrometer H polarization). In the particular case (a Double Beam Switch observation) we are using we see that there a comb (frequency calibration measurement), a hot-cold internal calibrator measurement (hc), a tuning measurement (other) and two science measurements datasets for ON and OFF target (datasets 4 and 5). We will pick out dataset 4 for our purposes (double-click highlighted "product(load)" gives Figure 12.3). This produces a list of metadata for the selected product and a dataset (with green dot beside it) at the bottom of another Editor window. Drag-and-drop the dataset to the "Variables" view and this dataset is automatically given a name in the session -- typically "newVariable."

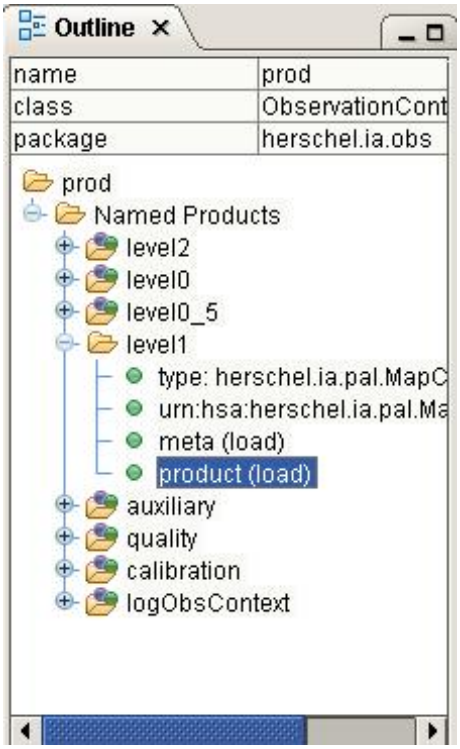


Figure 12.1. Selecting Level 1 data from a downloaded archive observation done by HIFI.

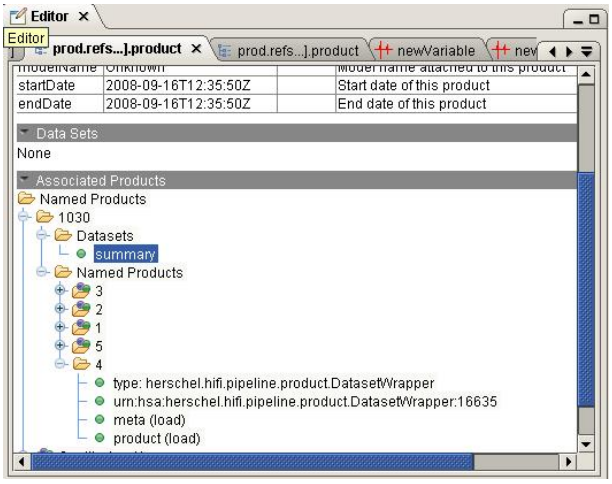


Figure 12.2. Display of product set.

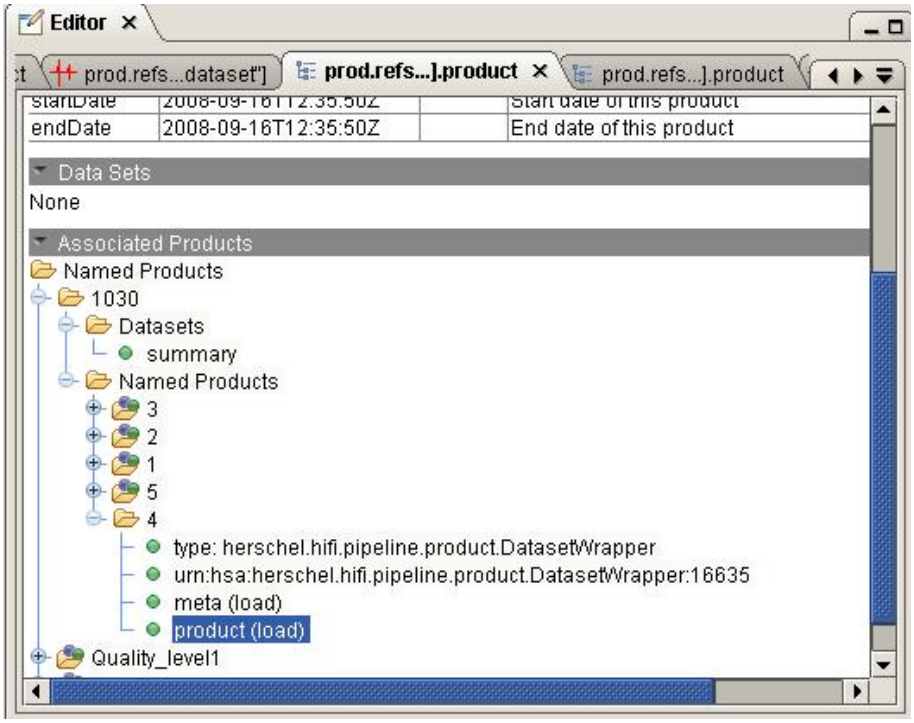


Figure 12.3. Choosing the product with the dataset we want.

A double-click on newVariable in the "Variables" view will open the dataset using the `SpectrumExplorer` (see HowTo on Spectral Display for information on how to manipulate the visualization). In the example dataset used here there are 18 spectra.

12.3. Using HIPE to Access the Spectrum Arithmetic Tasks

In HIPE the "Tasks" view gets filled with the currently available tools. Tasks that are available for use on datasets of spectra will appear under the "Applicable Tasks" folder that appears in the Tasks view. Available tasks include add/subtract/multiply/divide/average which can be seen in the Applicable Tasks folder after highlighting "newVariable" (see Figure 12.4).

Figure 12.4. Display of tasks available for our dataset of spectra.

In this section we discuss each of the available arithmetic tasks in turn.

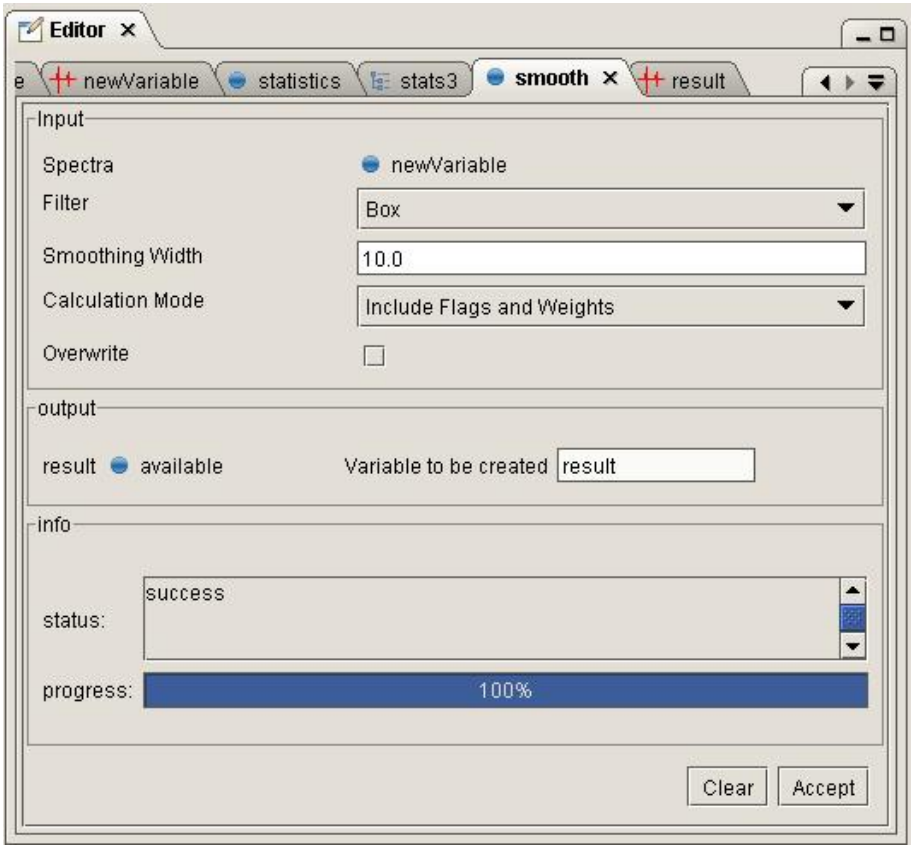


Figure 12.7. Using the `smooth` task

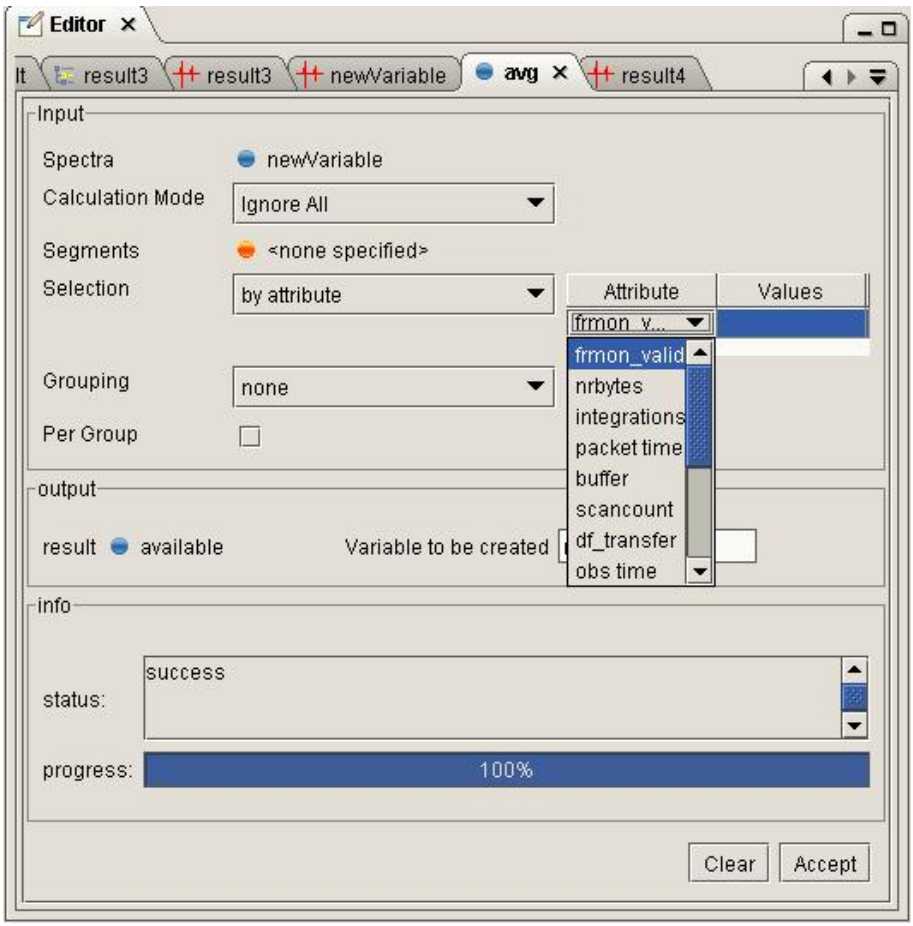


Figure 12.8. Using the avg task

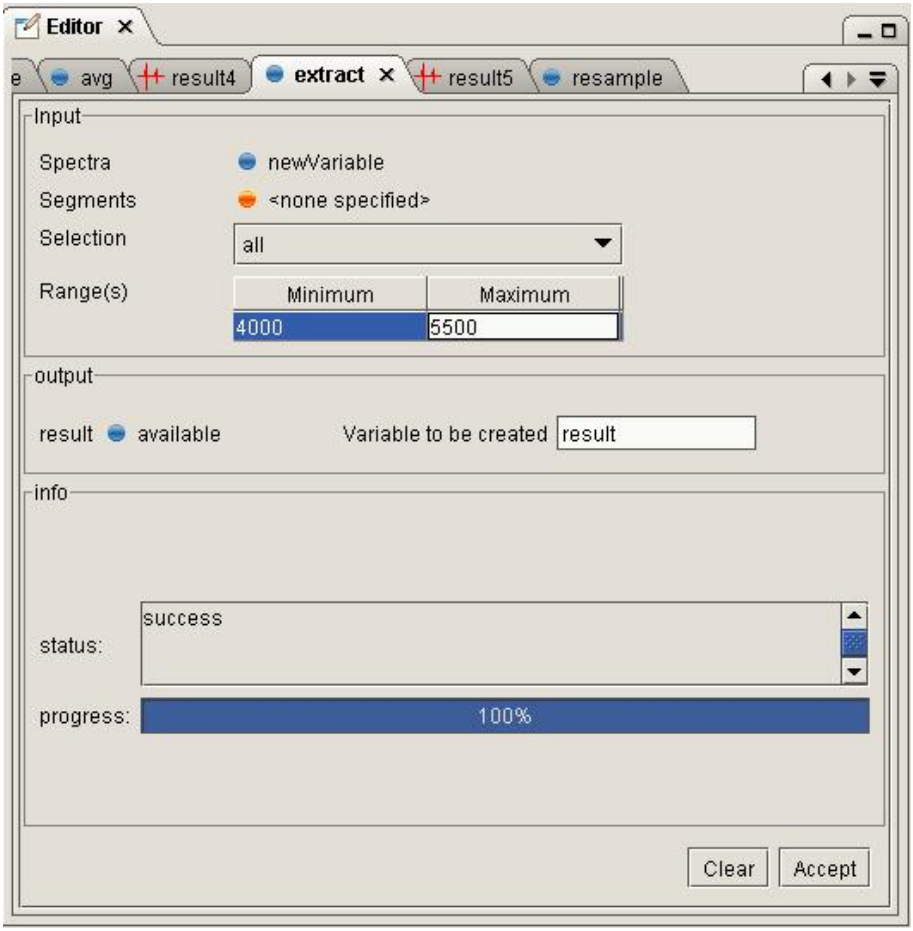


Figure 12.9. Using the `extract` task

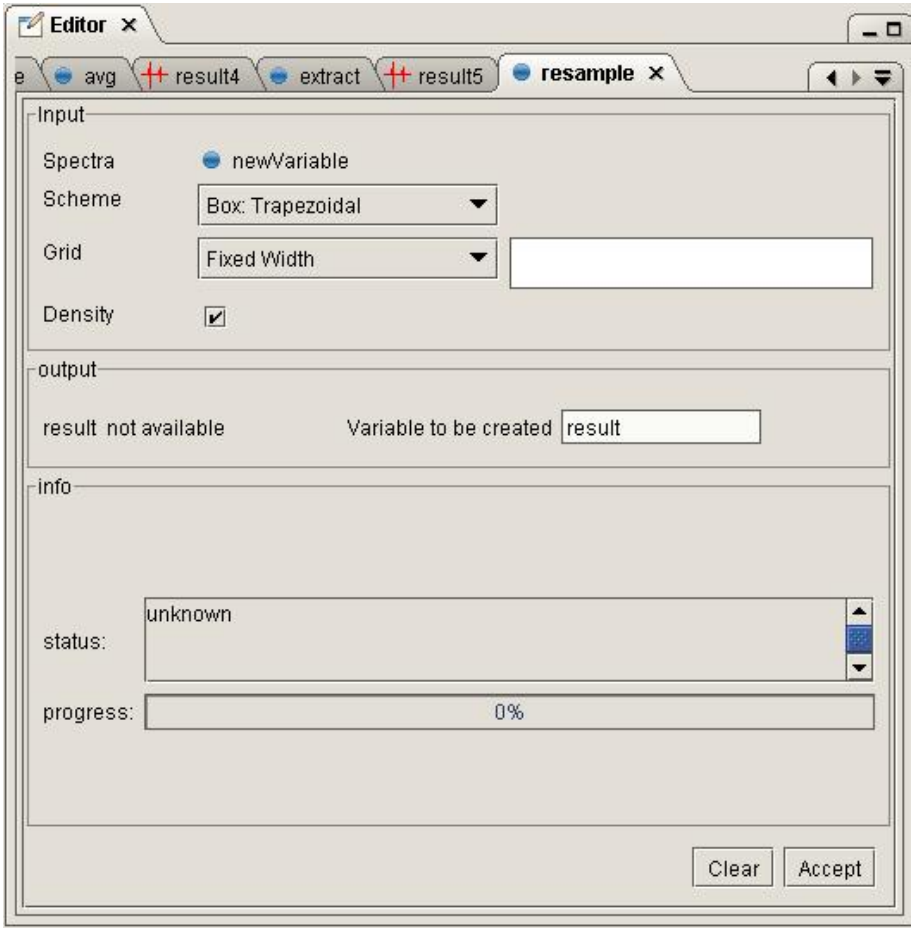


Figure 12.10. Using the `resample` task

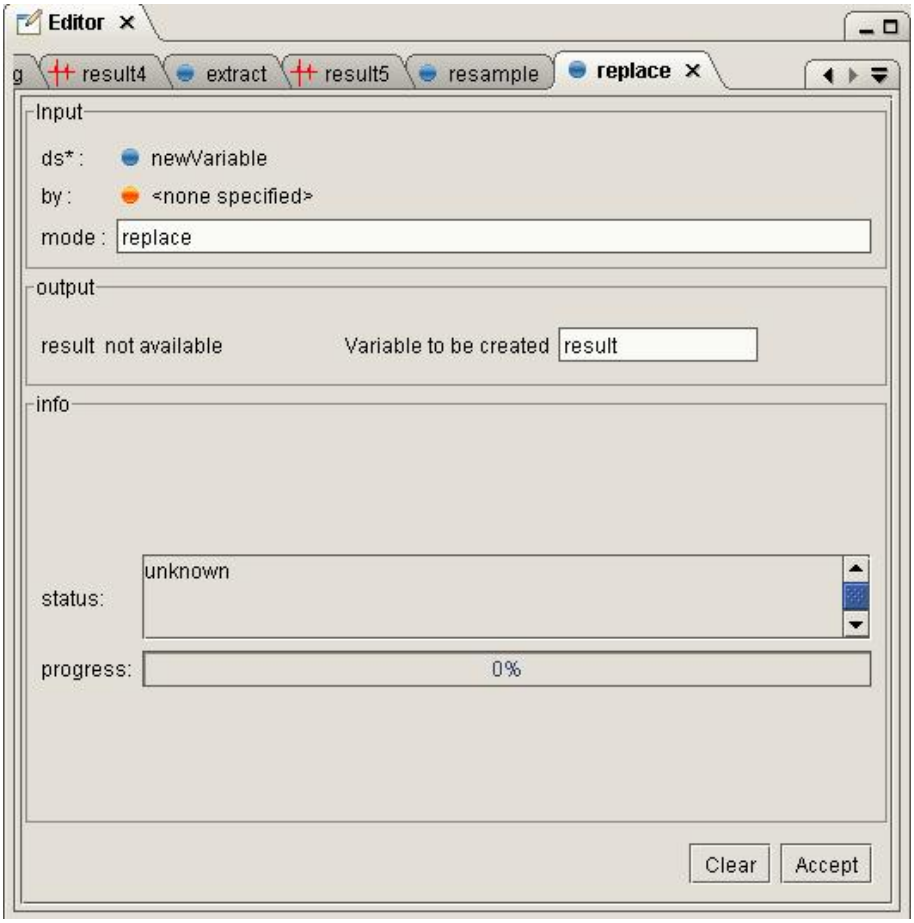


Figure 12.11. Using the `replace` task

- *select*: Provides a means of selecting those spectra that can be combined. A given attribute value or range of values can be used or simply the index number of the spectrum within the group (see Figure 12.5).

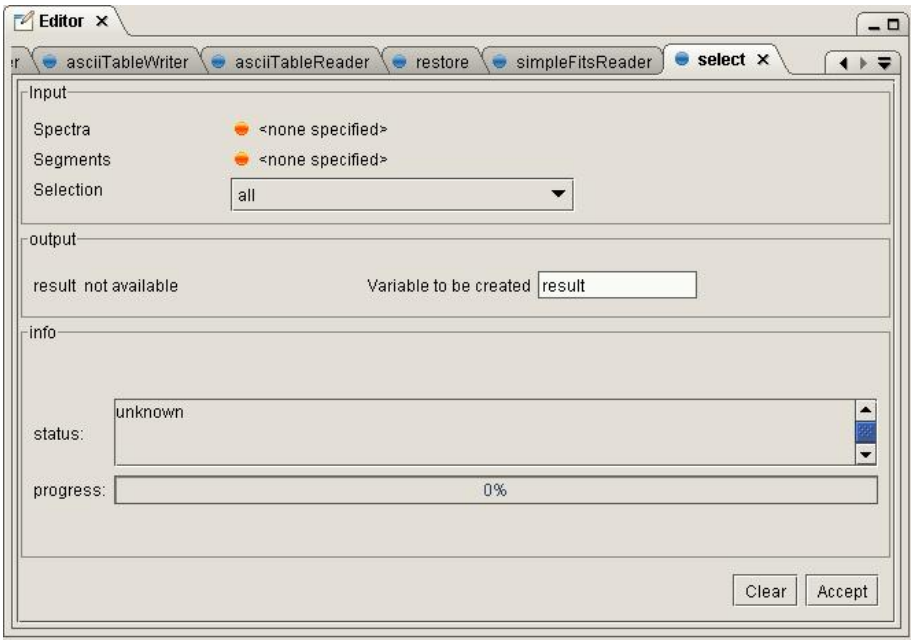


Figure 12.5. Using the `select` task

- *add/subtract/multiply/divide*: Provide means of adding/subtracting/multiplying/dividing groups of spectra or single spectra together (pair-wise), or adding/subtracting/multiplying/dividing a scalar value to/from all spectra in the selected dataset. Numbered segments, e.g., subbands, can be selected for addition if available within the dataset (see Figure 12.6 for adding the scalar value 200.0 to all spectra in our dataset)

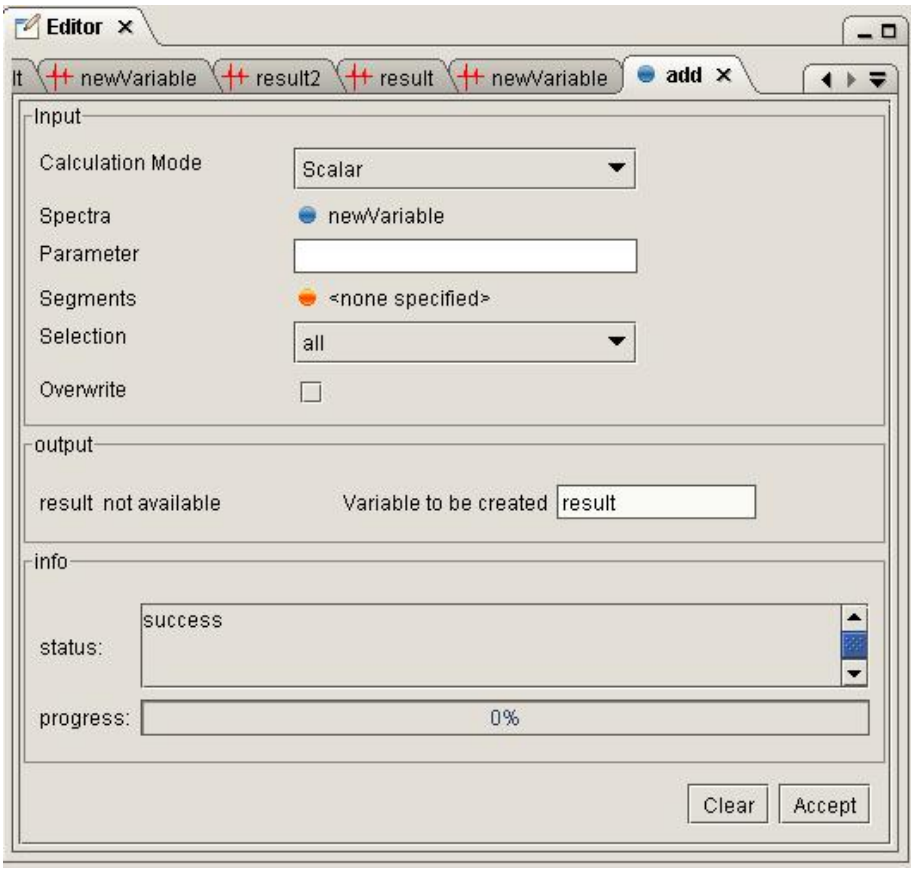


Figure 12.6. Using the `add` task

- *statistics* This allows for statistical operations to be performed on the datasets (it automatically works on individual sub-bands presently). It provide as mean, median, variance, standard deviation or percentiles for samples / selections of spectra from a dataset that can contain many datasets (spectra) when the "Accept" button is clicked. The result is an output that contains a number of datasets holding statistical information on the datasets. The main output is the "summary" table that is typically the last dataset listed of the set (double-clisk on output variable, e.g., "stats", in the Variables view. Use an appropriate viewer (Dataset viewer or Tableplotter to see the results).

- *smooth*This allows a transformation of the data via a box or gaussian (of user-selected width) smooth of the spectra in a dataset. Flags and weights for the different spectral points can be added in the future. To run this tool, click on the dataset, e.g., "newVariable", in the "Variables" view to highlight. The Applicable Tasks in the "Tasks" view include `smooth`. Double-click on this to get the self-explanatory dialog shown in Figure 12.7. The task runs by hitting the "Accept" button.

- *avg*This allows the average of a selection of spectra from a dataset. Flags and weights for individual channels/pixels can be used if available. Spectra can be selected by their index number in the dataset or by attributes (such as buffer number -- a pull-down selection list is available.). To run this tool, click on the dataset, e.g., "newVariable", in the "Variables" view to highlight. The Applicable Tasks in the "Tasks" view include `avg`. Double-click on this to get the self-explanatory dialog shown in Figure 12.8. The task runs by hitting the "Accept" button.

- *extract*This allows the extraction of a data from a minimum to a maximum frequency/wavelength range for the complete set of spectra in a dataset. Flags and weights for individual channels/pixels

can be used if available. Spectra can also be selected by their index number in the dataset or by attributes (such as buffer number -- a pull-down selection list is available.). To run this tool, click on the dataset, e.g., "newVariable", in the "Variables" view to highlight. The Applicable Tasks in the "Tasks" view include `extract`. Double-click on this to get the self-explanatory dialog shown in Figure 12.9, where the channels with frequencies 4000 to 5500 MHz have been selected. The task runs by hitting the "Accept" button.

- *resample*This allows the resampling of data using a Trapezoidal or Euler box, with a choice of variable or fixed width. Flags and weights for individual channels/pixels can be used if available. Spectra can also be selected by their index number in the dataset or by attributes (such as buffer number -- a pull-down selection list is available.). To run this tool, click on the dataset, e.g., "newVariable", in the "Variables" view to highlight. The Applicable Tasks in the "Tasks" view include `resample`. Double-click on this to get the self-explanatory dialog shown in Figure 12.10. The task runs by hitting the "Accept" button.

- *replace*This allows the replacement of certain frequency/wavelength channels. To run this tool, click on the dataset, e.g., "newVariable", in the "Variables" view to highlight. The Applicable Tasks in the "Tasks" view include `replace`. Double-click on this to get the dialog shown in Figure 12.11. The task runs by hitting the "Accept" button.

It is planned that the arithmetic toolbox will provide generic functionality for all instruments (HIFI, PACS and SPIRE). Instrument-specific behavior will be pre-configured by defaults in the system but will be able to be overwritten by the user.

Full command-line versions of the spectral arithmetic tools is also available and is described in the "DP Basic User's Manual."

Chapter 13. The Cube Spectrum Analysis Toolbox

13.1. Introduction

The Cube Spectrum Analysis Toolbox (CSAT) is an interactive, user friendly toolbox which provides navigation, quick access, manipulation and analysis of spectral cubes in HIPE.

The navigation and quick access parts offer the following features:

- Navigation through the cube along the spectral axis
- Quick look and extraction of individual spectra
- Quick look and extraction of spectral regions

The cube manipulation tools offer the following features:

- Extraction of sub-cubes in the spectral or spatial domain
- Creation of monochromatic images in the spectral domain.

The analysis tools offer the following features:

- Creation of PV diagrams (position-velocity maps)
- Creation of velocity maps creation

This chapter will cover the following topics:

- The graphical user interface of the CSAT and the features it offers
- Accessing these features from the command line

The CSAT works with cubes of type `SimpleCube` and `SpectralSimpleCube`. Please see the *User's Manual* for details on how to create and manipulate these cubes: Section 4.11.1.

Note

To be usable by the CSAT, a cube must have a valid WCS (World Coordinate System). See the *User's Manual* for details: Section 4.13.

13.2. The CubeSpectrumAnalysisToolbox GUI

The cube tool graphical user interface offers you a more user-friendly way to run the various functions (*Tasks*) of the spectral cube analysis toolbox. Each of the functions can also be run straight from the HIPE command line.

The cube tool is available in the package herschel.ia.gui.cube, while the individual tasks are located in the package herschel.ia.toolbox.cube. The cube tool accepts data of type `SimpleCube` or `SpectralSimpleCube`; you can find out what type your cube is either by hovering over it in the *Variables* panel, or with the following command (assuming `myCube` is the name of your cube):

```
print myCube.class
```

For example, cubes produced by HIFI and SPIRE pipelines are in `SimpleCube` format. For PACS, the cube coming out of the task `specProject` is also in `SimpleCube` format. All these can immediately be ingested in the GUI.

The cube tool can be launched in the following ways:

- By right clicking on a `SimpleCube` or `SpectralSimpleCube` product in the *Variables* panel, where the cube tool (along with other tools) is offered as a viewing option (`CubeAnalysisToolbox`).
- From the command line, with a command like the following:

```
myCubeResults = CubeSpectrumAnalysisToolbox(myCube)
```

This command opens the cube tool with `myCube` in it. Alternatively, you can do the same in two steps:

```
myCubeResults = CubeSpectrumAnalysisToolbox()  # Opens an empty window
myCubeResults.setCube(myCube)
```

With the command line method, anything you create via the cube tool will be put in the `myCubeResults` variable, from where it can be accessed later. Both usages will return the result of all the operations done with this tool as a new variable in HIPE. The command line allows in addition to access the results in a second way and, for some features, in a different format. All this will be described later.

13.3. Using the GUI

When you launch the CSAT the following window will appear:

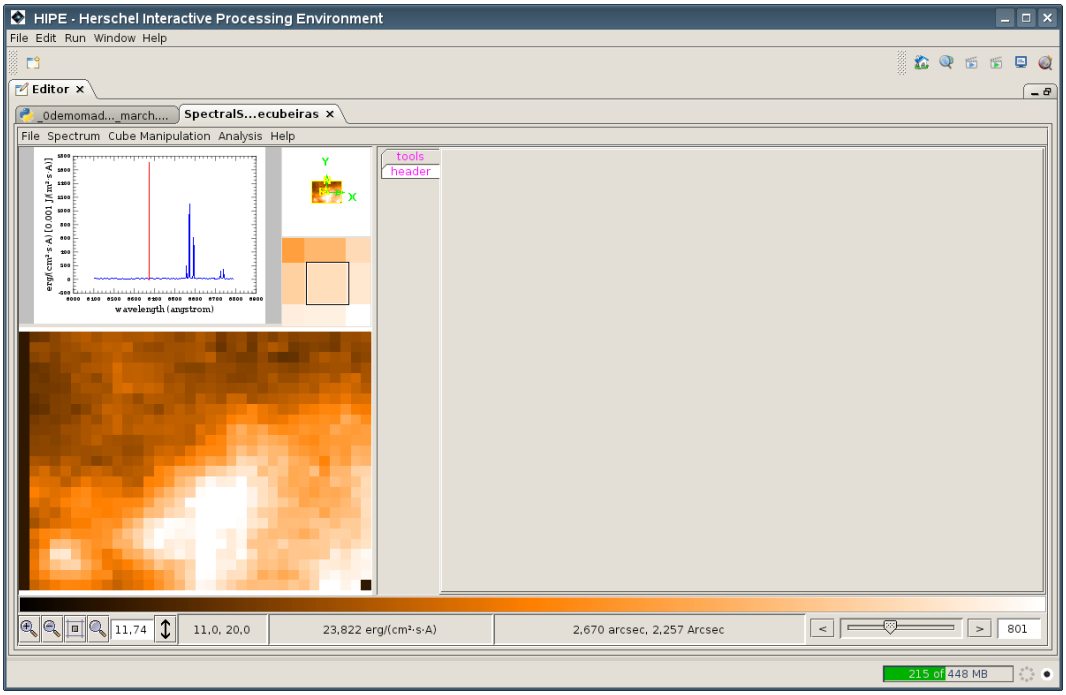


Figure 13.1. The first view you will have of the cube tool GUI

The CSAT will appear within the *Editor* HIPE view. We recommend you maximise the view or take it out of the main HIPE window (see the *Views* section in the HIPE chapter for details: Section 2.7.1). You may then want to zoom-to-fit-window on the cube image, and resize the whole GUI so all the information boxes fit the information in them.

Note that the CSAT is still under construction, so some features may be different from what is written or shown here.

Note

If your spectrum displays with odd ranges, it is possible that previous plot settings are still in effect. Try selecting the spectrum plot's Properties and choose Auto range → First Layer → Both Axes.

13.3.1. Design

The CSAT interface is split into two main regions:

- On the left side (the *image side*) the imported cube is displayed as a large image. The spectral slice of the image currently shown is adjustable with a slide bar to the lower right of the interface. Above the image you can see the following:
  - A real-time display of the spectrum in the *spaxel* (spatial pixel) that is under the mouse in the image, with a red vertical line corresponding to the layer (spectral cut) currently selected.
  - A *zoom and navigate* section. In the upper part you can adjust the view of the total cube plane that is shown in the large image. You can also see the N-S and E-W axes, if the WCS is included in the imported cube. The lower is a zoom of the spaxels around the mouse pointer.
- On the right side (the *working side*) are found, located in sub-tabs of the *cube* tab, the results of various selections you will have done on the cube (this is explained later). You can also find a *header* tab with the cube's header information (see figure below).

At the bottom of the interface you can find the following:

- Buttons for zoom, pan and adaptive zoom.
- Pixel coordinates, intensity value and sky coordinates (if present in the cube) of the spaxel under the mouse pointer.
- An adjustable colour bar, and a slide bar for navigating along the spectral dimension of the cube. The units of the slide bar are not spectral, but indicate the position in the spectral dimension (the array position). Spectral units are shown on the plots.

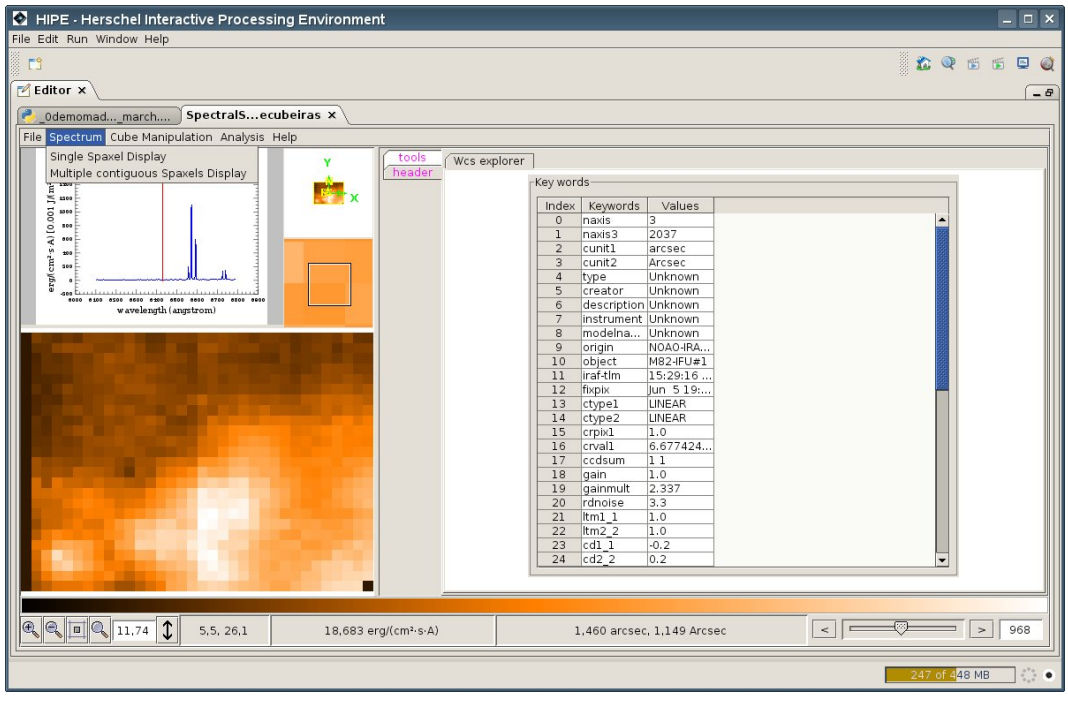


Figure 13.2. The header explorer

13.3.2. The Spectrum menu

The Spectrum menu is dedicated to spectrum extraction and allows you to select spectra out of the cube on the image side, from single spaxels or from a spaxel region. The result is displayed in the working side and can be manipulated, printed, and saved.

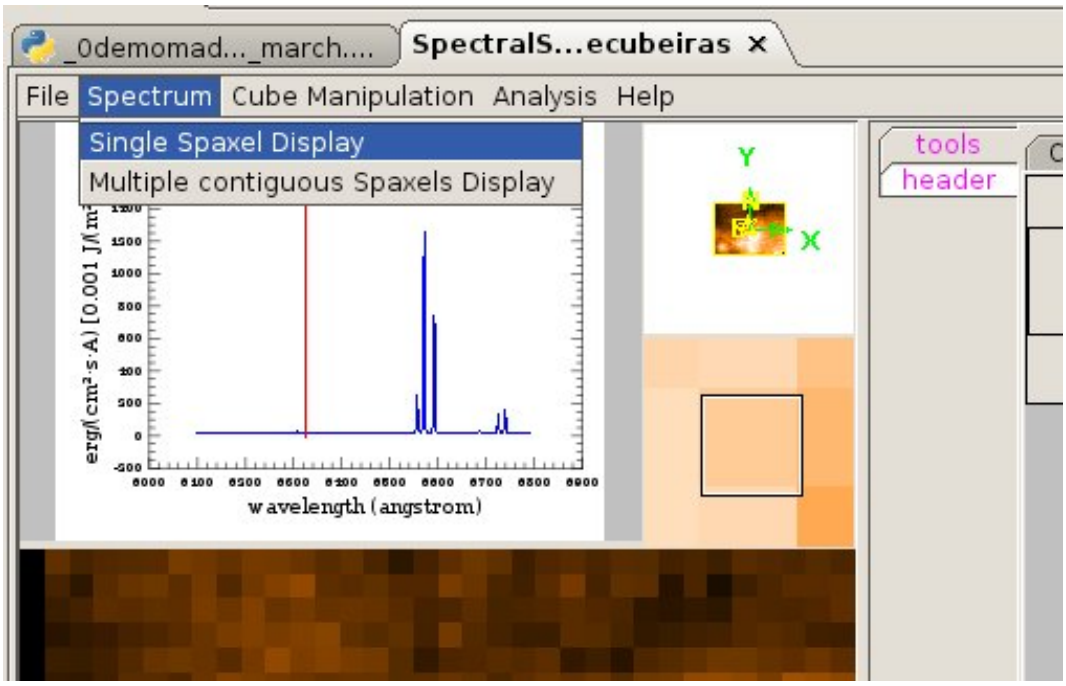


Figure 13.3. The Spectrum menu

13.3.2.1. The Single Spaxel Display

Selecting Single Spaxel Display from the Spectrum menu brings up a new pane on the working side and a blue rectangle on the cube image, that follows the mouse pointer.

The panel on the working side contains two sections:

- An upper section containing radio button and text fields for performing various operations on the spectrum (see below for details).
- A lower sections containing the plot of the selected spaxels.
- Below the plot, a status bar showing the pixel and sky coordinates of the selected spectrum.

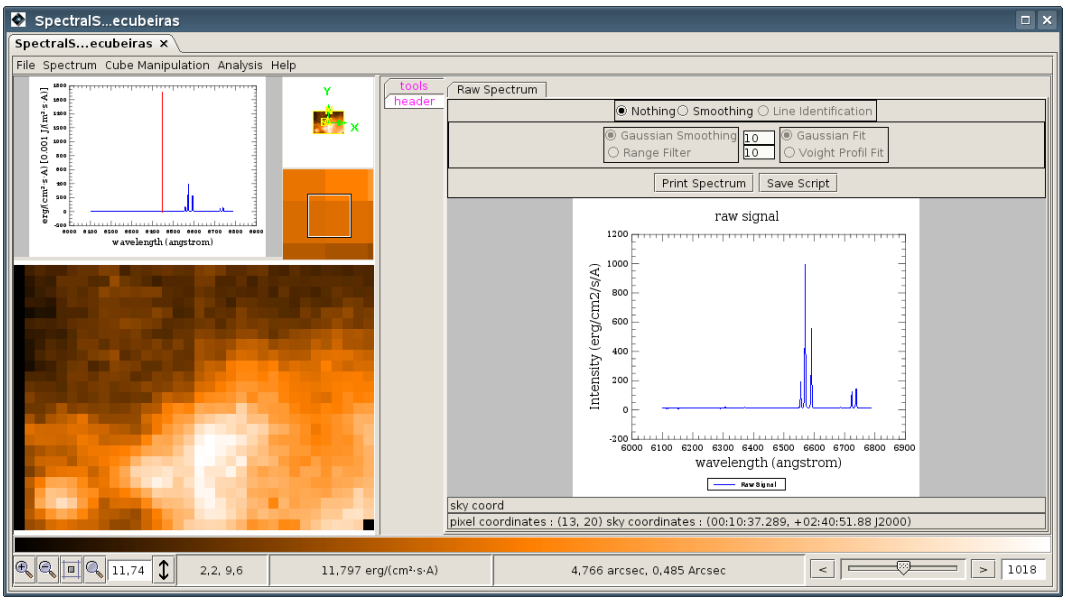


Figure 13.4. The single spaxel spectrum extraction window

The plot section displays the spectrum corresponding to the spaxel where the mouse cursor is located. With a left-button click you can *freeze* the display on the position of the mouse pointer. At the same time, the spectrum is sent to the *Variables* panel of HIPE with a default name, in `Spectrum1d` format.

The plot can be manipulated as any other plot in HIPE.

In the upper part of the tab you can find the following additional functionalities:

- Smoothing: selecting this you can perform a Gaussian smoothing or a boxcar filter, for both of which you can chose the width of the filter, in units of channels (that is, not spectral units but rather the number in the spectral axes array positions). A red smoothed spectrum is now superimposed on the blue original in the plot. Currently to see this on a frozen spectrum you need to select smoothing before you select the spaxel to freeze, or you need to go back to the image and select a spaxel again).

- Print spectrum; Save spectrum (in a multi-extension FITS file).

- Save script: will save a Jython script containing the sequence of commands you have executed (corresponding to the buttons you have pressed) within this tab, including selecting the spaxel.

To extract a new spectrum from an other pixel or to extract a new smoothed spectrum, the interface needs to be *reactivated* by clicking on one of the radio buttons of the upper panel or by re-selecting the menu.

Note that the interface is based on the `PlotXY` HIPE tool. At present you can chose to zoom on the plot on the working side, but if you go back to the image side to select a new spaxel to display, you lose the zoom you just defined and the complete new spectrum is shown.

Note

All single spectra extracted in this interface are sent to HIPE as `Spectrum1d` with a default name starting with `singlepixspectr_`, followed by the spaxel position from which it was extracted.

If you opened the CSAT from the command line, you can access the last spectrum extracted with the following method, which will return a `Spectrum1d`.

```
getSinglePixelSpectrum()
```

13.3.2.2. Multiple Contiguous Spaxel Display

The second menu item allows you to select a spectrum that is the average of a region of spaxels. The selection of this menu entry brings up a new panel in a new tab in the working side. The upper part of this new panel contains the following:

- an area to define the kind of region from which to extract the spectrum, with radio buttons
- an information section about the selected area
- buttons to execute the extraction, save the data, save the scripts and print the current spectrum
- a section for optional smoothing

The lower part of the panel shows a plot of the resulting spectrum.

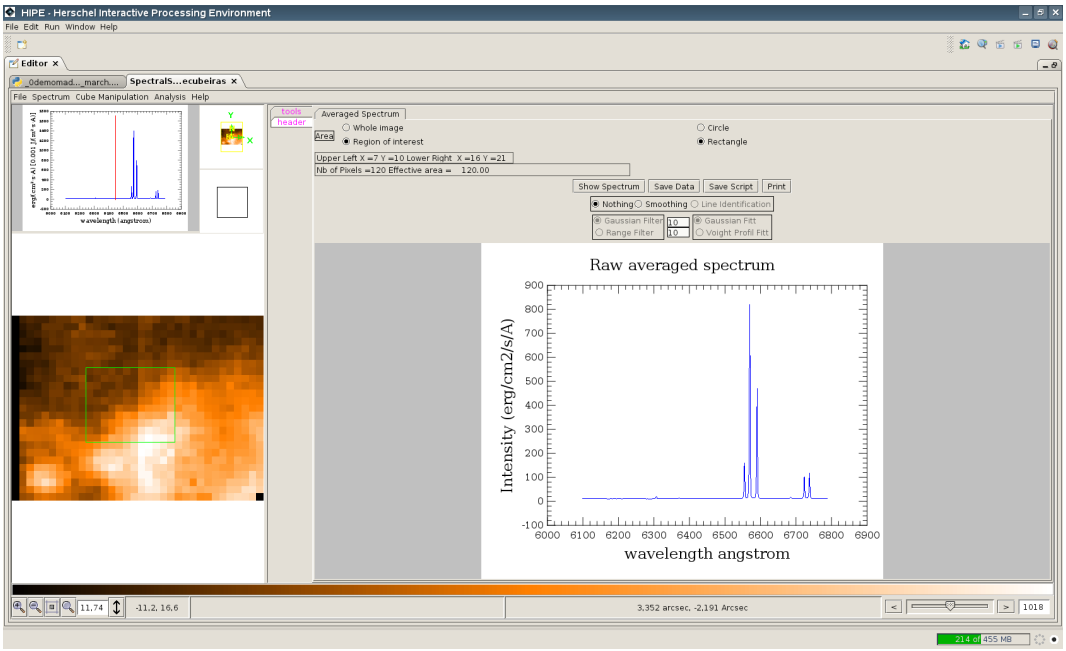


Figure 13.5. The region spectrum extraction window

The spectrum shown is the average on the region drawn on the original cube in the image side. Different kinds of region can be defined via radio buttons:

- the whole image
- a rectangle, taking only an integer number of spaxels, and with border along spaxels not through them
- a circle, for which the centre and the radius are float values. All pixels inside the circle have unit weight, while all pixels on the border of the circle have a weight corresponding to the part inside the circle (weight < 1)

When you select the *region* mode you define the region by holding the left mouse button and dragging the mouse on the cube to define the area. A green contour appears following the movement of the mouse. Release the button to set the region. To move or resize a region, first click on it to select it. You can then move the region by dragging it, and you can resize it by dragging the blue dots that appeared when you selected it.

Note

Currently the *CIRCLE* mode is not an *ELLIPSE* mode. When modifying a circle keep it as a circle: even if you can reshape it as an ellipse, it will still actually be a circle.

To see the result of your selection click on the Show Spectrum button. The spectrum is displayed in the plot section and is also sent to HIPE with a default name in `Spectrum1d` format. You can adjust the properties of the plot in the same way as with most plots in HIPE.

If you want to see a new result after having moved the selection, changed the selection method or modified a region, you must click on the Show Spectrum button.

Also available within this tab are:

- Save Script and Print
- an information bar
- a Save Data button which saves the spectrum in a multi-extension FITS file. This button will soon disappear since this can also be done in the main HIPE interface.
- the same smoothing options as offered with the single spaxel section. The smoothing information must be defined before clicking on the computation button.

Note

All single spectra extracted in this interface are sent to HIPE as `Spectrum1d` with default name `regionspectrum_` followed by the type of region selected (`whole` , `Rectangle` or `Circle`). The information about the location of the region is stored in the metadata of the `Spectrum1d`.

If you opened the CSAT from the command line, you can access the last average spectrum extracted with the `getAvgspectrum()`, which returns a `Spectrum1d`.

13.3.3. Cube Manipulation

The Cube Manipulation menu has two entries:

- Spectral Range Extraction, for extracting smaller parts of the original cube.
- Integrated Map, for integrating over a selected spectral domain.

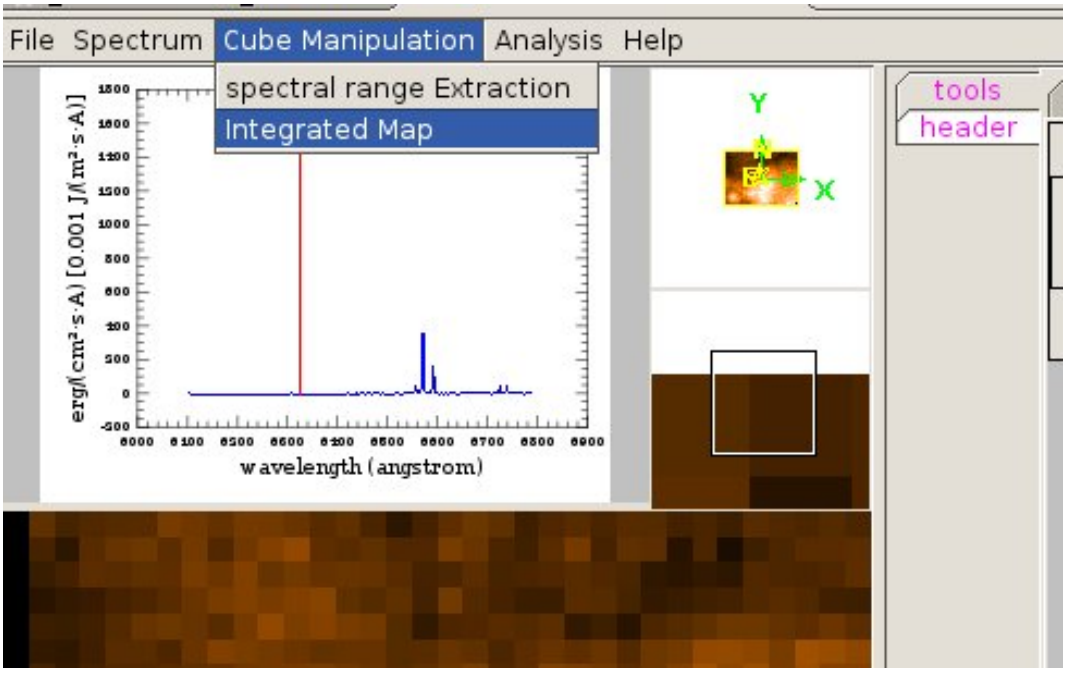


Figure 13.6. The Cube Manipulation menu

13.3.3.1. Spectral Range Extraction

This options allows you to select a particular spectral range and then save the cube over this spectral range only. This menu also allows you to proceed to a spatial extraction. *Note that this functionality is still under construction.*

After selecting this entry a new tab will open on the working side of the GUI.

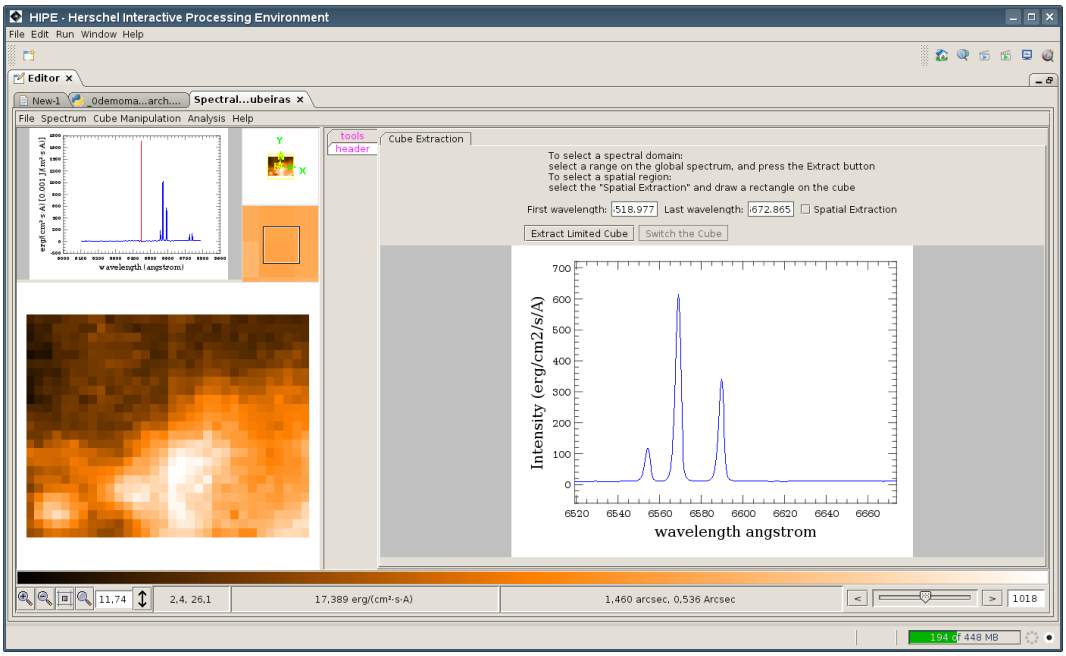


Figure 13.7. The Range extraction window

This new tab contains a plot with the global spectrum of the cube and a parameter and information section which shows the following:

- two fields with the first and last spectral values of the cube
- Two buttons, one of which (Switch the Cube) is currently not functional

The wavelength limits shown on the plot are of the current cube, i.e. the original cube on first start-up, and then updated each time you extract a cube on a smaller range.

Extracting a smaller cube with spectral limits

The main purpose of the cube extraction tool is the extraction of a smaller spectral range. To extract a cube on a smaller spectral domain do the following:

- Zoom in on the spectral range you are interested in with the usual mouse box selection on the plot in the right part of the window (do not type numbers in the boxes). You can zoom out by right-clicking on the plot and choosing Zoom Out from the contextual menu).
- Click on the Extract Limited Cube button, which will open a standard dialogue window where you can save your selection in FITS format. The behaviour will be the same if you choose to extract a spatially limited cube (see below for details).

If you do not specify a file name, the cube is saved with the default name `extractedCube_` + date + hour + `.fits`).The cube is also sent back to HIPE, with the default name `rangecube`.

Extracting a cube with spatial limits

You can crop the cube spatially (in fact you can crop the cube on both the spatial *and* the spectral dimension). To crop on the spatial dimension, select the Spatial Extraction checkbox (see figure below) located above the plot, next to the wavelength/frequency limits.You can then select a spatial region

of the cube by dragging the mouse pointer over the image (in the image side) to draw a rectangle. This feature comes in addition to the spectral selection i.e the resulting cube will cover the rectangle drawn on the left *and* the spectral domain selected on the right (if you did not do a spectral zoom it will retain the original range).

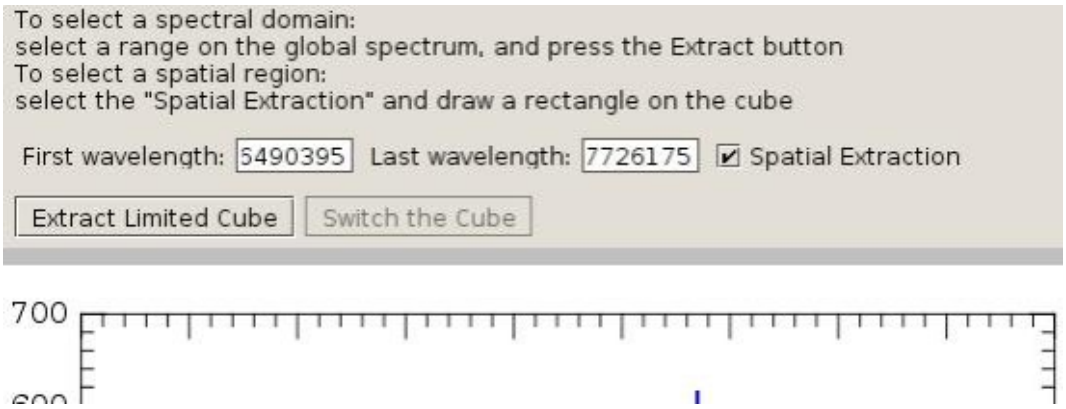


Figure 13.8. The range of extraction and the radio button for the spatial extraction

After clicking on the Extract button the process is the same as explained previously: the same windows appear and the cube is returned to HIPE. The default name is now `rangespatialCube`, of type `SpectralSimpleCube`.

If you opened the CSAT from the command line, you can access the last extracted cube via the `getRangeExtractedCube()` method.

13.3.3.2. Integrated Map

The Integrated Map menu item allows you to define a spectral range over which to integrate the individual *layers* of the cube (or *image slices*). The result is a `SimpleImage` (that is, a two-dimensional product) and contains the sum of the individual layers for the given range.

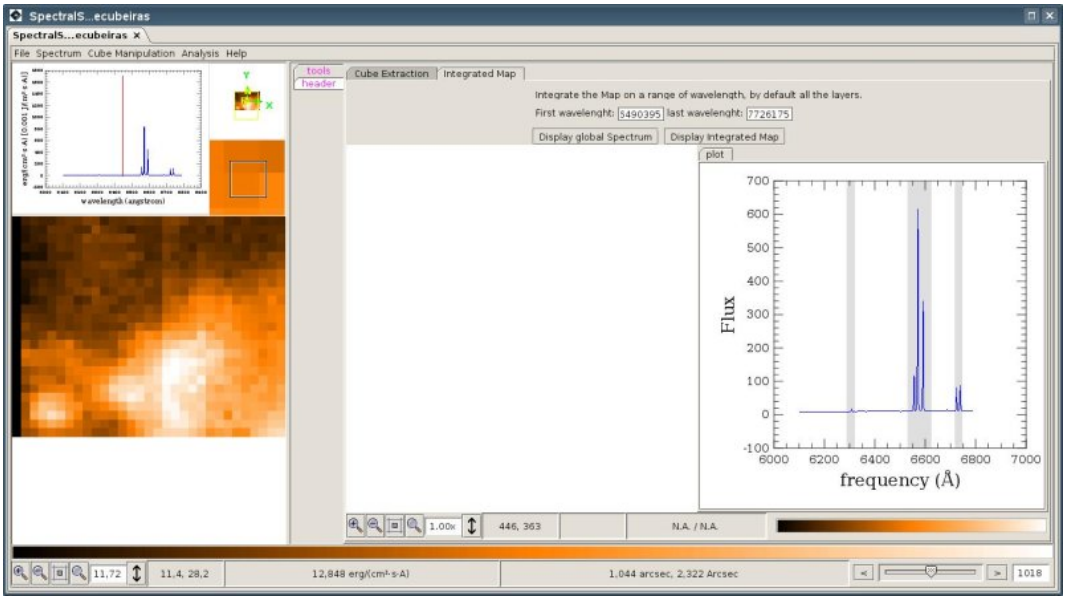


Figure 13.9. The integrated map interface

The right part of the window is divided in three:

- An Info & Parameters section, displaying the values and units of the spatial axis of the original cube and two buttons, Display Global Spectrum and Display Integrated Map. This will be modified in the next release of the CSAT.

- A Display section, which receives the result of the integration as one `SimpleImage` per specified range.
- A Spectrum Plotter section, which shows the global spectrum of the cube (average of all the spectra).

To integrate one or more ranges (or spectral domains) do the following:

1. Display the global spectrum in the Spectrum Plotter by clicking on the Display Global Spectrum.
2. On this spectrum choose the Select Range tool by right clicking and choosing Tools → Select Range. Then with the mouse select one or more ranges to integrate.
3. Launch the integration by clicking on the Display Integrated Map button.

Note

This tool only allows you to select separate ranges, i.e one range must end before the next one starts, and there is no possibility to extract overlapping ranges in one step. However it is possible to clean the selection after having integrated a first set of integrated maps and therefore to create a new set of images to integrate.

The resulting images are displayed in the same graphical component; if you selected many ranges the images are "stacked" and can be selected by using the slide bar.

Every time you select a range and click on the button to integrate and display the resulting image, these images are sent to HIPE as a set of new variables of type `SimpleImage`.

If you opened the CSAT from the command line, you can retrieve the resulting images as a list of images with the following command, which returns a list of images:

```
a = cat.getIntegratedMaps()
```

The images can then be accessed like this:

```
img1 = a[1]  # A SimpleImage
```

13.3.4. The Analysis menu

The Analysis menu is dedicated to the analysis of the cube itself. All operations take the cube as primary input and return a 'scientific' result.

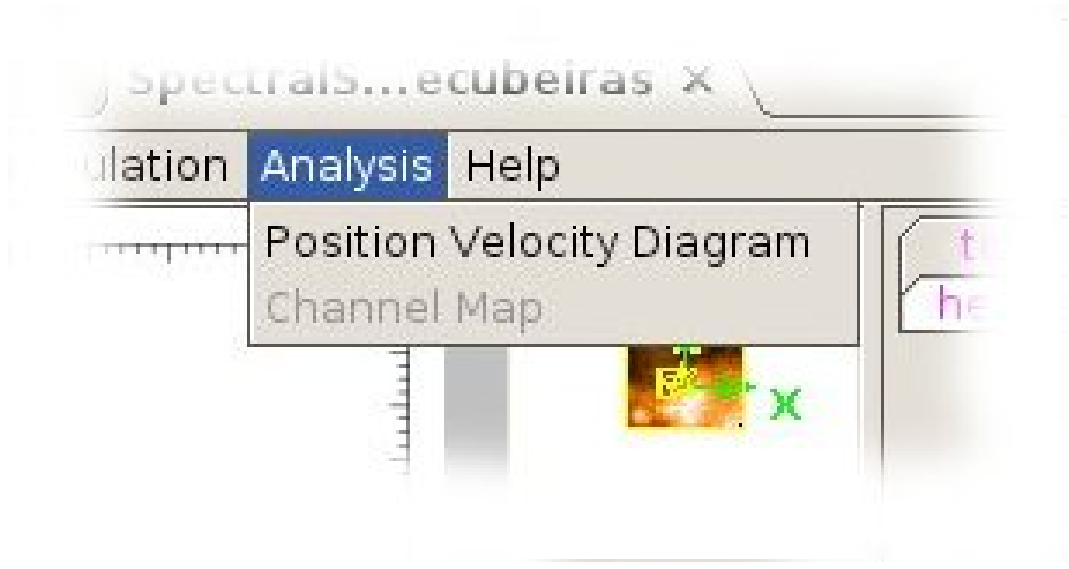


Figure 13.10. The analysis menu

13.3.4.1. Position-Velocity diagrams and maps

This item allows you to create Position-Velocity (PV) diagrams from a cube. Choosing this option will open a new tab on the working side. You can chose between two modes (indicated with radio buttons):

- Axis, which works by allowing you to select a slit along which the PV diagram is computed.
- Map, which makes a 2D map with intensities being the velocity values. *This menu option is still under construction!*

For both modes you need to define, using the slide bar at the bottom of the working side, a reference layer. This layer will define the spectral value corresponding to the velocity "0" (zero) for the PV diagram to be computed. You will notice that as you move this slide bar the red line on the plot on the image side of the GUI moves (at the moment you release the mouse button). In this way you can "translate" reference layer units into spectral units. As you move this slide bar the numbers in white boxes next to the Map radio button will change, but you cannot change the numbers by typing directly in the boxes.

Input data

The data for these velocity features can come with different units on the third axis: frequency, wavelengt or velocity.

When the data are in frequencies or wavelength the velocities are computed using the non relativistic Doppler effect. When the data are in velocities the selection of the reference layer just shifts the zero and the velocities are directly read from the cube.

PV diagrams

The *Axis mode* introduced above produces PV maps. To compute such a map, you need to provide the following:

- An axis along which to extract the individual spectrum. This is done by dragging the mouse on the cube on the image side. You must click on the starting point of the line, move the mouse to the end point (without holding the mouse button pressed) and click again. The line appears in green on the cube.
- A reference layer in the cube to define the velocity zero point. This is done by choosing the displayed layer with the slide bar at the bottom right of the working side.

The line can be considered a slit with a default width of one spaxel. This width can be modified by filling the dedicated field in the parameter section of the panel.

What the task actually does is to create an average spectrum for the spaxels along the slit you set: for a slit width of one, the spaxels selected are those that the green line actually goes through; for a width of two, an additional half spaxels either side is selected, this meaning that the spectra from these spaxels are selected but the intensities are weighted by 0.5; and so on. It is then from this averaged spectrum that the PV diagram is created. Hence, a wider slit will increase the signal-to-noise ratio of the spectrum the PV diagram is constructed from.

For data given in wavelentgth, the velocities are computed using the usual formula $v = c\,\Delta\lambda / \lambda_{ref}$. For data in frequencies they are computed using $v = c\,\Delta\nu / \nu_{ref}$ and are given in m/s.

The computation of the PV diagram is done by clicking on the Compute Velocity Map button. This will produce an image such as the one shown below. For Axis mode the horizontal axis is offset from the left side of the slit you drew (or offset from the top if the slit is directly up-down) and the vertical axis is velocity on the left and colour scale on the right. For Map mode the two axes are the spatial axes of the cube (in WCS units) and colour indicates velocity, with a colour bar on the right showing the scale, which runs from maximum to minimum velocity.

Note

If the PV diagram looks too small it may be due to the zoom settings. Try panning out or zooming-to-fit using the magnifying-lens buttons below the PV diagram. If it is too dark, edit the cut levels (right-click on the PV diagram itself).

Note that currently you can only zoom on both axes at the same time; later we will allow for zooming on each axis independently. Also note that the PV diagram is constructed from the whole cube that is currently shown on the image side, i.e. over the whole spectral range you have. Therefore, it is likely you have a lot of velocities that are very large numbers! We are redesigning the GUI to allow you to select a smaller spectral region from which to make a PV diagram, but currently if you want to do this, you need to first create a new cube from a small spectral region (the Spectral Range Extraction menu item) and rerun the GUI on that cube.

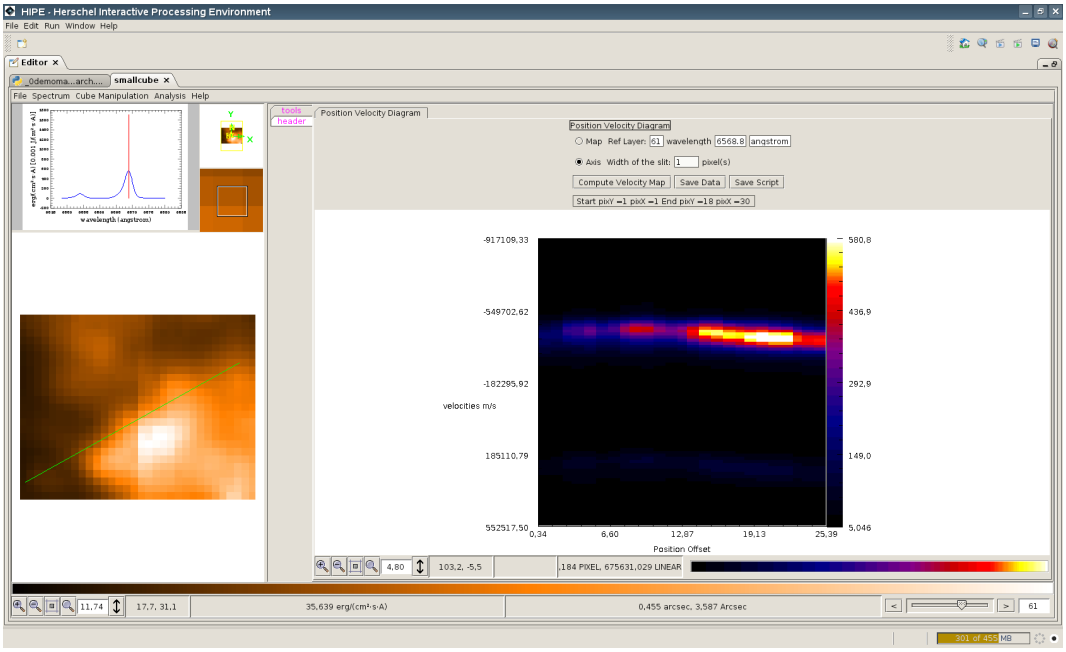


Figure 13.11. Position-velocity diagram

Once you have created the PV diagram, you may wish to adjust the properties of the diagram. You do this in the familiar way, by right-clicking on the diagram. You are then offered the options to edit cut levels, edit colours, zoom, annotate, create a screenshot (JPG), print and flip the Y-axis.

As with all other tabs on the working side, you can save data in FITS format, and the data is returned to HIPE as a `SimpleImage` with a WCS containing velocities and offset on the axis.

Note

A Save Script button will soon be available. This button will save a script containing the sequence of commands that produced the PV diagram.

The graphical interface uses a task which reshapes the result of the PV diagram in order to obtain an aspect ratio compatible with the display; in the saved script, calls to this task will be commented in order to have the "raw" shape as output. This task is not part of the CSAT, so it won't be explained in this document.

13.4. Running the tasks outside the graphical interface

Here we tell you how to run the tasks called by the GUI outside of the GUI itself. It is also possible to access the products that the cube tool creates, e.g. when you selected the spectrum from a single spaxel, via typed commands in the HIPE console.

13.4.1. Accessing the individual products

As you perform activities in the cube tool (e.g. select spectral or spatial regions) the results are held in tabs on the working side, but they are also held as new products that you can access from the HIPE command line or GUI.

- As you do things with the CSAT, new products are created and are listed in the HIPE Variables panel, with names similar to `singlepixspectrum` (and then `singlepixspectrum1` for the next selection, and so on). These should appear no matter how you started the cube tool.
- In addition, your creations are (supposed to be) stored in one of two separate products, also listed in the HIPE Variables panel, that were created when you started the cube tool. If you started with click-selection then the product is currently called `cat`, while if you started from the command line then it is called `mycuberesults`. *However, currently the `cat` variable contains nothing useful and should be ignored.*

The advantage of this is that you can access your cube tool creations outside its GUI. As with anything listed in the Variables panel in HIPE, you can inspect these new products by right-clicking and choosing one of the viewers offered. We recommend these for inspecting the data visually, rather than trying to produce plots from the command line. The reason is that the syntax that tasks such as `PlotXY` use on the cube tool products is still under consideration and will change with time.

You can also access these products from the command line with the following syntax:

```
singlespectrum = mycuberesults.getSinglePixelSpectrum()
```

Here you are extracting into `singlespectrum` the result of the last single spaxel spectral selection that you did in the cube tool. The data type of `singlespectrum` is `Spectrum1d`. For each of the CSAT products there is a different command to retrieve it. This is explained in the table below: on the left is the command (the one after `mycuberesults.` in the above example), the middle column is the data type of the product, while on the right-hand side there is the CSAT command that created the product.

Table 13.1. Syntax for extracting cubetool-products from the command line

getSinglePixelSpectrum	Spectrum1d	single spaxel spectrum display
getAvgspectrum	Spectrum1d	region spectrum display
getRangeExtractedCube	SimpleCube	range extraction
getIntegratedMapImages	ArrayList of SimpleImage	integrated maps
getVelocityAxisImage	SimpleImage	position-velocity diagram
getVelocityMapCube	SimpleCube	position-velocity diagram

13.4.2. Details for specific tasks

This section is for those who may wish to incorporate the CSAT in their own scripts or call up individual tasks that the GUI otherwise runs for you. Here we show you the calling syntax and I/O structure. Note that in almost all cases the units of the spectral dimensions are not wavelength or velocity but layer/channel (i.e. array location).

13.4.2.1. Single spaxel selection

To extract the spectrum from spaxel (4,5) use:

```
myspectrum = extractSinglePixelSpectrum(simplecube=mycube,posX=4,posY=5)
# or
myspectrum = extractSinglePixelSpectrum(simplecube=mycube,posX=4,posY=5).spectrum
```

where

- `mycube` is in `SimpleCube` format
- PosX,Y are the X,Y coordinates of the spaxel

The first command creates output in `Spectrum1d` format, with metadata taken from the input cube. The second command creates a spectrum of `Double1d` without metadata. The spectrum has columns of flux, weight, flag, segment (segment for now is just a placeholder, its value everywhere here is one) and wavelength, this latter being in the same unit that your `SimpleCube` had.

A faster way to run the second command after the first one is the following:

```
myspectrum1 = extractSinglePixelSpectrum.spectrum
```

As long as you have not run `extractSinglePixelSpectrum` since running it the first time, this method does not re-run the task on your cube but simply extracts out the result in a different format.

13.4.2.2. Multiple contiguous spaxel selection

To extract the average spectrum from the whole cube use (and see the instructions regarding creating `Spectrum1d` from the initial `Double1d` just above):

```
# output as a Double1d containing the flux
myspectrum2d=extractRegionPixelSpectrum(simplecube=mycube,wholeImg=True)
# for Spectrum1D format for the output type then after that
myspectrum1d=extractRegionPixelSpectrum.finalspectrum
# or just type
myspectrum1d=extractRegionPixelSpectrum(simplecube=mycube,wholeImg=True).finalspectrum
```

To extract an average spectrum from a region you need to make a Double2d array with columns of [X,Y,weight], to indicate which spaxels to select, and starting with entry [0,0,0]. Weight will determine by what fraction the spectrum from each X,Y will be multiplied in the average, i.e. Can be considered to be an area-weight. Assuming that this array has the name *foo* below:

```
# output as Double1d
myspectrum=extractRegionPixelSpectrum(simplecube=mycube,wholeImg=False,posArray=foo)
# or Spectrum1d, type just after that
myspectrum=extractRegionPixelSpectrum.finalspectrum
# or only type (on a single line)
myspectrum = extractRegionPixelSpectrum
(simplecube=mycube,wholeImg=False,posArray=foo).finalspectrum
# and you can also see the effective area in spaxels you have extracted
totalWeight=extractRegionPixelSpectrum.totalWeight
```

13.4.2.3. Smoothing filters

This is a long sequence of commands, so we list here the commands and some explanation:

```
filt=FilterSpectrumTask()
filt.rawSpectrum = myspectrum
filt.spectralDimension = -”Physical meaning of the spectral axis”
# is a string
filt.spectralUnit = -”unit”
filt.sizeOfSpectrum = sizeOfSpectrum
# sizeOfSpectrum is an integer representing the length of the spectrum
filt.specIndex = specIndex
# specIndex is a Double1d, previously created, containing the spectral
#   values for the flux
filt.modelFilter = -"GAUSSIAN"
# model to use
filt.widthFilter = 10
# width of the filter in units of array/channel, not spectral units
filt.execute()

# and then
```

```
FilteredSpectrum = filt.filteredSpectrum
# is the Double1d array containing the filtered flux
MaxValueFitSpectr = filt.maxValue
# is a double
PosMaxFitSpectr = filt.maxPosition
# is an integer
```

The input is a pre-existing `Double1d` called `myspectrum`.

13.4.2.4. Spectral range selection

This is currently rather awkward to run in HIPE, but here are some developer-oriented example scripts.

Since this task can be used in two ways (selecting a sub-spectral range for the whole cube or extracting on a sub-spatial domain *and* a sub-spectral domain) we give here two scripts.

Spectral range extraction only:

```
rangeextraction=RangeExtractionTask()
rangeextraction.simplecube=mycube
rangeextraction.startIndex=200
rangeextraction.endIndex=600
rangeextraction.Crop = False
rangeextraction.perform()
# access the results
res1=rangeextraction.rangeCube   #result stored in a simplecube
errcode=rangeextraction.error
logmssg=rangeextraction.log
```

Spatial and/or spectral domain extraction:

```
rangeextraction=RangeExtractionTask()
rangeextraction.simplecube=mycube
rangeextraction.startIndex=200
rangeextraction.endIndex=600
rangeextraction.Crop = True
rangeextraction.Xmin = 1
rangeextraction.Xmax = 8
rangeextraction.Ymin = 3
rangeextraction.Ymax = 9
rangeextraction.perform()
# access the results
res1=rangeextraction.rangeCube   #result stored in a simplecube
res2=rangeextraction.specRangeCube # result stored in a spectralsimplecube
errcode=rangeextraction.error
logmssg=rangeextraction.log
```

13.4.2.5. Integration Map

To run the command line of the integration map you need to create two arrays of spectral values. The graphical interface helps you to do this, but if you know the structure of your cube you can easily prepare these arrays. The first array stores the starting indices of the integration domain, while the second stores the ending indices. For integrating only on one domain the two arrays contain only one element each.

Here is an example of a call to this integration map task for the extraction of three integrated maps:

```
integr=IntegrateMapFromCubeTask()
integr.cube=mycube
#startarray and endarray store the indices of the layers, not their spectral values

startarray = Double1d(3)
startarray.set(0,10)
startarray.set(1,400)
startarray.set(2,800)
```

```
endarray = Double1d(3)
endarray.set(0,60)
endarray.set(1,600)
endarray.set(2,860)   #i.e. here assuming that the depth of the cube is >  860

integr.startArray =startarray
integr.endArray =endarray
integr.nbStack = 3
lstimages = integr.perform()

#results
print lstimages.size()  #return 3 if everything went well
#access the results:
image1= lstimages[0]   #image1 is a SimpleImage
```

The images returned by this task contain in their header information on the central spectral value and the bandwidth on which the integration was made.

- The bandwidth or size of integration is stored in the keyword BANDWIDTH.
- Depending of the details of the original cube, the central spectral value is stored in one of these keywords:
  - WAVELENGTH if the spectral dimension was wavelength
  - FREQUENCY if the spectral dimension was frequency
  - VELOCITY if the spectral dimension was velocity
  - SPECTRAL VALUE if there was no physical dimension available in the cube

13.4.2.6. PV Diagram

To run on the command line for Axis mode (Map mode will be documented at a later date) you need to make a list of the spaxels to be read into the task, with columns [index, X, Y, weight]. Index is the offset along the slit from the beginning, and if the X and Y are in order this will simply be 0,1,2,3... For slit widths >1 all the spaxels of one "column" have the same index. For example, your "list" can be: [0;4;0;0.5] on the first line, [0;5;0;1] on the second, [0;6;0;0.5] on the third... Then you run the commands:

```
# for output of type Double3d
velocityMap =
positionVelocityDiagram(simplecube=mycube,axis=True,coordSlitArray=list,
   widthSlit=1,nbpixelsAxis=15,referenceLayer=200)
# for output of type SimpleCube you then type
cubevelocitymap=positionVelocityDiagram.cubeVelocityMap
# and to access other parts of the creation
velocityMapAxis = positionVelocityDiagram.velocityMapAxis # Double2d
velocityMapAxisProd = positionVelocityDiagram.velocityMapAxisProd # simpleImage
```

`nbpixelsAxis` is the length of the slit (for map mode it should be ignored), *not* the total number of spaxels to be read. `referenceLayer` is the reference layer along the spectral axis (note, not in spectral units) that sets 0 velocity.

Chapter 14. HowTo Fit Spectral Features

Spectral features (baseline, lines and noise) are fitted using the *spectrum fitting toolbox* in the HCSS.

The data that is used by toolbox can be any Java or Jython object, as long as it implements the SpectralSegment interface (e.g., extracted from a Spectrum1d object). An example of a SpectralSegment could be the spectrum from one subband of the HIFI WBS spectrometer.

14.1. How to fit spectra in HIPE

1. Select the spectrum to fit from the variable list and then double click on the task fitSpectrum (Figure 14.1).

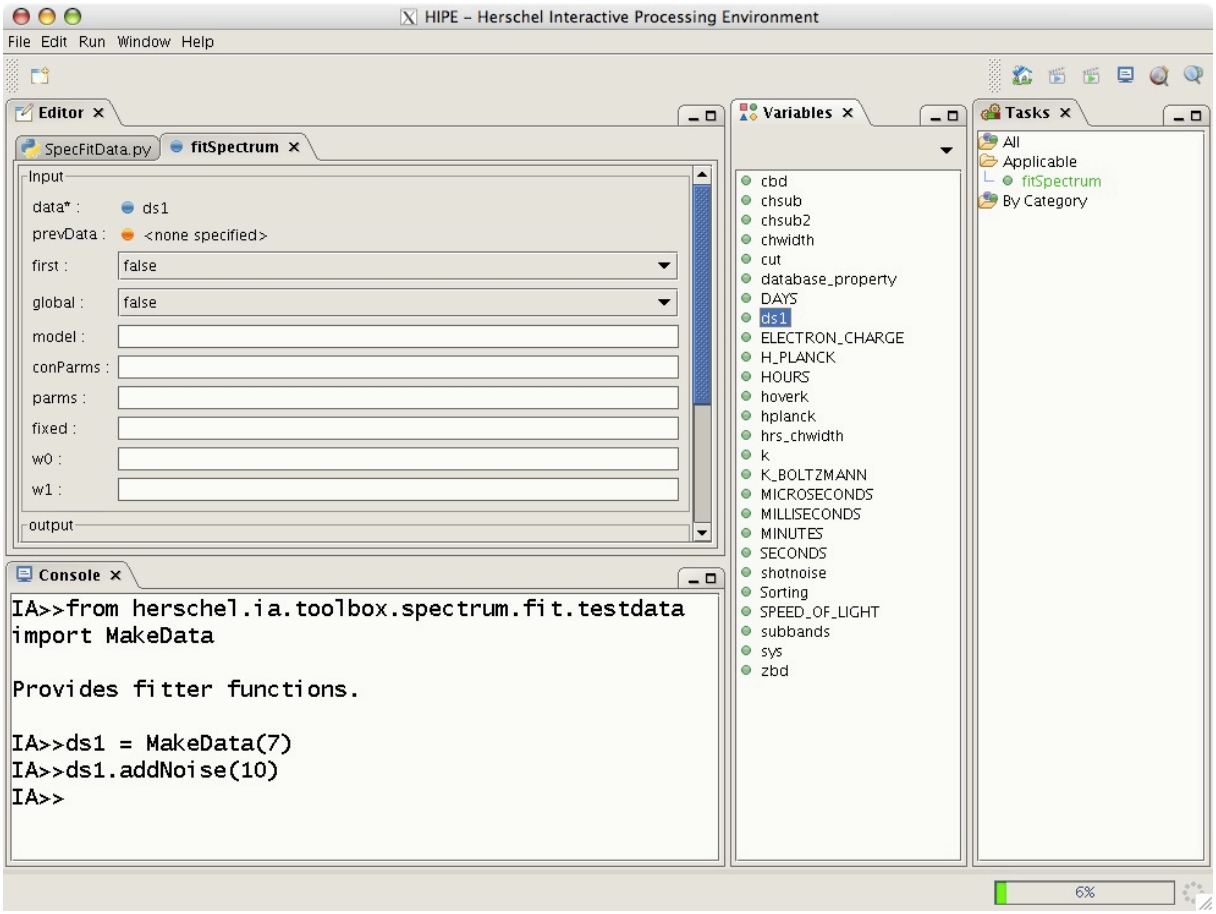


Figure 14.1. Starting fitSpectrum. Select "fitSpectrum" from the task list. This can most easily be found in the "Applicable Tasks" folder.

2. Apply a model to the spectrum via the GUI that pops up.

   All 1D models that are in ia.numeric.toolbox.fit can be selected from the drop-down box 'Use model'. The default is a Gaussian ("gauss") model, for which the 'Height of peak', 'X-Position of peak', and 'Width (sigma)' must be defined. The height and width have the value '1' already filled in, supply a value for the position and click RUN (Figure 14.2).

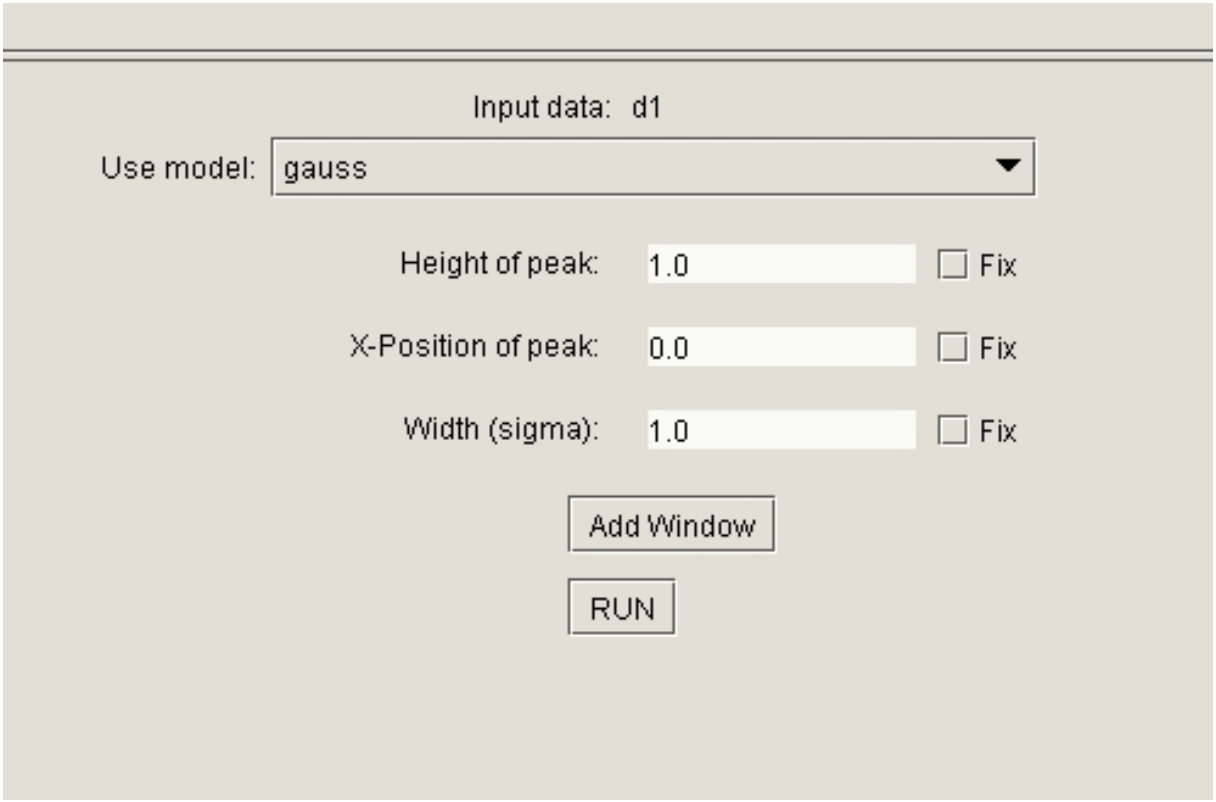


Figure 14.2. The Gaussian fit GUI. Supply the Gauss fit parameters.

A Lorentzian ("lorentz") model can be fitted in a similar way. If you select a polynomial ("poly") fit, then only the order of the polynomial needs be defined. The 'fix' tickboxes can be used to fix the value of the parameters (Figure 14.3).

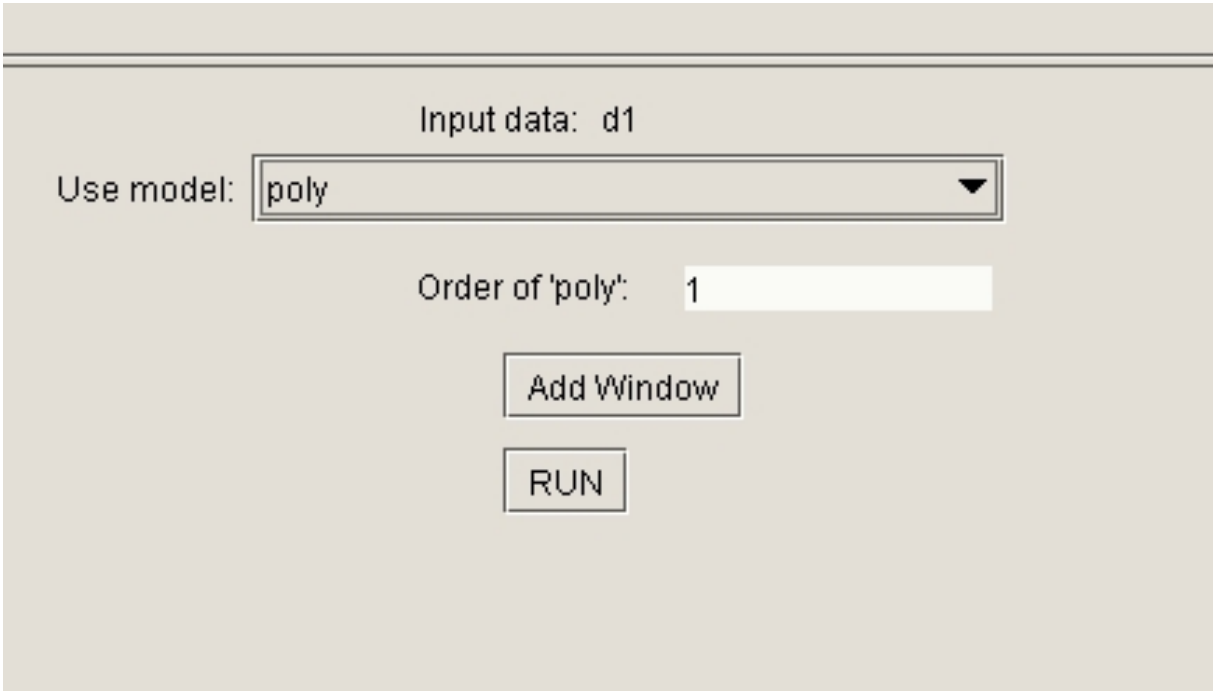


Figure 14.3. The polynomial fit GUI. Supply the order of the polynomial fit.

If any other model is selected, the GUI will look like Figure 14.4 . If the model requires constructor parameters (see the "DP Basic User's Manual" for details of all available models' parameters), the 'Constr Parms' field must be filled with a comma-separated list, for example, the "power" model needs a degree. If no constructor parameters are needed, as for the "sinc" model, leave the field empty. In the 'Parameters' field the fit parameters must be filled in with a comma-separated list - be sure to give the correct amount of parameters (see the JavaDoc). In the 'Fixed' field the parameters that must be fixed can be listed in (guess what) a comma-separated list. If the first and third parameter must be fixed, fill in: 0,2.

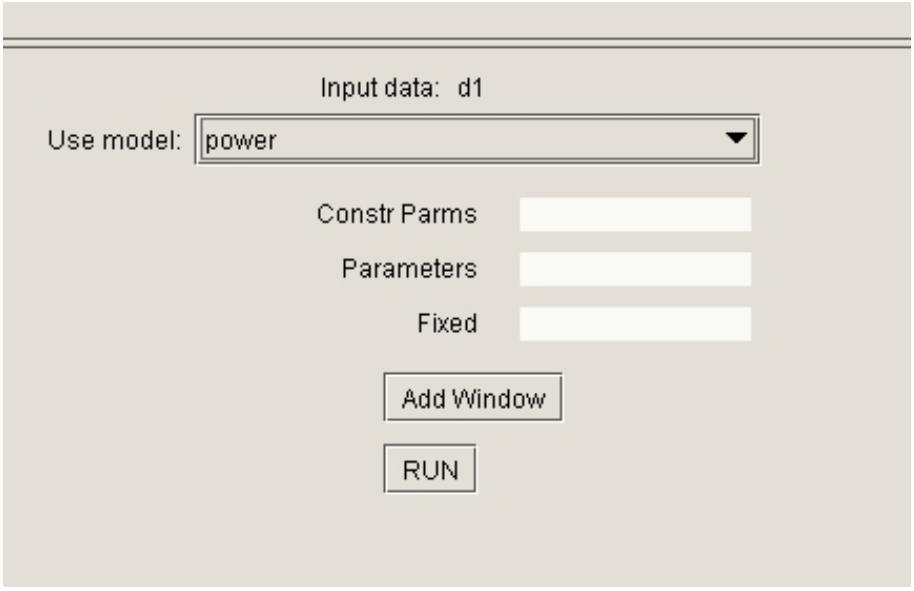


Figure 14.4. Other fit models. Supply all the model fit parameters.

3. A fit can be applied over a specified range.

   Click on 'Add Window' to define the fit X-range (between 'from' and 'to'). Up to five ranges can be defined by clicking on 'Add Window' again (Figure 14.5).

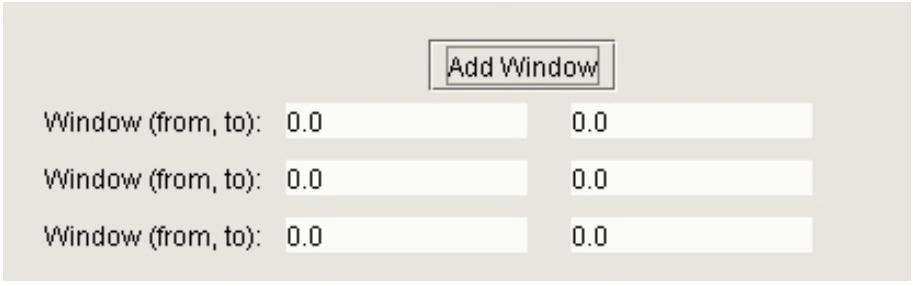


Figure 14.5. fitSpectrum can be applied over a specified range. Set the fit X-range

4. The result of a fit is a "fitResult" variable. Double clicking this variable opens a view window with the data and fitted model (top) and the residual (bottom). A table of the fitted parameters and their standard deviations can be seen by then clicking on the "ShowFitResult" task (Figure 14.6). If the fitResult contains several models, the parameters for all models are listed here.

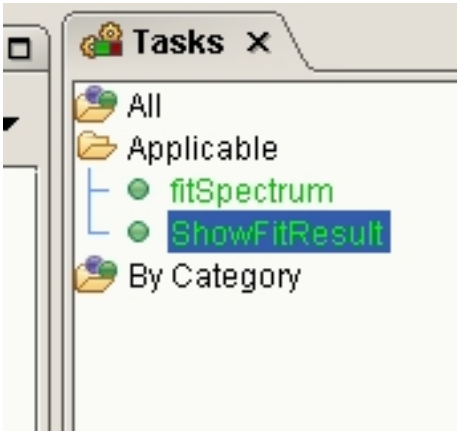


Figure 14.6. Tasks available after "fitSpectrum". Click on "ShowFitResult" to see fitted parameters and their standard deviations.

5. Another model can be applied to the result of a fit. So you can, for example, fit the baseline and then fit a spectral feature.

   Click on a 'fitResult' and then again on the 'fitSpectrum' task and follow the proceedure given above. This results in another fitResult variable to which you can apply another fit model, and so on.

6. Once satisfactory models for all spectral features have been found, all the models can be applied to the original data.

   Click on your final fitResult variable and then click again on the fitSpectrum task. The GUI contains a checkbox 'global fit' (Figure 14.7), check this and click on 'RUN'. No new model can be added at this stage.

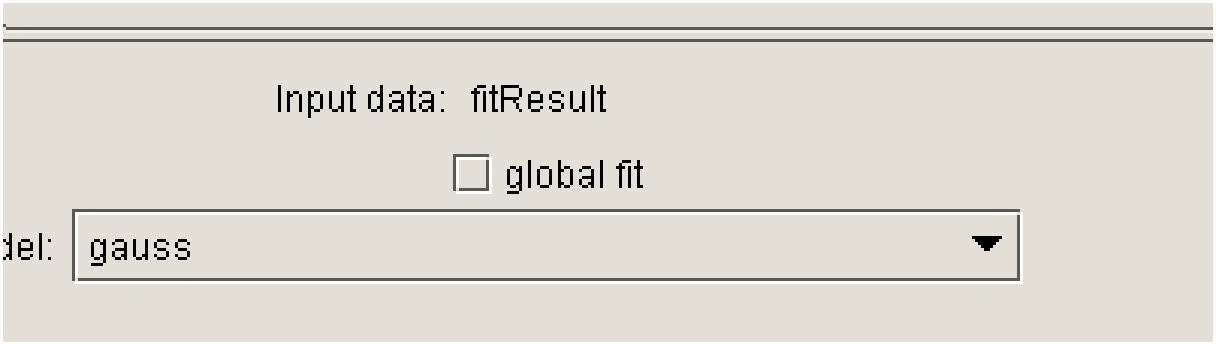


Figure 14.7. Global fit. Check global fit to apply all models to original data.

14.2. How to fit spectra from the command line

1. Download the toolbox into the session, note that in JIDE it is called SpectrumFitter rather than fitSpectrum!

```
from herschel.ia.toolbox.spectrum.fit import SpectrumFitter
from herschel.ia.toolbox.spectrum.fit.testdata import MakeData
```

   For demonstration purposes, we will use `MakeData` to create some test data to fit.

```
data=MakeData(7)
data.addNoise(10)
# Instantiate the fitter
sf=SpectrumFitter(data)
```

A plot window should look similar to that shown in Figure 14.8.

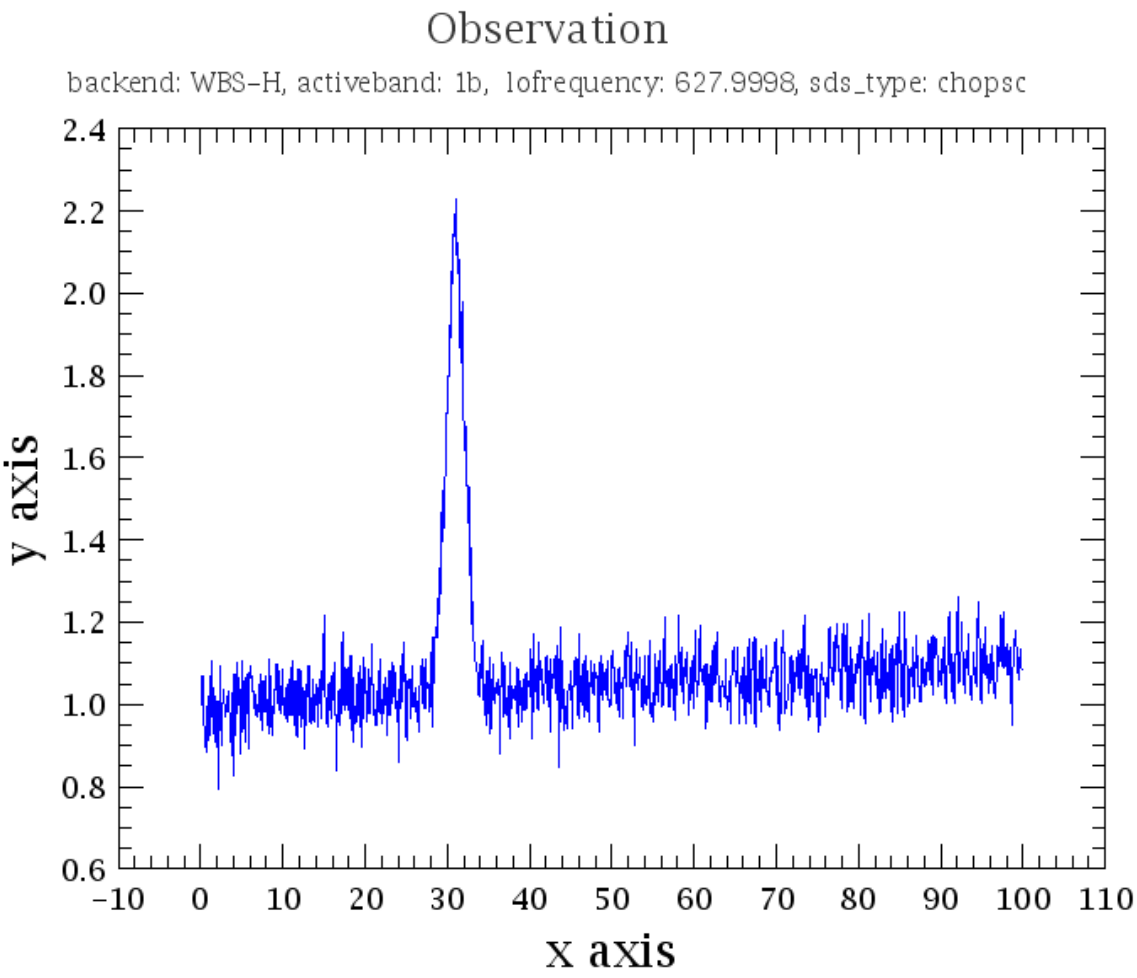


Figure 14.8. Test data to fit. Start the SpectrumFitter

2. The SpectrumFitter is an interactive tool and is best used in conjunction with the SpectrumModel tool, which allows you to select (and change) models and fitting parameters. The three models you are most likely to use are Gaussian, Lorentzian and Polynomial; the model fits, their parameters, and their usage in the SpectrumFitter tool are summarized in Table 14.1:

Table 14.1. Model fits, their parameters and usage in the SpectrumFitter tool

Model	Mathematical fit	Parameters	Usage
Gaussian	$f(x) = a_0 \, exp\left\{\frac{-(x-x_0)^2}{2s_0^2}\right\}$	a_0 = amplitude of line	`sf.addModel ('gauss', [a0,x0,s0])`
		x_0 = location of line peak	
		s_0 = width of line (sigma)	
Lorentzian		p_0 = amplitude of line	

Model	Mathematical fit	Parameters	Usage
	$f(x) = p_0 \left[\frac{p_2}{(x-p_1)^2 + p_2^2} \right]$	p_1 = location of line peak	`sf.addModel ('lorentz', [p0,p1,p2])`
		p_2 = half width at half maximum of line	
Polynomial	$f(x) = c_0 + c_1 x + ... + c_n x^n$	n = order of polynomial	`sf.addModel ('poly', [n], [c0,c1, ..., cn])`
		c_0 .. c_n = polynomial coefficients	

Note that you must know (roughly) where you expect a spectral feature in your data to be, in addition to its expected shape and approximate shape parameters. So, an initial guess is required - if this guess is completely wrong you may end-up fitting noise rather than your spectral lines.

Now, fit first the baseline with a polynomial and then fit the line with a Gaussian.

```
# First the baseline
# Apply the model
model=sf.addModel('poly', [2],[0,0,0])
# Do the fit
sf.doFit()
# Inspect the residual after the baseline is removed
sf.residual()
# Keep the fit
sf.fitOK()
# Now the line
sf.addModel('gauss', [1.0,30,0.1])
sf.doFit()
sf.residual()
sf.fitOK()
```

These steps result in the plot below. A black line (not seen here) displays the model and is replaced by a green line showing the fit (the Gaussian model here). The red line is the final fit for the entire spectrum. The residual is shown in a separate plot.

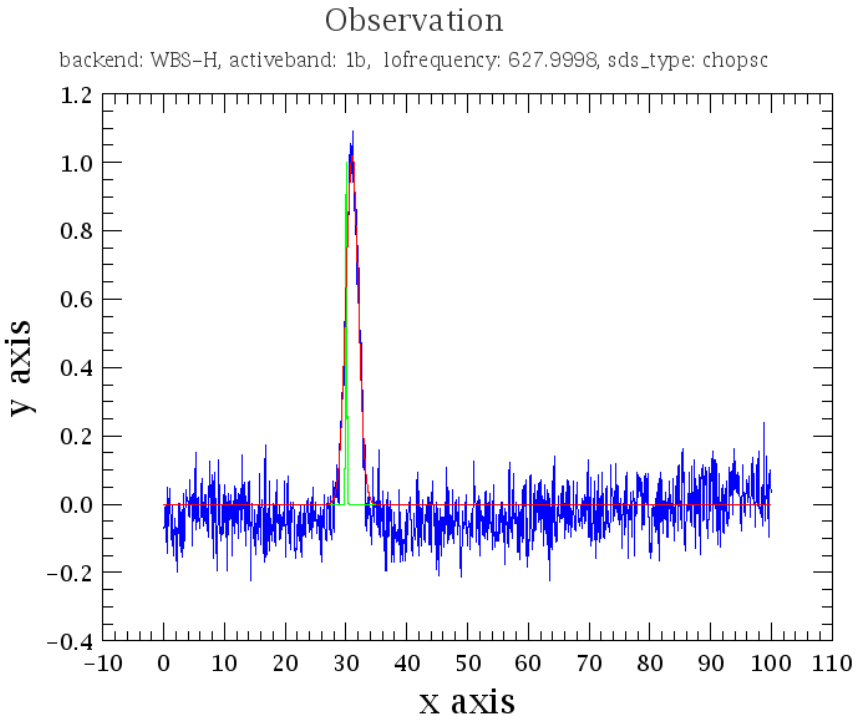


Figure 14.9. Fit result. Fit results for spectrum

3. It is possible to do both fits at the same time, globally, since the instance of our SpectrumFitter remembers what it has done so far.

   ```
   sf.doGlobalFit()
   ```

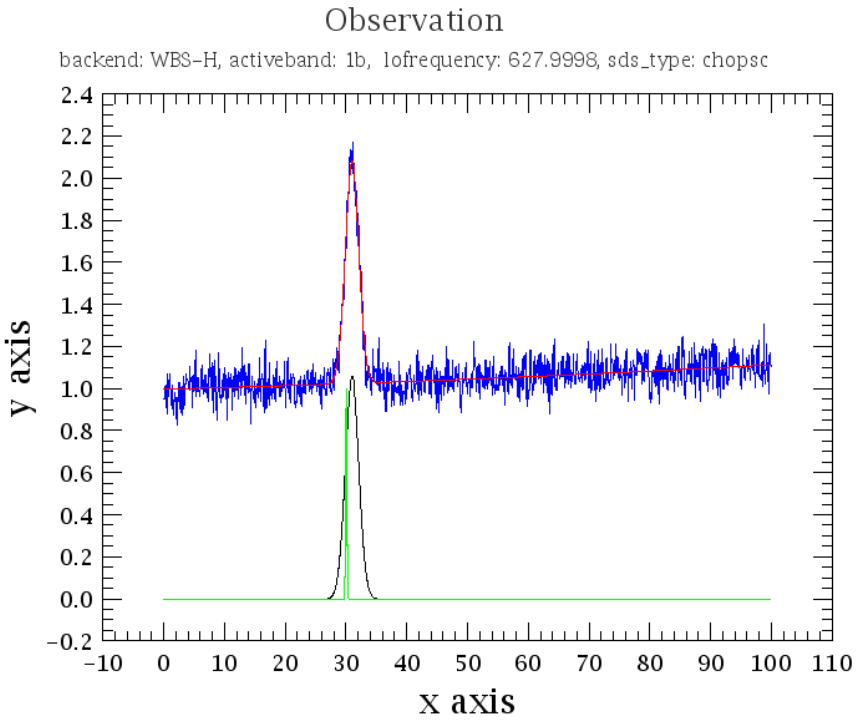


Figure 14.10. Global fit. Use the models together in a global fit

4. It is also possible to mask data. The following will do a polynomial fit only using data from 0 to 20 and from 40 to 100.

```
model=sf.addModel('poly', [2],[0,0,0])
# After you've created the model, add the masks.
model.setMask(0, 20)
model.setMask(40, 100)
```

   To best see how this works, include this masking in the example given above.

   After you have added a mask, you can also *remove* and *invert* it. You do this with the `unsetMask` and `invertMask` methods, respectively, as shown in the following code:

```
# Remove the first mask set in the previous example
model.unsetMask(0, 20)
# Invert the remaining mask
model.invertMask()
```

   After the above code, the mask will cover the whole spectrum *except* the (40, 100) interval.

5. The fitted model parameters and their standard deviations are printed to screen with:

   `print sf`

6. It is possible to manipulate the models produced by SpectrumFitter in various ways:

   - If you wish to change the initial parameters of any of the models (`model = sf.addModel(...)`), use setParameters:

     `model.setParameters([...])`

A new fit will be made on the fly.

- There are two ways to remove models:

  ```
  sf.removeModel(m)
  ```

  Or:

  ```
  m.remove()
  ```

- Subtract the model from the dataset:

  ```
  sf.subtractModel(m)
  ```

  This also removes the model from the fitter tool.

- Once you are satisfied with a fit, you can set the fitted parameters as the default for the models:

  ```
  m.useResults()
  ```

  This may be useful when using the same models for a following dataset.

- To apply them to a different dataset:

  ```
  sf.setData(otherData)
  ```

  Note that this replaces the data held in the SpectrumFitter with the SpectralSegment held in the variable 'otherData'. Once again, the fit will be redone on the fly.

Chapter 15. HowTo Display and Manipulate Images in HIPE

Herschel Editorial Board

15.1. Introduction

All image display tasks work on a `SimpleImage` that can be derived from a FITS file import (see HowTo chapter on FITS and ASCII input/output) or even from an image file such as a JPEG -- which is what we will use for illustrative purposes in this chapter.

Images can come with associated flux information (in header or, in Herschel DP, meta data). The flux information/units can also be applied to a given image by hand.

Images either have a World Coordinate System (WCS) stored in the meta data information, or a WCS may be applied. This chapter will also include information on the WCS parameters that can be found or applied to a given image.

Throughout this chapter illustrations are given from the *Full Work Bench* perspective.

15.2. Creation of a `SimpleImage` for Display

The `SimpleImage` format data is the standard map/ image data format that comes from the pipelining of Herschel data following standard pipeline processing. Images downloaded from the Herschel Science Archive are in this format. The following short script can be adapted to create a `SimpleImage` from any JPEG file and associate a very simple WCS to it. The following can be copied and pasted into the Editor view after opening a Jython script window, or copied into the Console view and run from there.

```
# Create some fake WCS information
myWcs = Wcs(crpix1 = 29, crpix2 = 29, crval1 = 30.0, crval2 = --22.5, \
 cdelt1 = 0.00028, cdelt2 = 0.00028, ctype1="RA---TAN", ctype2 = -"DEC--TAN")
# Create a SimpleImage with WCS in it
myImage2 = SimpleImage(wcs = myWcs)
#Put the image into the SimpleImage
# *.jpeg, *.jpg, *.tiff, *.tif, *.png, *.fits, *.fts or *.fit
# files are accepted.
importImage(image = myImage2, filename="directory name/ngc6992.jpg")
```

A SimpleImage called "myImage2" is created and is available in the "Variables" view (See Figure 15.1). It should be emphasised that it is possible to use ANY image created in an instrument pipeline for the examples given in this chapter.

The importing of the image is also possible via the "importImage" task available in the Tasks view list. Click on "myImage2" in the "Variables" view then double-click on the appropriate task, "importImage". A name can be typed in or a selection made by Browse...ing the system.

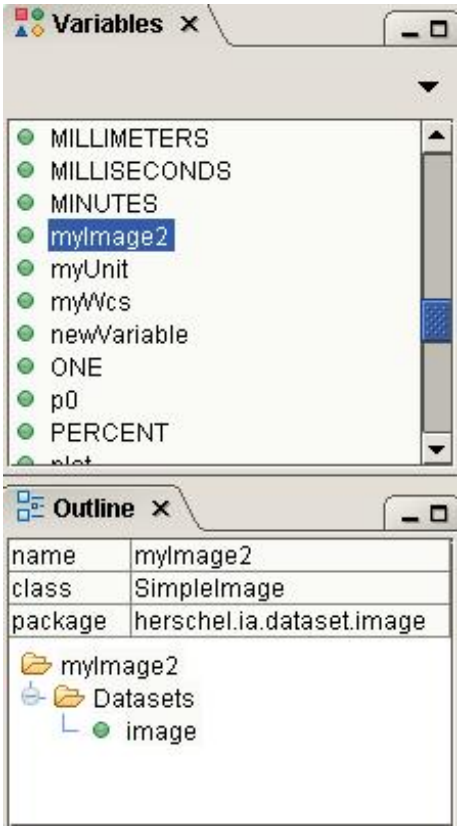


Figure 15.1. The Variables view shows the "myImage2" highlighted. A double click on this automatically brings up the image in a new Editor window (top left). In the Tasks view the folder "Applicable", when opened, shows the tasks that can be applied to this image.

Double-clicking on the variable "myImage2" in the Variables view will automatically display the image in a new Editor window. A single right click in the same place will indicate that this can be "Open(ed) with..." a Product viewer as well. This shows header (metadata) information for the whole image product, which can have a number of datasets. For the `SimpleImage`we have created for our example there is a single image dataset.

15.3. Viewing the Metadata and Array Data Associated with an Image Dataset

An image can have several datasets. For example, we can include a flag image dataset for flagging bad pixels (see "DP Basic User's Manual" for more information). Each of these datasets have associated metadata, which has the same role as header information in a FITS file. It indicates associated flux and coordinate information plus processing history (if appropriate) etc.

To view the metadata (and array data) associated with an image dataset requires opening a Dataset viewer. This can be done in two ways.

- First a right-click on your image variable name in the "Variables" view (e.g., on "myImage2"). A short menu including "Open With...." appears. Choose the product viewer. The product view is shown which includes some overview information/metadata plus a list of datasets (at the bottom of the datasets -- and could include a number of image layers). Do a right-click on one of the datasets to see the "Open With..." in the short menu. Select Dataset viewer.

- A single click selection of the image in the "Variables" menu list shows its outline in the "Outline" view. Opening the folder in the Outline view to see the datasets in it and right-click on a dataset to see the short menu with "Open With...." and the dataset viewer selectable.

Any of the above will provide a view of the metadata plus the data values of the array making up the dataset within a window in the "Editor" view. View of either the metadata or array data can be toggled using the arrows to the left of the metadata/array data names in the "Editor" window (see Figure 15.2).

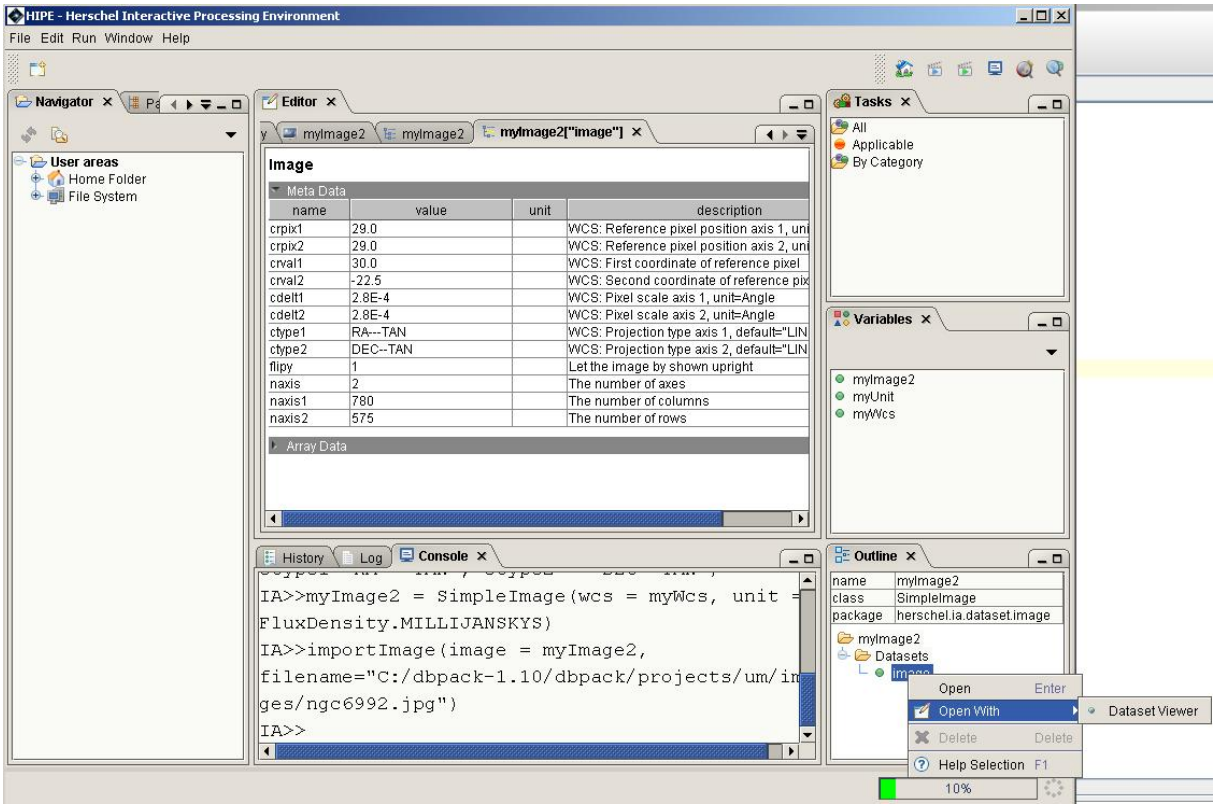


Figure 15.2. Metadata and Array data view using the Dataset viewer with an image.

15.4. A Simple Display of an Image

The simplest way to display an image in HIPE is to double-click the image name (e.g., in the "Variables" list). The default activity for this is then the display of the image in a new window in the "Editor" view (see Figure 15.3).

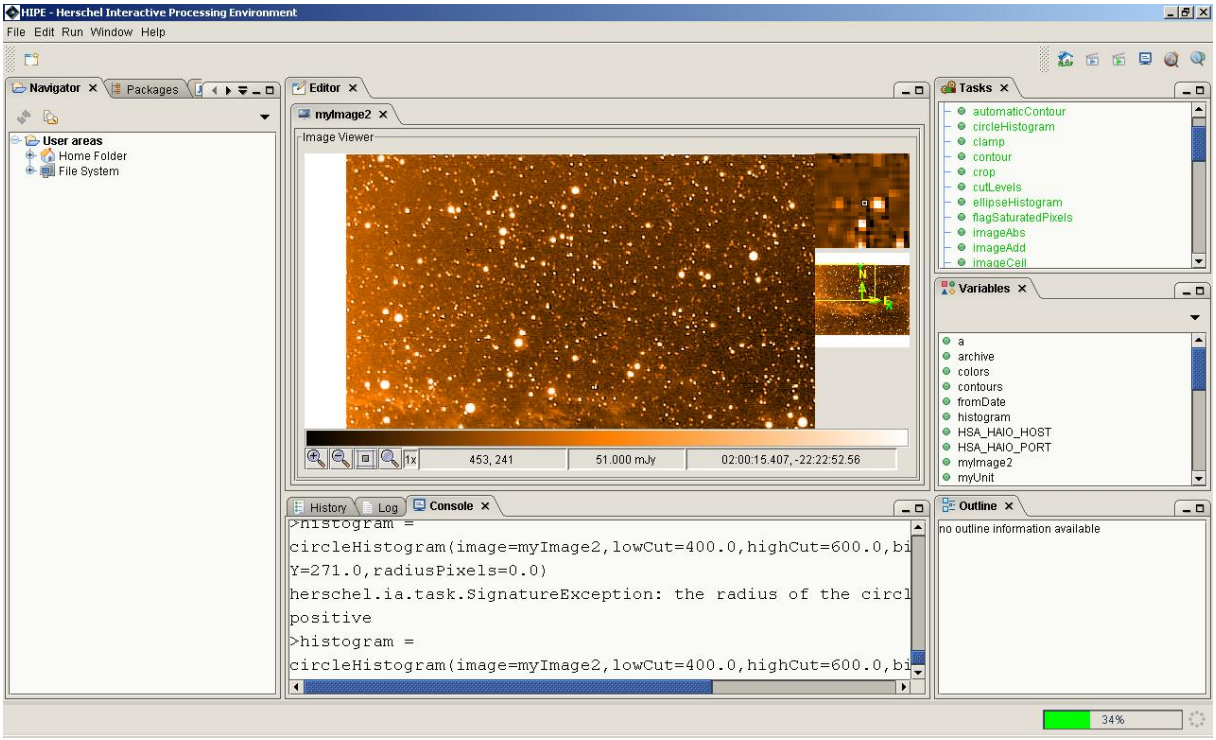


Figure 15.3. Automatic Display obtained via double-click of a SimpleImage variable appearing in the "Variables" view.

The display shows a zoom/pan image as the main image, plus two smaller images that show,

- A zoomed image is shown around the mouse position on the main image (at top right).
- An overview of the full image showing the zoom/panned position of the main display (at bottom right) outlined by a rectangle. This box also illustrates the directions N and E on the display based on the WCS coordiantes of the image. The position of the rectangular zoom/pan region can be adjusted by clicking on the box and dragging it to another part of the display. In Figure 15.3 the box has been dragged to the top left of the image.

15.4.1. Magnifying an Image

To bottom left of the view (see Figure 15.3) are a set of magnifying glass images that, in order left to right, zoom in, zoom out and go back to the original image size. In between the magnifying glass images is an icon with a small square surrounded by a box -- which allows an image to be displayed that fits the whole SimpleImage into the viewing area.

15.4.2. Image Coordinates and Pixel Intensity

The mouse position over the image is constantly updated at the bottom of the image displayed with both the pixel coordinates and the world coordinates (if a WCS is available in the SimpleImage being viewed) being presented to the right of the magnification icons.

In between the two pieces of coordinate information the pixel intensity for the pixel falling under the mouse position is also constantly updated (see again Figure 15.3) as the mouse is moved across the image.

15.5. Editing and Printing Images

We can edit an images in a number of ways. The following are available after doing a right-click of the mouse button while the mouse is over a displayed image (see Figure 15.4).

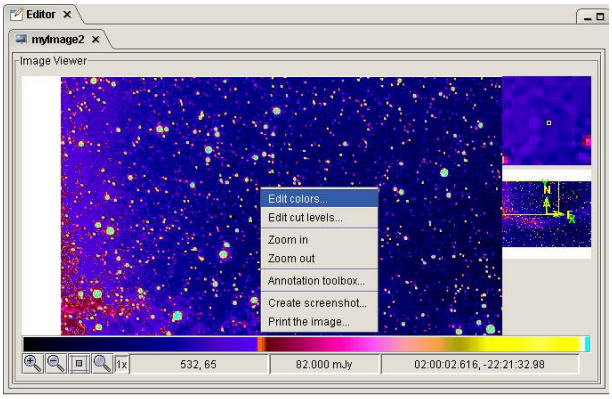


Figure 15.4. Edit functions available via a right-button mouse.

- Edit colors -- the colour lookup table can be adjusted to a number of different types plus linear/log/exponential scalings.
- Edit cut levels -- the cut levels for which the colour lookup table is to be applied can be adjusted for an image.
- Zoom in/out -- as per the zoom icons discussed earlier.
- Annotate the image -- an annotation can be placed at the position of a mouse click.
- Create screenshot/print image -- the displayed image can be saved to an image file or printed on a user-selected printer.

15.5.1. Editing the Colour Look Up Table (LUT)

The standard colour scheme for image display is for "Real" colours shown in a "Ramp" intensity with a "linearScale". Selection of the "Edit colors..." from Figure 15.4 displays the colour menu Figure 15.5. Hitting the "Reset" button always enables the default colour display.

To select any other colour scheme simply click on the colour type and/or intensity or algorithm to create a new colour scheme. The scheme is applied to the image immediately. The window (Figure 15.5) can be dragged away from the image.

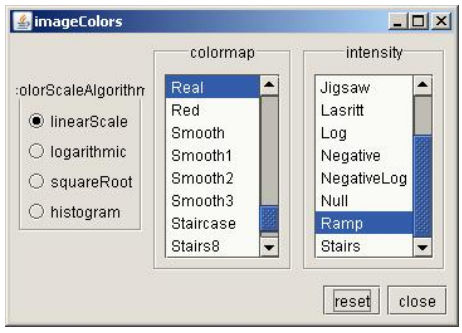


Figure 15.5. Colour table selection menu. Hitting "Reset" takes you back to the original colour table.

15.5.2. Editing the Cut Levels

The default cut levels for images is 99.5 per cent of pixel values. Selection of the "Edit cut levels..." from Figure 15.4 displays the a cut level selection including a histogram of the current pixel intensity values Figure 15.6. Hitting the "Reset" button always enables the default cut levels.

To select any other cut level the user can do one of two things.

- A button selection of cut levels (90, 95, 98, 99, 99.5 or 100 per cent). Note that selection of any of these will adjust the histogram display above.
- Adjustment of the upper and/or lower-level cutoffs of the histogram by click-and-drag of the yellow arrows (left or right) shown at either end of the histogram view.

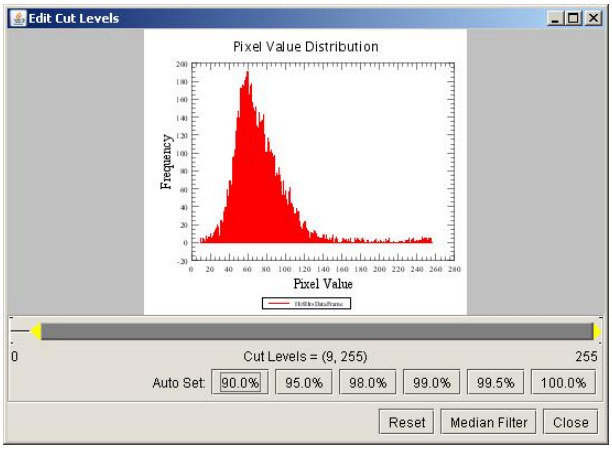


Figure 15.6. Cut level selection window. Hitting "Reset" takes you back to the original cut levels of 99.5 per cent.

15.5.3. Zoom In/Out

Selection of either of these provides an increase or decrease in zoom by a factor of 2.

15.5.4. Annotation Toolbox

The annotation toolbox is shown in Figure 15.7.

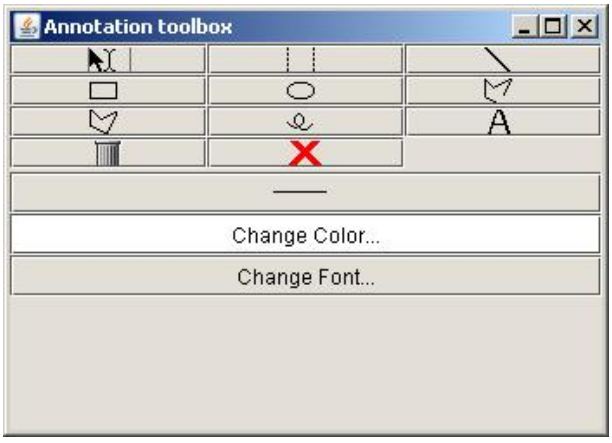


Figure 15.7. The annotation toolbox.

The icons in the annotation toolbox appearing in Figure 15.7 have the following usage (from left to right and from top to bottom):

- Select annotation
- Select all annotations in a region
- Draw a line
- Draw a rectangle
- Draw an ellipse

- Draw a polyline
- Draw a polygon
- Draw with the free hand on the image
- Add a text annotation
- Remove the selected annotation(s)
- Remove all annotations

Letting the mouse linger over an icon also displays its function.

The polygon and polyline methods will enable you to select points on the image which should be used as a corner of the polygon using the mouse. Double-clicking the mouse will end the selection procedure.

The three buttons below the ones already described change the view of the annotation. From top to bottom:

- Change the thickness of the line
- Change the colour of the annotation. The present colour of annotations is shown in the background.
- Change the font of the text annotation

Note

The *Select all annotations in a region* button only works when there are already annotations on the image. Pressing the button will select all the annotations which are in the selected region. This button can be used to change the colour or the line width of several annotations at once.

15.5.5. Screenshots and Printing Images

The last 2 possibilities within the image edit menu allows screenshots to be created in JPG, PNG or BMP format or a printing to a user-selected printer. The user is also given the choice of whether the image produced includes all overlays and annotations or not.

15.6. Image Transformations

Image representations can be adjusted in the following ways:

- Clamp: or clipping an image.
- Crop: extract a seubsection of an image.
- Clamp: or clipping an image.
- Rotate: rotate image by an arbitrary angle
- Scale: image rescaling in user-selected factors in X and Y.
- Translate: move positive or negative pixel or sky amount of image within the frame.
- Transpose: flip or rotate by n x 90 degrees

15.6.1. Applying Image Transformations

All image transformations can be applied in the same way. First, select a `SimpleImage` in the "Variables" view, then go to the "Tasks" view and select -- from the Applicable Tasks folder -- the appropriate image transformation (*crop, clamp, rotate, scale, translate* or *transpose*). To select one

of the transformation tasks, double-click on its name on the Tasks view. This will bring up a dialog for the task.

Dialogs work in a similar fashion for all image transformations. Options are presented in a pull-down menu (e.g., the form of the interpolation of pixel values when rotating an image, see Figure 15.8) or with an editable input such as the rotation in degrees and the option for the name of the output variable created following the transformation. Hitting "Accept" will run the task.

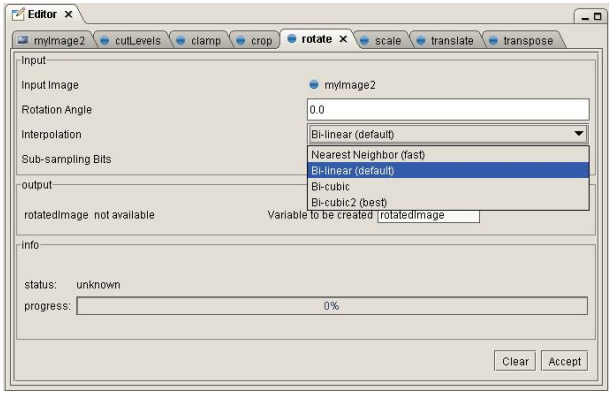


Figure 15.8. Example image transformation dialog. Rotating an image using the "rotate" task. Several interpolation options are available.

15.6.2. Image Transformation Options

For each image transformation there are a set of options for the user.

15.6.2.1. Clamp Options

This allows the floor and ceiling of an image to be set. Values above the max or below the min input by the user are set to the max and min values assigned by the user respectively.

15.6.2.2. Crop Options

A section of the image to be extracted to another SimpleImage. The range of X and Y pixel coordinate values are input by the user.

15.6.2.3. Rotation Options

When rotating the image, several types of interpolation are possible. By default, bi-linear interpolation is used. There are four types of interpolation possible.

- **Bi-linear [default]** -- *the default* - interpolates one pixel to the right and one below.
- **Nearest neighbour [fast]** -- direct pixel copying, the fastest.
- **Bi-cubic** -- uses interpolation via a piecewise bi-cubic polynomial.
- **Bi-cubic2 [slow]** -- variant of bicubic interpolation that can give sharper results than bicubic.

15.6.2.4. Scale Options

Allows for different magnification in X and Y pixel directions. Possible interpolation types are as for the "rotate" task.

15.6.2.5. Translate Options

Allows either X and Y pixel translations or sky translations (coordinates input as strings of the form "hh:mm:ss.s" and "dd:mm:ss.s") can be input by the user.

15.6.2.6. Transpose Options

Allows for different simple transpositions of images. The following transpositions can be done with this task.

- Flip vertical (flips top and bottom)
- Flip horizontal (flips from side to side)
- Flip diagonal (bottom left to top right)
- Flip antidiagonal (top left to bottom right)
- Rotate 90 degrees (clockwise rotation)
- Rotate 180 degrees
- Rotate 270 degrees

15.7. Image Arithmetic

Images can be arithmetically manipulated (scalar or pair-wise combinations) to provide changed versions of the original. In all cases, image arithmetic can be done by opening a dialog, filling in the dialog and then clicking "Accept" to run the task (e.g., Figure 15.9).

Possible arithmetic tasks are:

- Absolute value (imageAbs). To obtain the absolute value image from the input.
- Add/Divide/Multiply/Subtract (imageAdd, imageDivide, imageMultiply, imageSubtract). This allows either a scalar or a second image as the amount to be added/divided/multiplied/subtracted. The second image can be input into the dialog by click-and-dragging of it from the "Variables" view to the orange dot position in the dialog for the second image (see Figure 15.9) For images, the combination is by pixels or WCS reference.
- Exponent of the image. Including to the power N and 10 (imageExp, imageExpN, imageExp10).
- Log of the image. Including base 10 or N (imageLog, imageLog10, imageLogN).
- Image to the power n (imagePower).
- Image rounding (imageRound).
- Square and square root of the image (imageSquare, imageSqrt).

Most of the above are self-explanatory. One example is shown in Figure 15.9.

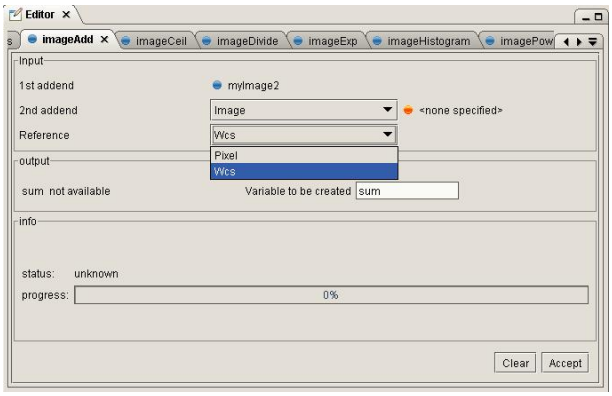


Figure 15.9. Example image arithmetic dialog.

15.8. Working with the World Coordinates System (WCS)

The WCS information for an image is stored in its metadata which can be viewed using the Dataset viewer.

The Wcs class enables the user to define a transformation between the pixel coordinates and world coordinates. The following illustrates how we can type in (at a command-line) a WCS. We create our Wcs() object which we then add to a fake `SimpleImage` we set up to start with.

```
i = SimpleImage()
i.image=RESHAPE(Double1d.range(200*300), [200,300])
# create a fake image 200x300 pixels in size

myWcs = Wcs() # set up the Wcs() object
myWcs.ctype1 = -"LINEAR"  # start adding things to it....
myWcs.cdelt1 = 5
myWcs.crval1 = 200
myWcs.cunit1 = -"K"
myWcs.crpix1 = 0

myWcs.ctype2 = -"LINEAR"
myWcs.cdelt2 = -.05
myWcs.crval2 = 2.0
myWcs.cunit2 = -"V"
myWcs.crpix2 = 0

i.wcs = myWcs  # apply the set of WCS information to our image
print i.wcs #to see the WCS of the image
```

Warning

The above code will generate an image with the value 200 assigned to the `NAXIS2` keyword and 300 assigned to `NAXIS1`. In other words, the image size will be 200 pixels along the *y* axis and 300 pixels along the *x* axis. The coordinate values will be displayed in this order (*y*, *x*) in the Image Viewer. For an explanation of why the *y* size comes *before* the *x* size, see section *A note on array ordering* in the *User's Manual*.

The above example will create a coordinate system, where the temperature and current are set for the axes. The x-axis is LINEAR (ctype1), has the central pixel in column 0 (crpix1), has a value of 200 in the central pixel (crval1), uses steps of 5 (cdelt1) and has as unit Kelvin. The y-axis is also LINEAR (ctype2), has the central pixel in row 0 (crpix2, this is the top of the image), has a value of 2 in the central pixel (crval2), uses steps of 0.05 (cdelt2) and has as unit Volts.

It is also possible to use the Wcs class to define transformations between pixel coordinates and sky coordinates. This can be done using the standard Wcs parameters. An example is given below. It also indicates how we can "set" WCS values in our WCS object :

```
wcs2 = Wcs()  # ❶
wcs2.setCrpix1(128)
wcs2.setCrpix2(128) # ❷
wcs2.setCrval1(101.676612741936)
wcs2.setCrval2(0.829427624677429) # ❸
wcs2.setCtype1("RA---TAN")
wcs2.setCtype2("DEC--TAN") # ❹
wcs2.setRadesys("ICRS")
wcs2.setEquinox(2000.0) # ❺
wcs2.setParameter("cd1_1", --1.9064468150235E-6, -"")
wcs2.setParameter("cd1_2", 3.39797311269006E-4, -"")
wcs2.setParameter("cd2_1", 3.39811958581193E-4, -"")
wcs2.setParameter("cd2_2", 1.580446989748E-6, -"") #❻
```

❶ A `Wcs` is created.
❷ The central pixel is set. In this case, the central pixel is at (128, 128).
❸ The value of the central pixel is set. In this case, the first central pixel is located at 6h46'42.387" and the second pixel at 0 degrees 49'45.94".
❹ The type of the axes is set. The first axis defines the right ascension (in a gnomonic projection) and the second axis defines the declination (in a gnomonic projection).
❺ The coordinate system is set (here, we use the standard ICRS type). The equinox is also set.
❻ The linear transformation matrix is set. This defines the pixel size and the rotation of the images.

For more information on the WCS see Chapter 4 of the "DP Basic User's Manual."

Chapter 16. HowTo Do Basic Image Analysis in HIPE

Herschel Editorial Board

16.1. Introduction to Interactive Image Analysis with HIPE

Basic image analysis described in this chapter involves the following tasks that are available within the HIPE environment.

- aperture photometry
- image/area histograms
- 1D profile plotting
- contour plotting and overlays

All tasks work on a `SimpleImage` that can be derived from a FITS file import (see HowTo chapter on FITS and ASCII input/output) or even from an image file such as a JPEG -- which is what we will use for illustrative purposes in this chapter.

16.2. Setup and Display of Images for Analysis

In the chapter on Image Display we note how to create image coordinate systems and how to formulate the `SimpleImage` format from external sources. `SimpleImage` format data is the standard map/image data format that comes from the pipelining of Herschel data during standard pipeline processing. Images from the Herschel Science Archive (HSA) are in this format. The following short script can be adapted to create a `SimpleImage` from any JPEG file and associate a WCS to it. The following can be copied and pasted into the Editor view after opening a Jython script window, or copied into the Console view and run from there.

```
# Create some fake WCS information
myWcs = Wcs(crpix1 = 29, crpix2 = 29, crval1 = 30.0, crval2 = --22.5, \
 cdelt1 = 0.00028, cdelt2 = 0.00028, ctype1="RA---TAN", ctype2 = -"DEC--TAN")
# Create a SimpleImage with WCS in it
myImage2 = SimpleImage(wcs = myWcs)
#Put the image into the SimpleImage
# *.jpeg, *.jpg, *.tiff, *.tif, *.png, *.fits, *.fts or *.fit
# files are accepted.
importImage(image = myImage2, filename="directory name/ngc6992.jpg")
```

A `SimpleImage` called "myImage2" is created and is available in the "Variables" view (See Figure 16.1). It should be emphasised that it is possible to use ANY image created in an instrument pipeline for the following tasks.

The importing of the image is also possible via the "importImage" task available in the Tasks view list. Click on "myImage2" in the "Variables" view then double-click on the appropriate task, "importImage". A name can be typed in or a selection made by "Browse..."ing the system.

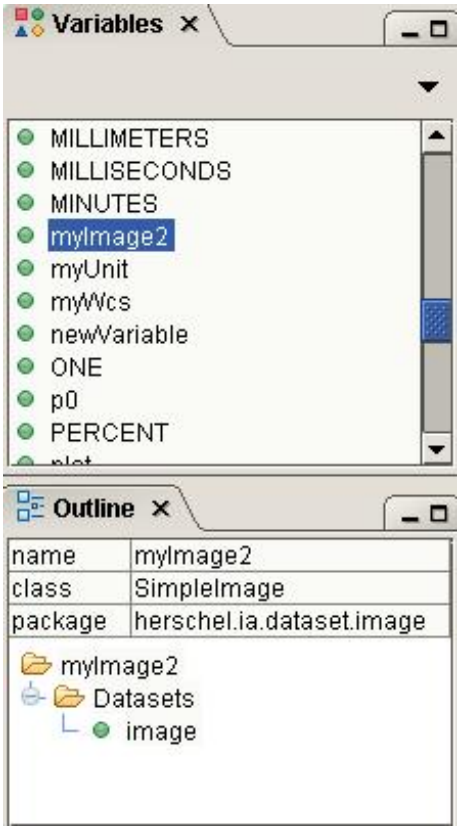


Figure 16.1. The Variables view shows the "myImage2" highlighted. A double click on this automatically brings up the image in a new Editor window (top left). In the Tasks view the folder "Applicable", when opened, shows the tasks that can be applied to this image.Variables view with `SimpleImage` variable highlighted.

Double-clicking on the variable "myImage2" in the "Variables" view will automatically display the image in a new Editor window. A single right click in the same place will indicate that this can be "Open(ed) with..." a Product display as well. This shows header information and the fact that there is a single image dataset in the `SimpleImage` product we have created.

The image appearing in the Editor view is displayed with the standard zoom/pan and editing capabilities associated with it that are discussed in the chapter "HowTo Create, Display and Manipulate Images."

16.3. Getting a `SimpleImage` a product out of the Herschel Science Archive (HSA)

When downloading a product out of the science archive we access images from an `ObservationContext`. An `ObservationContext` contains all the information associated with a single observation and its processing (including all associated calibration files). In a download (see chapter on HowTo Access Data) from the HSA we have products made available from several levels of processing at using the Herschel Science Center's Standard Product Generation pipelines.

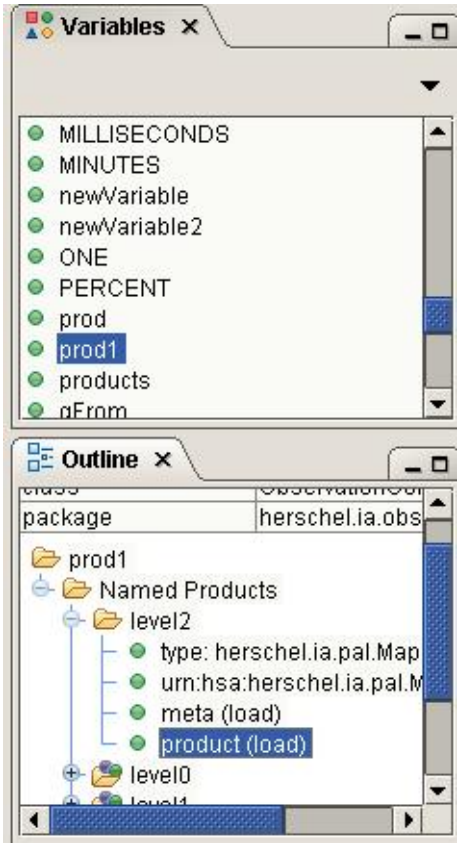


Figure 16.2. An `ObservationContext` called "prod1" has been obtained from the HSA. Clicking on the folders it contains in the "Outline" window allows us to get at the Level 2 product -- the final pipeline output for this observation.Contents of an `ObservationContext`

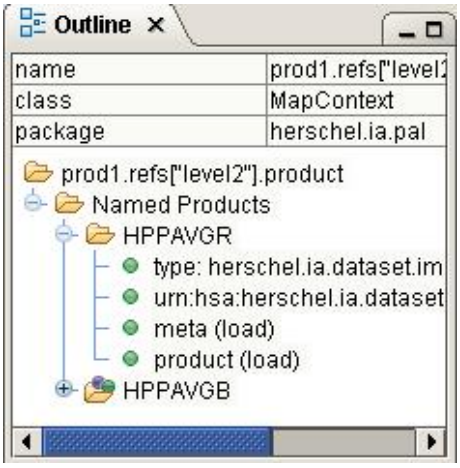


Figure 16.3. A double-click on the product highlighted in blue in Figure 16.2 provides this Outline view. A double-click on the product highlighted displays the image from the green channel of this PACS observation.PACS green channel image access

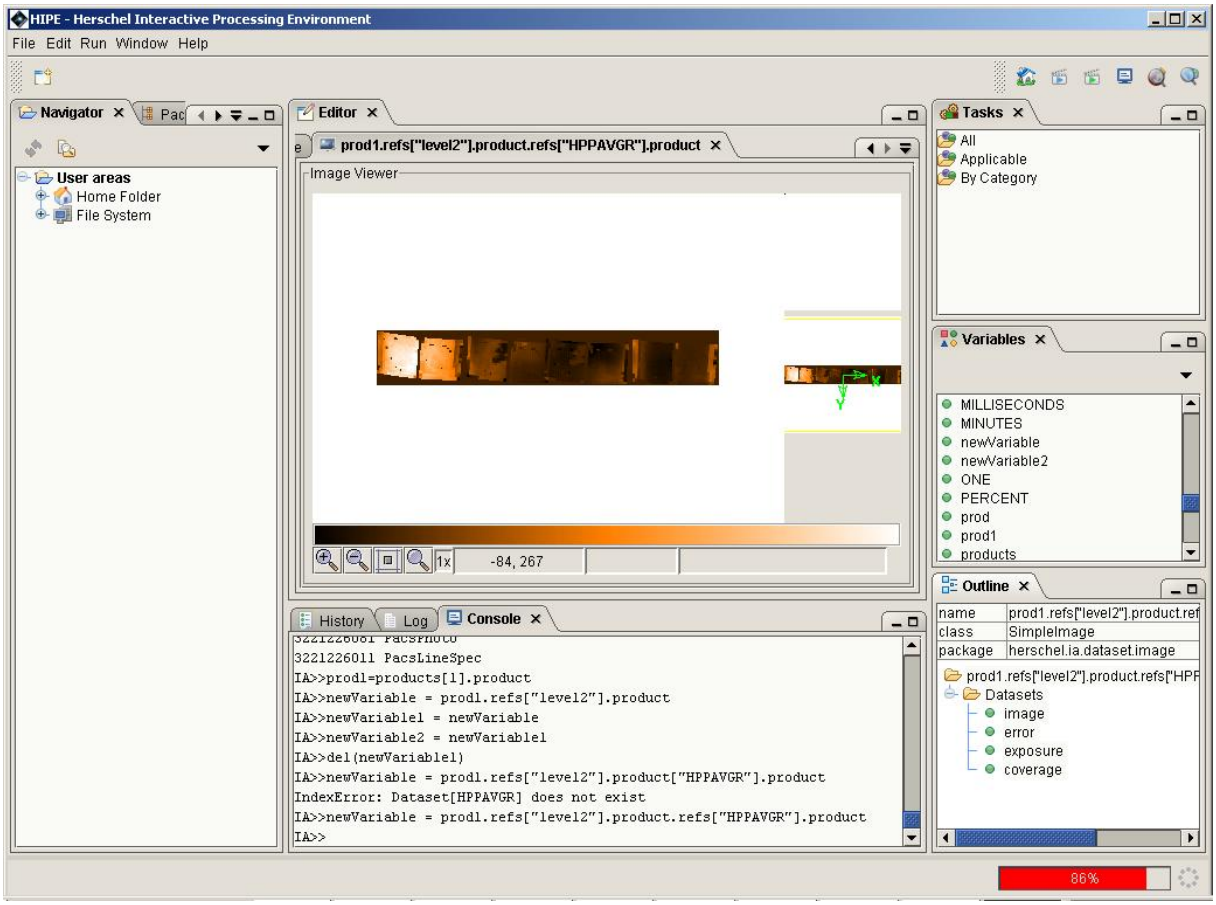


Figure 16.4. The PACS green channel image displayed in the full work bench of HIPE.

In the "Variables" and "Outline" displayed in Figure 16.2 and Figure 16.3 we see first an `ObservationContext` called "prod1" which is a PACS photometer test observation -- which has been expanded in the "Outline" view. A double click on the "Level2" product will show the outline of the final processed image (which contains two PACS images in two channels of the photometer taken simultaneously, a green channel and a blue channel). This is shown in Figure 16.3. We can also get the `SimpleImage` (e.g., name it "image1") by extracting it from the `ObservationContext`. The line below can do this from the command-line of the "Console" view.

```
image1 = prod1.refs["level2"].product.refs["HPPAVGR"].product
```

A double click on the product automatically opens up an image display of the test image. In the "Outline" window we can actually see that there are several datasets which include an error map, a coverage map and exposure map associated with the image (see Figure 16.4). A right click on any of the associated datasets and going to "Open With..." allows a Dataset viewer to appear which shows metadata and array data for the particular dataset.

16.4. Basic Analysis Capabilities

It should be noted that the overview and zoomed images displayed to the right of the displayed image during basic image analysis are the reverse for those when just dislaying the image, as illustrated in the "HowTo Display and Manipulate Images" chapter.

The basic analysis capabilities described in this chapter -- for application to `SimpleImage`s are;

- *1D profile plotting. Slices can be taken through the image*

- making a *histogram* of the whole image or of a certain region of interest, which is bounded by a circle, an ellipse, a rectangle or a polygon (the user should draw the bounding figure on the image)
- *aperture photometry* with a circular target aperture and an annular or a rectangular sky aperture
- *contour plotting and overlays*

Note

Note that all these functionalities are also available via the command line in the HIPE "Console" view. Using GUI/dialog interaction will copy the equivalent command to the "Console" view. This can be copied and pasted into a script (if wanted) for possible use in further, batch, processing.

16.4.1. 1D Profile Plotting

The 1D profile plotting capability allows the user to draw a straight line on an image and plot the intensity along that straight line.

After double-clicking on the Variable "myImage2" in the previous section the image was displayed in an "Editor" window. The Applicable Tasks are also available including the task `profile` (see Figure 16.5). A double-click on this item in the tasks list brings up another display of the image and allows interaction with the mouse

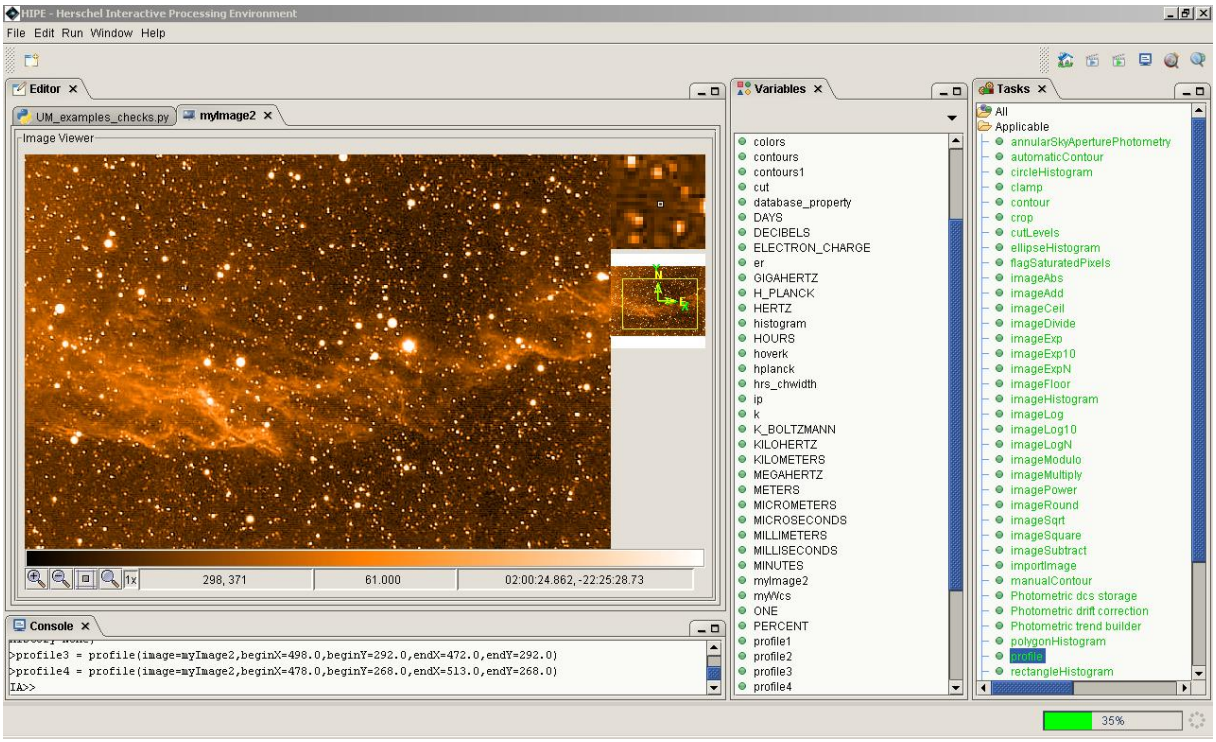


Figure 16.5. The available tasks show `profile` is available for "myImage2". Double-clicking on profile after first highlighting myImage2 in the Variables window creates a new display of the image together with the `profile` tool capabilities.Accessing the `profile` task

Now we can start drawing the straight line on the active image. The beginning of the line can be fixed by clicking once on the image. While moving the mouse over the active iamge, the straight line will be updated, until the end of the straight line is fixed by clicking a second time on the image. Simultaneously, the intensity plot along the straight line in an extension of the window below the displayed image (see Figure 16.6). The window is scrollable so the whole profile display can be seen by scrolling down (see Figure 16.7).

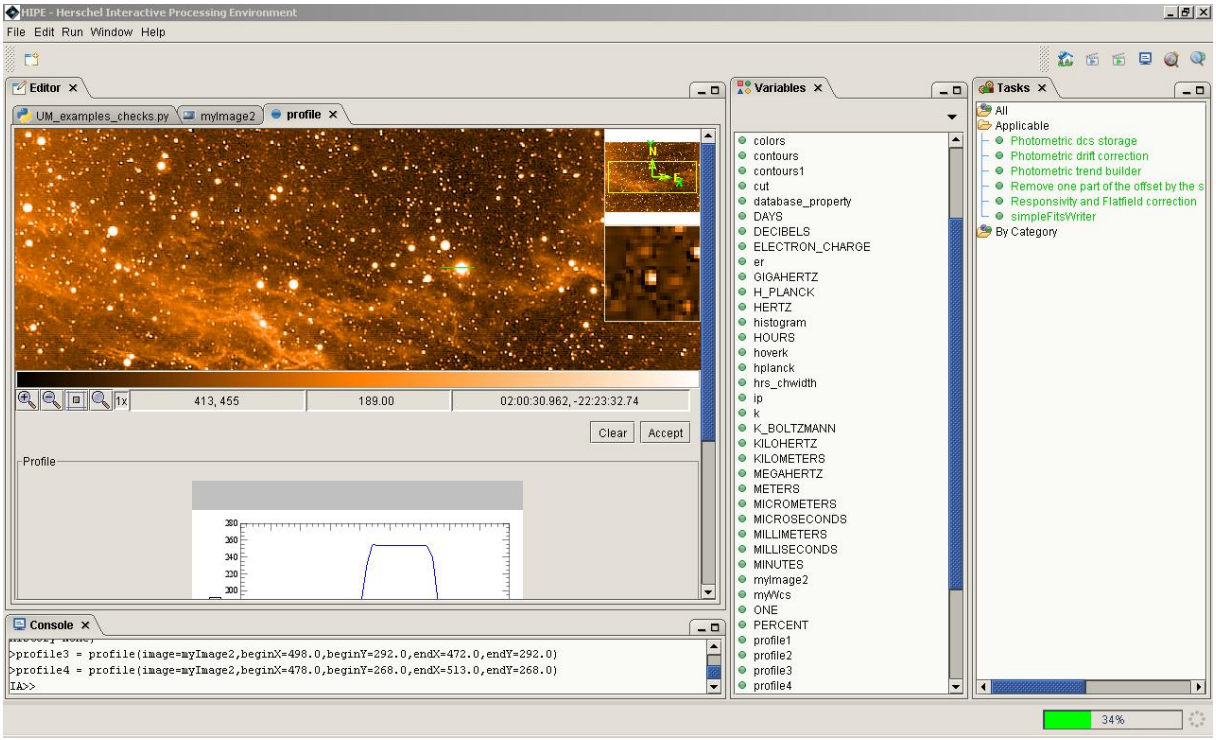


Figure 16.6. A 1D profile plot interaction

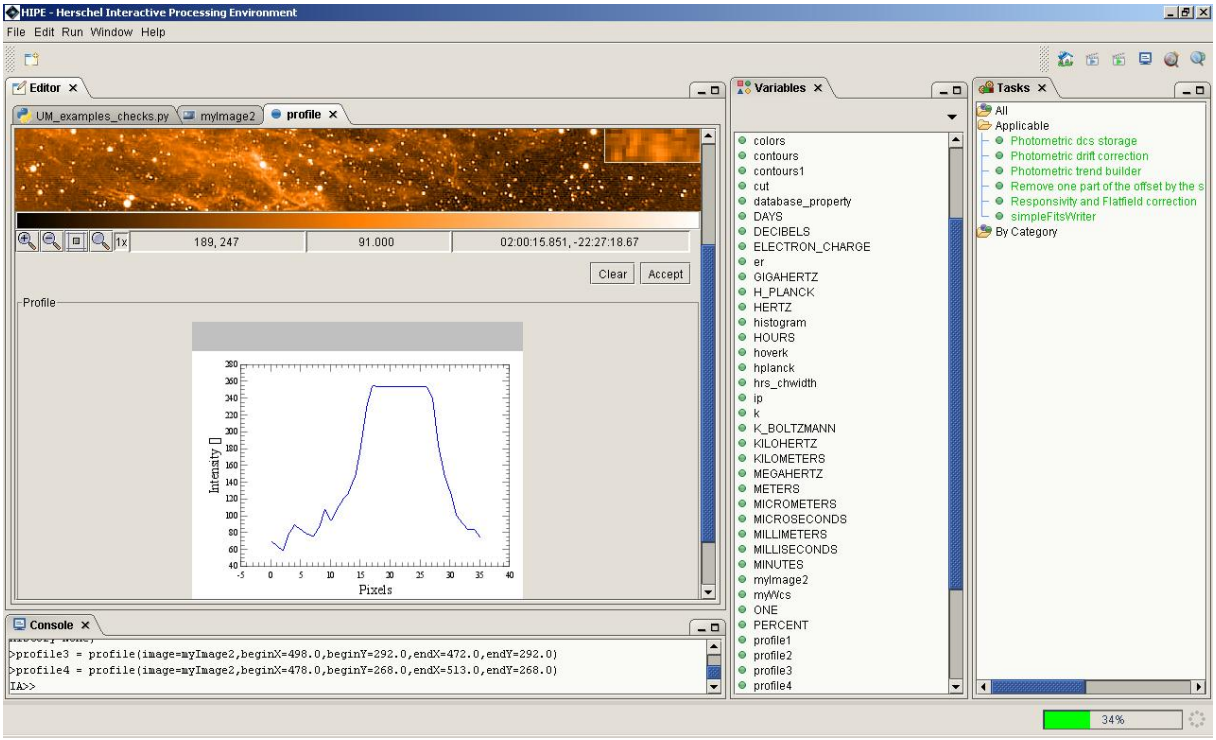


Figure 16.7. Same as for Figure 16.6 but scrolling down to show the profile display.

16.4.2. Area Histogram

One can make a histogram of an image as a whole, or of a certain region of interest which is specified by the user. This region can be bounded by a CIRCLE, an ELLIPSE, a RECTANGLE or a POLYGON, which has to be drawn on the image, or can be for the whole image.

We start the procedure for making an area histogram by choosing one of the `imageHistogram`, `polygonHistogram`, `circleHistogram`, `ellipseHistogram` or `rectangleHistogram` tasks.

First click on a `SimpleImage` in the list of Variables, e.g. "myImage2". Then double-click one of the histogram tasks in the Tasks view. This brings up a new image and activates the mouse so that an image area can be selected. To cover an area with a circle, ellipse or rectangle do a click-and-drag. On release, the area selected is shown overlaid on the image.

The histogram is constructed from the intensity values of the selected pixels and the input of the min and max cut levels in the boxes provoided, plus the number of bins for the histogram. Hitting the "Accept" button does several things.

- A histogram is formulated in the Editor window (scroll down).
- The equivalent command line is shown in the Console view which includes a named output object.
- The histogram values are placed in a dataset that appears in the Variables list. In Figure 16.8 this is called "histogram2".
- The output (e.g., "histogram2") appears highlighted in the Variables view and appears in the Outline view. Double-clicking this output value in the Variables view provides the histogram together with key information (see Figure 16.9).

For the `polygonHistogram` the only difference is that each corner is indicated by a single mouse click. The polygon area completion is indicated by a mouse double-click. Otherwise, this works in the same way as the other histogram tasks.

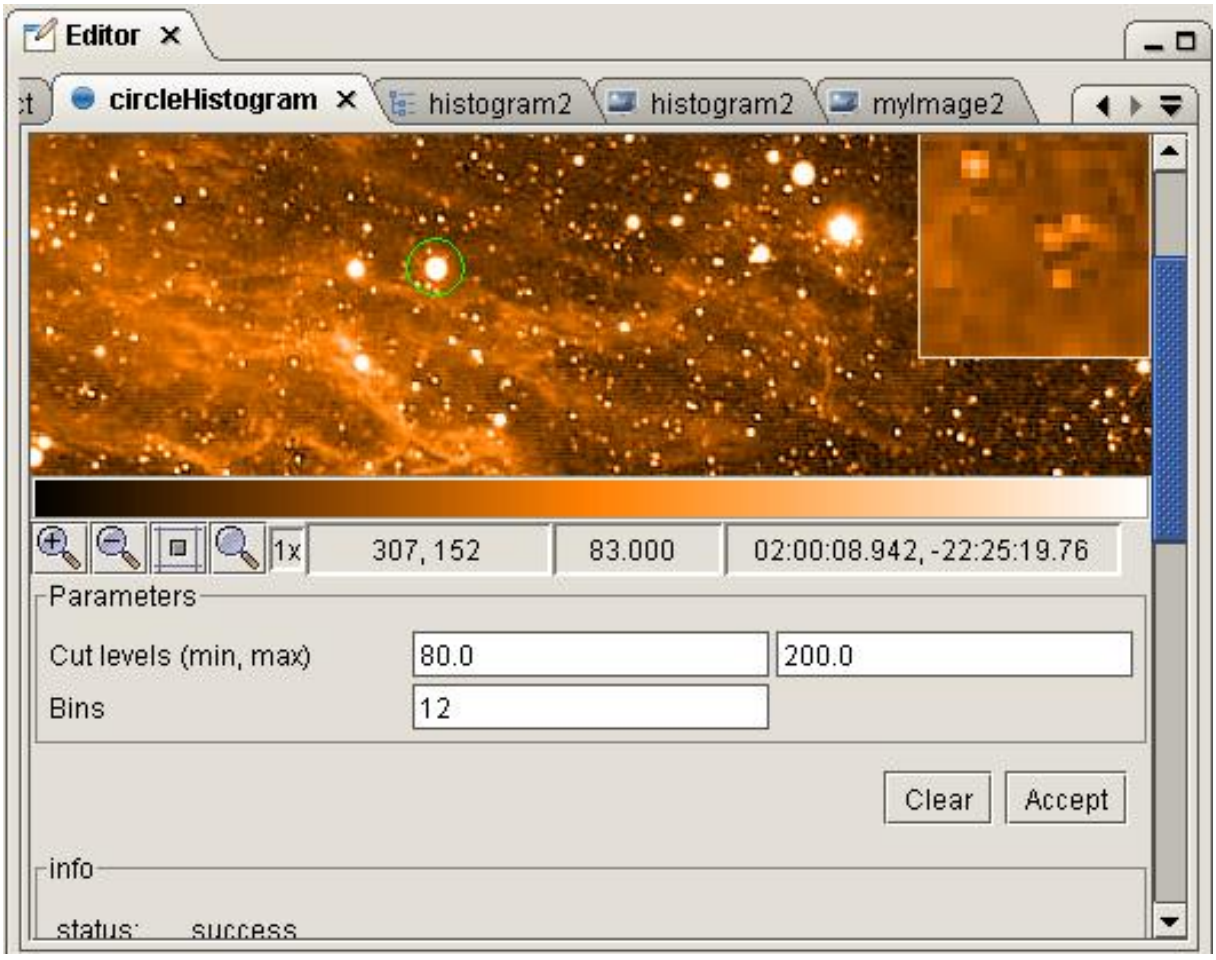


Figure 16.8. Circle histogram area selection and parameter selection. These appear in the HIPE "Editor" view.Circle histogram

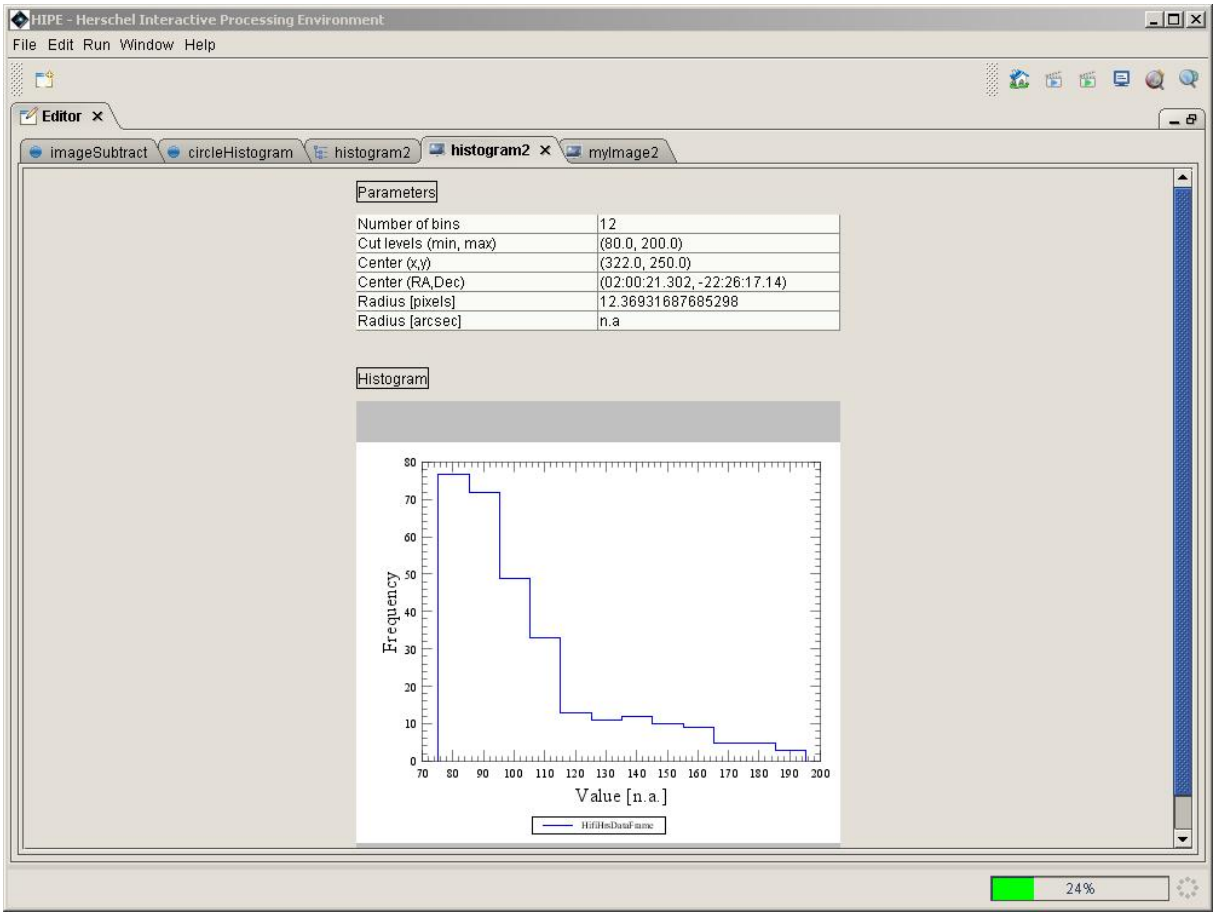


Figure 16.9. Display of the histogram results held in the histogram output in an expanded Editor view.Histogram display

16.4.3. Aperture Photometry

One can also perform aperture photometry on an image, using a circular target aperture and an annular or a rectangular sky aperture. There are five algorithms that can be used to estimate the sky : average, median, mean-median, the synthetic mode and daophot. In the mean-median method all values further away from the median than a specified number of times the standard deviation (i.c. 1.5) are discarded and the remaining values are averaged. The `daophot` method is a translation of the algorithm used in the aophot package from IDL to Java.

A start to doing annular aperture photometry can be made by choosing the `annularSkyAperturePhotometry` item in the Tasks menu following selection of the appropriate image in the "Variables" menu. This provides the image in the "Editor" view below which is the dialog for the aperture photometry task options (see ???). This is easiest seen by expanding the "Editor" view window and scrolling down below the image.

There are three mechanisms by which the photometry area can be identified.

- By click-and-drag mouse interactions.
- By pixel region selection.
- By sky coordinate selection.

The default is by mouse interaction. A single click on the image allows places a circle on the image at the mouse point. The user then inputs a value for the object aperture radius and inner and outer radii for sky subtraction. A selection should be made for the appropriate fitter (e.g. daophot) and pressing accept

leads to an output in the variable "result". The circular radii are shown on the image (see Figure 16.10). To redo -- press the "Clear" button.

A sky position or pixel position can also be selected. Selection of either of these possibilities enables an update to the input screen allowing the sky/pixel values to be input. For the sky position (at present) the format of input is "02:00:39.4" for RA and "-22:27:20.6" for Dec. Note that the quotations are necessary as the input is a string. This is likely to be changed to allow various input types in the future.

The results can be displayed by double-clicking on the "result" variable shown in the "Variables" menu (see Figure 16.11)

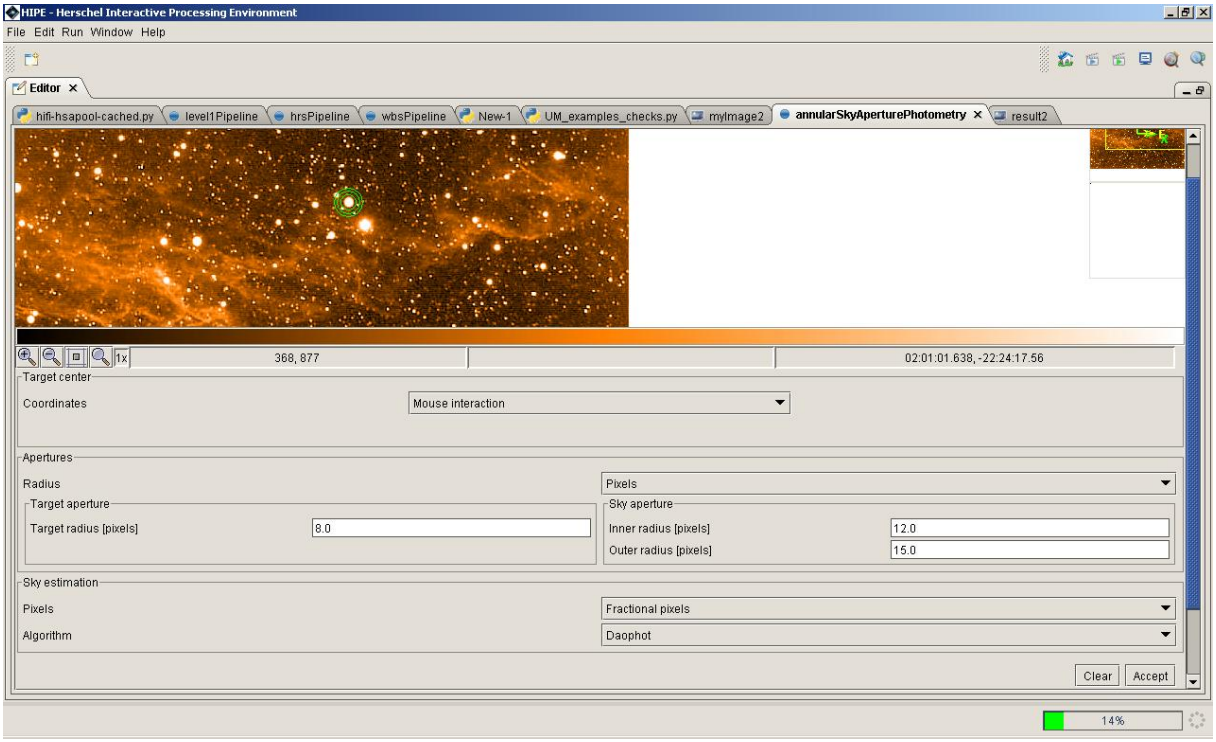


Figure 16.10. Aperture photometry with an annular sky aperture as displayed in HIPE.

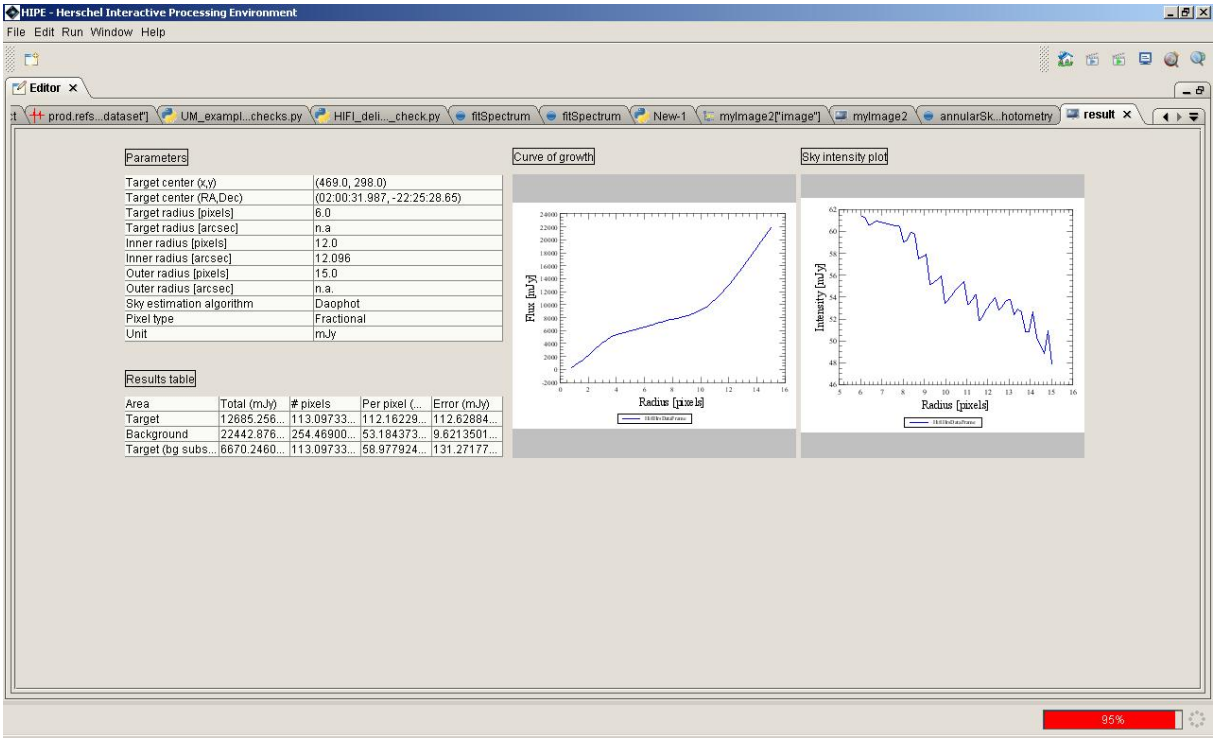


Figure 16.11. Aperture photometry results plot and tables. Note that n.a. relates to "not applicable" and typically will occur when units are not assigned to the image.Results of sky aperture measurement.

A similar capability is available for using a rectangular sky aperture. Rectangular aperture photometry can be done by choosing the `rectangularSkyAperturePhotometry` item in the Tasks menu following selection of the appropriate image in the "Variables" menu. Similar to the above, a single mouse click can be used to identify the target or a sky or pixel position can be indicated by the user. A rectangular sky aperture can then be selected by a click-and-drag across a region of the image (see Figure 16.12). Following the calculation for the first position, the same rectangular box can be used for the sky and a further single click on the image picks out a new object. Hitting the "Accept" button allows another result for this new position.

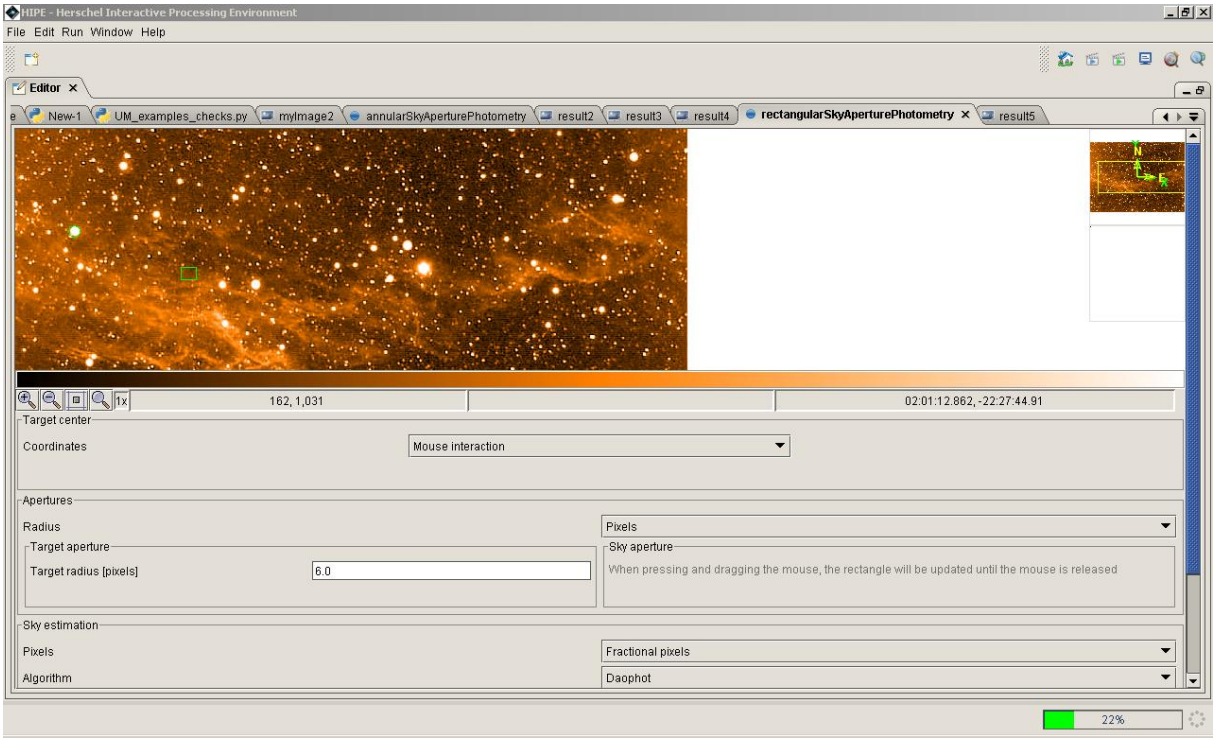


Figure 16.12. Aperture photometry with an annular sky aperture as displayed in HIPE.

The results for both aperture photometry tasks provide the curve of growth. This is a plot of the target flux as a function of the target radius. Such a plot can be used to see whether a valid target radius has been given. When an annular aperture is used to estimate the sky, a sky intensity plot is also shown. This plot shows the intensity per sky pixel as a function of a varying inner radius (the outer radius is fixed).

16.4.4. How to run the source extractor in HIPE

16.4.4.1. Introduction to source extraction in HIPE

HIPE includes a source extractor ("sourceExtractor"), designed primarily for use on PACS and SPIRE maps. It currently features implementations of the DAOPHOT (classic) and SUSSEXtractor algorithms for extraction of point sources with a known profile. This chapter explains how to use the source extractor; advanced usage is described in the User's Reference Manual entry for the SourceExtractorTask.

16.4.4.2. Extracting sources from a SimpleImage

The example image used in this section is a simulation of a SPIRE 250 micron image of a grid of sources. This is imported into HIPE, where it appears as a Simple Image. In the Work Bench perspective, in the Tasks view, under "Applicable", "sourceExtractor" appears as one of the applicable tasks for a SimpleImage. This is shown in Figure 16.13.

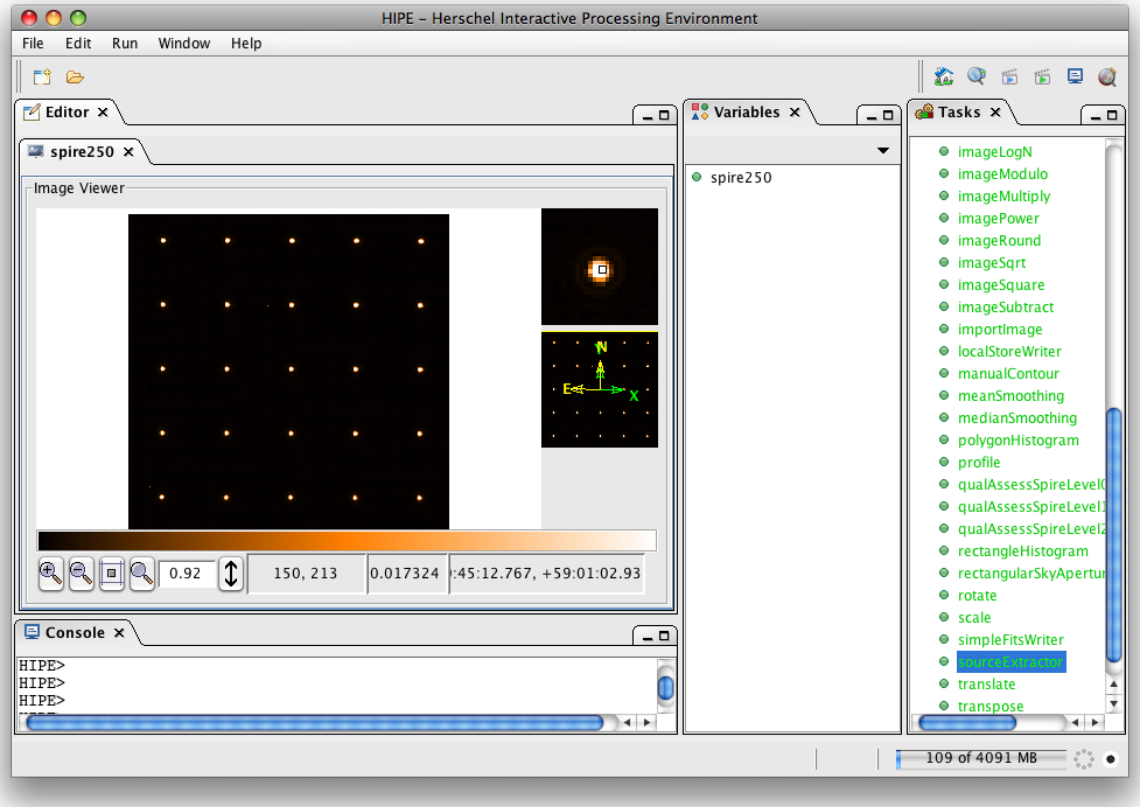


Figure 16.13. The Work Bench perspective, showing the example image, with "sourceExtractor" highlighted in the list of applicable tasks.

Experienced users are likely to use the sourceExtractor task as part of a Jython script. However, the easiest way to learn how to use the task is through the graphical user interface (GUI). Double-clicking on the "sourceExtractor" in the applicable tasks list will open the GUI for the source extractor. This is shown in Figure 16.14.

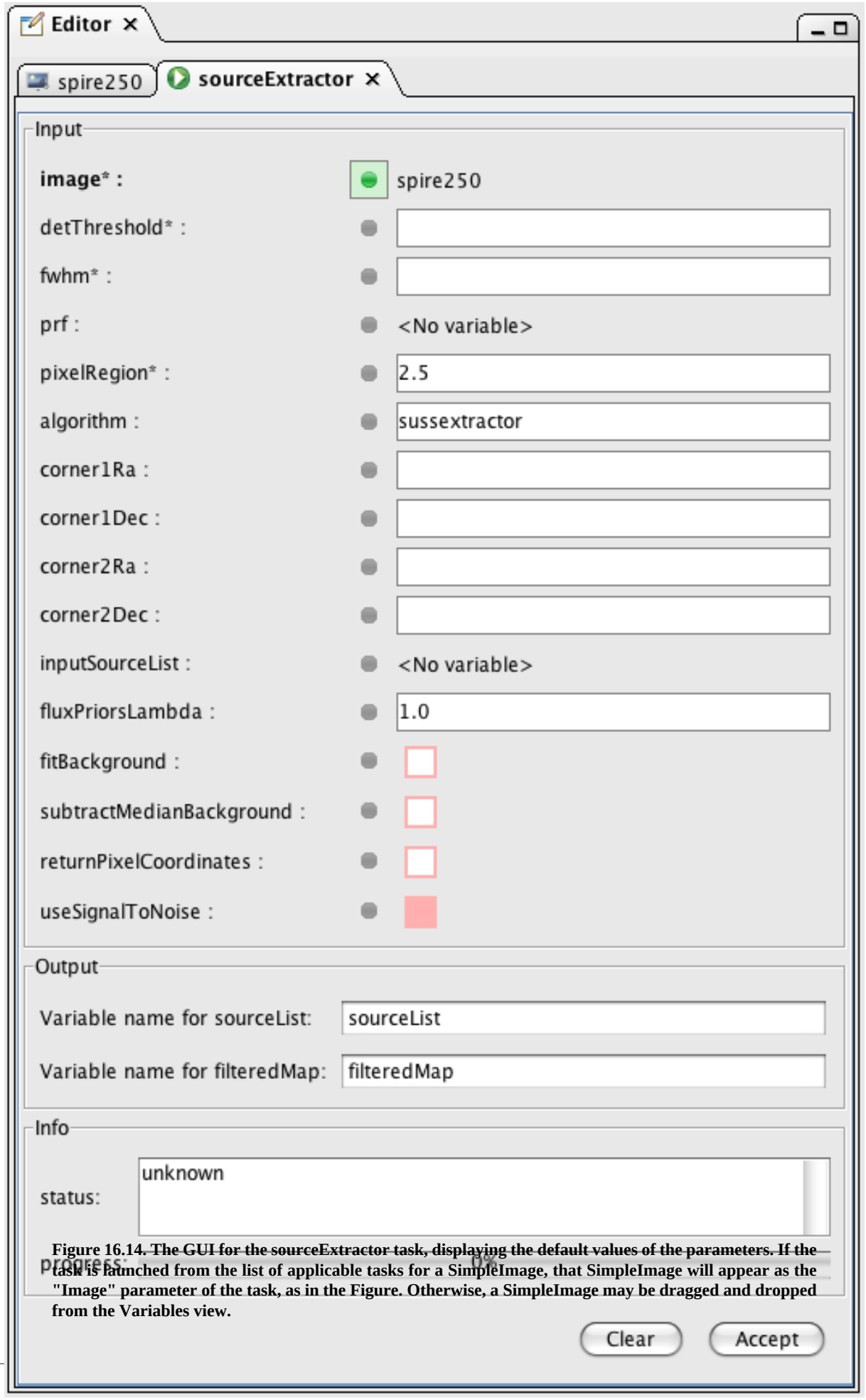


Figure 16.14. The GUI for the sourceExtractor task, displaying the default values of the parameters. If the task is launched from the list of applicable tasks for a SimpleImage, that SimpleImage will appear as the "Image" parameter of the task, as in the Figure. Otherwise, a SimpleImage may be dragged and dropped from the Variables view.

In order to launch the task, the mandatory parameters must be filled in. These are marked with an asterisk (*) in the GUI. If the mouse pointer is placed over the name of a task parameter, a box will appear giving information about that parameter. In this example, the parameters still to be filled in are the detection threshold ("detThreshold") and the full-width-half-maximum (FWHM) of the (Gaussian) point response function ("fwhm"). In this example, these are set to be 5.0 for the detection threshold (corresponding to a 5-sigma detection) and to 18.0 arcsec for the FWHM, corresponding to the approximate size of the SPIRE beam for the PSW array. Clicking "Accept" launches the task. The status and progress may be followed in the task GUI. Long-running tasks may be interrupted by selecting "Stop" from the "Run" menu. Figure 16.15 shows the task GUI once the task has been completed.

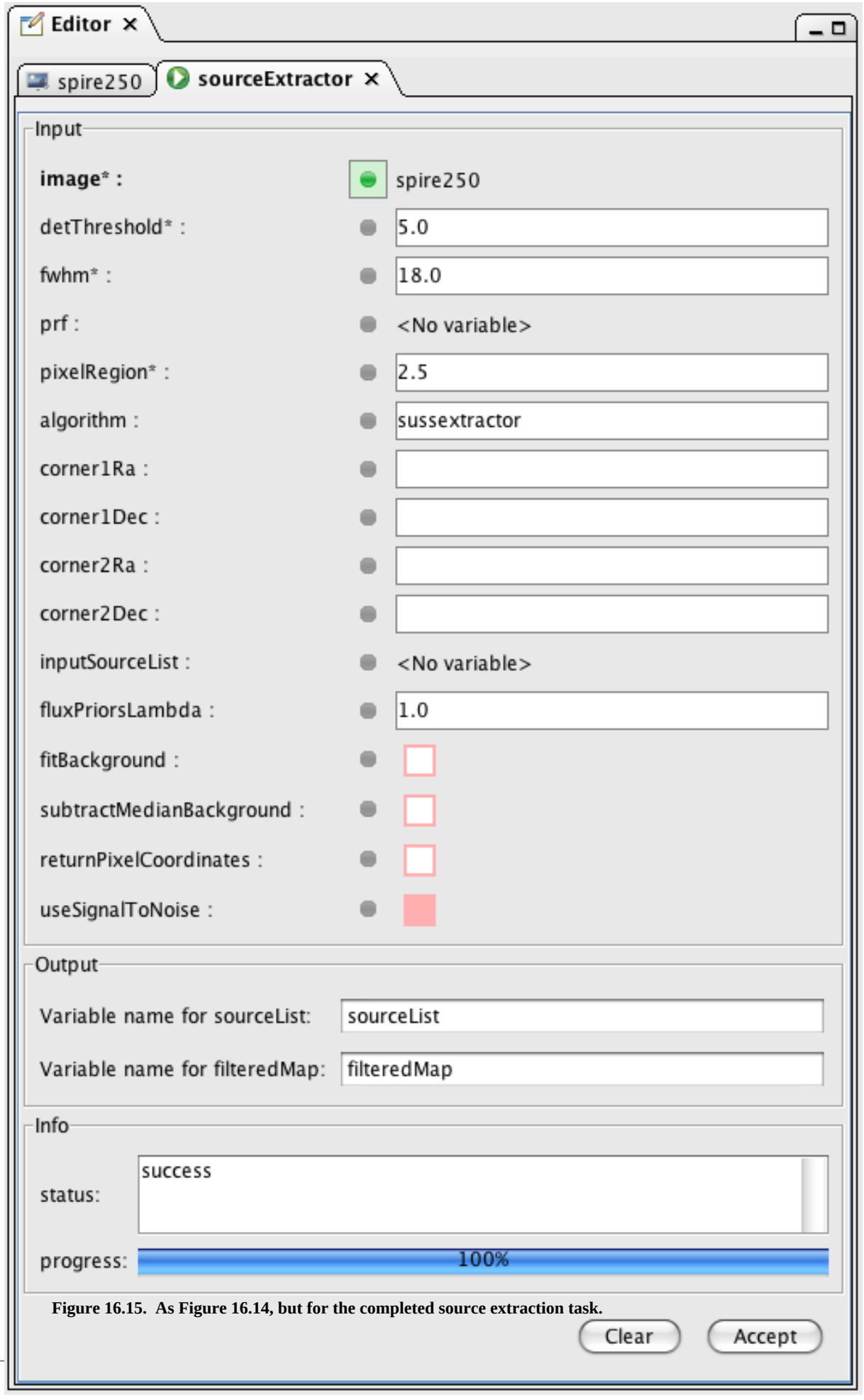


Figure 16.15. As Figure 16.14, but for the completed source extraction task.

The task GUI is essentially a convenient method of created a Jython command to execute the task. Once the task has been completed, the Console view displays the Jython commands executed to perform the sourceExtractor task. This shown in Figure 16.16.

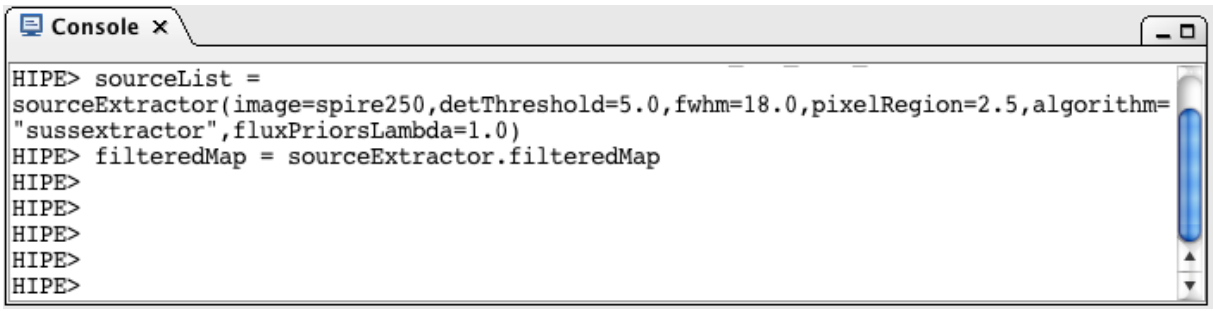


Figure 16.16. The Console view after the sourceExtractor task has been launched from the task GUI.

In order to repeat the source extraction as part of a Jython script, the user need simply copy these commands into the script, rather than entering parameters manually into the GUI. Two commands were executed by the task. (1) The first created the primary output, a SourceListProduct, given the name "sourceList". This will appear in the Variables view. Double- clicking on "sourceList" opens the SourceListProduct in the Product Viewer. This is shown in Figure 16.17, with the Dataset highlighted. The Dataset in a SourceListProduct is a "SourceListDataset", which is a kind of TableDataset.

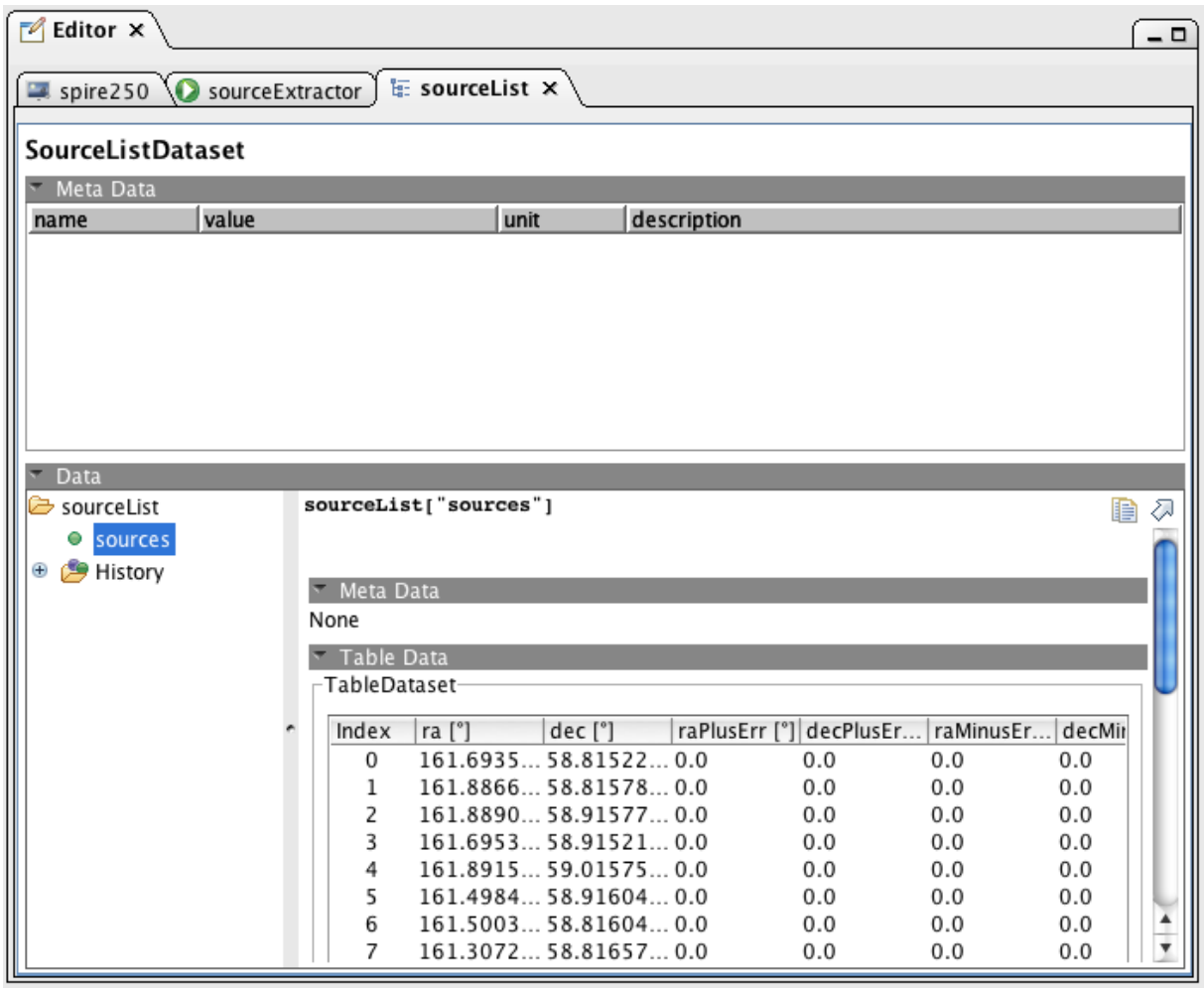


Figure 16.17. The SourceListProduct shown in the Product Viewer, with the dataset highlighted.

(2) The second command obtains an optional output from the sourceExtractor task, the "filteredMap", which is another SimpleImage. For the SUSSEXtractor algorithm, this is equal to the input map

convolved with the point response function, such that the value at each pixel gives an estimate of the flux of a source, in mJy, assuming there is a source located at the centre of that pixel. For the DAOPHOT algorithm, the filteredMap gives the input map convolved with the DAOPHOT kernel. This may be viewed by double-clicking on "filteredMap" in the Variables view.

16.4.4.3. Displaying extracted sources on an image

Having extracted the sources in an image, the user is likely to want to see where the extracted sources lie in the image. This can easily be done by displaying the original SimpleImage in the Editor view, and then dragging and dropping the SourceListProduct from the Variables view onto the displayed image. A circle is now overlaid at the location of each source. The result of this is shown in Figure 16.18.

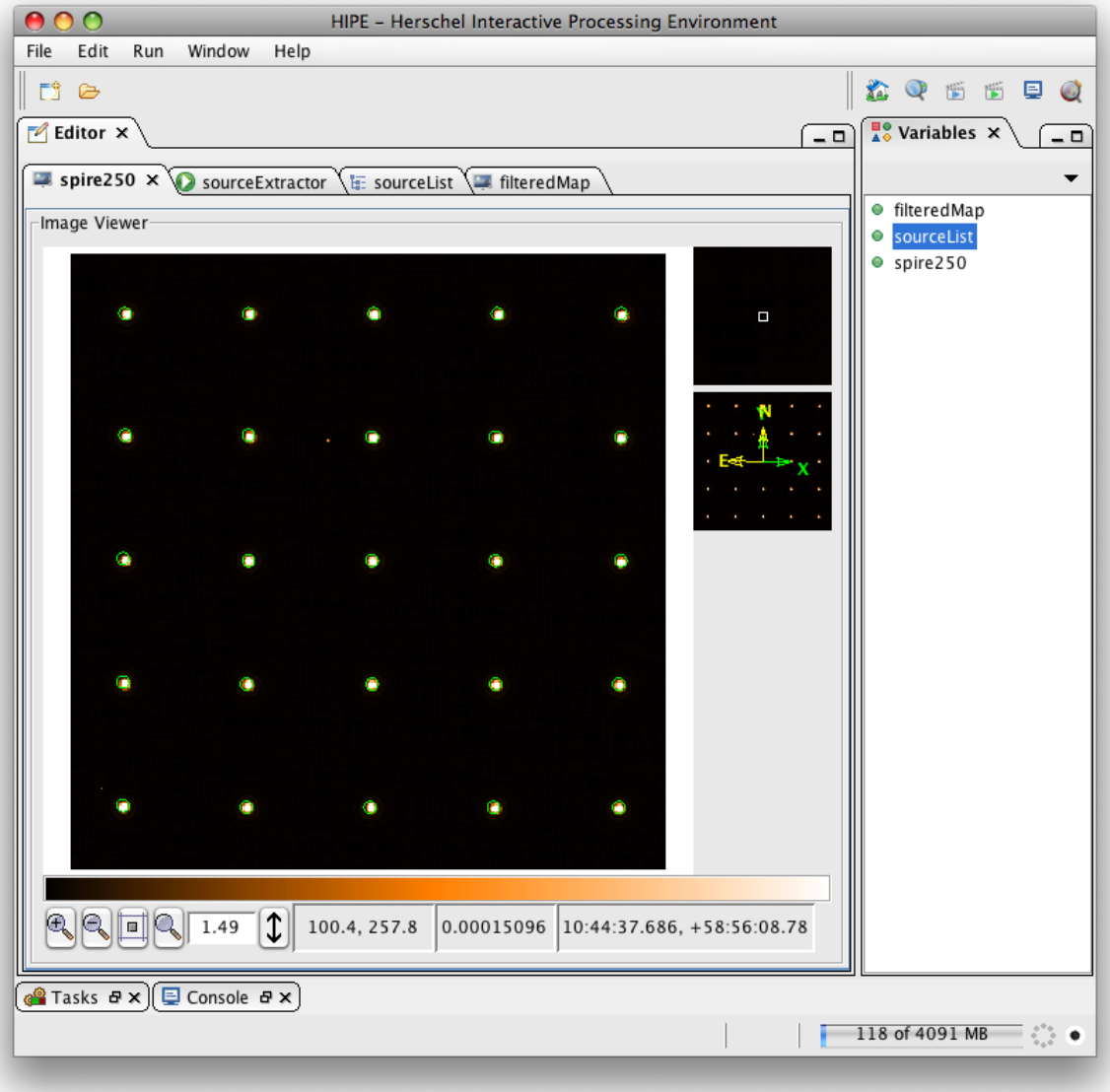


Figure 16.18. The original image with the locations of the extracted sources overlaid as circles.

16.4.4.4. Specifying positions of known sources

Often the user will be interested in the fluxes of sources at known positions, e.g., from previous observations made at mid-infrared wavelengths. If these sources are contained in a SourceListProduct, this can be used as an input to the sourceExtractor task to specify the positions of sources. The task will then give the fluxes of sources at those positions. This is shown in Figure 16.19, where the

known sources are provided by another SourceListProduct, "knownSourceList". This is specified as the "inputSourceList" parameter for the task.

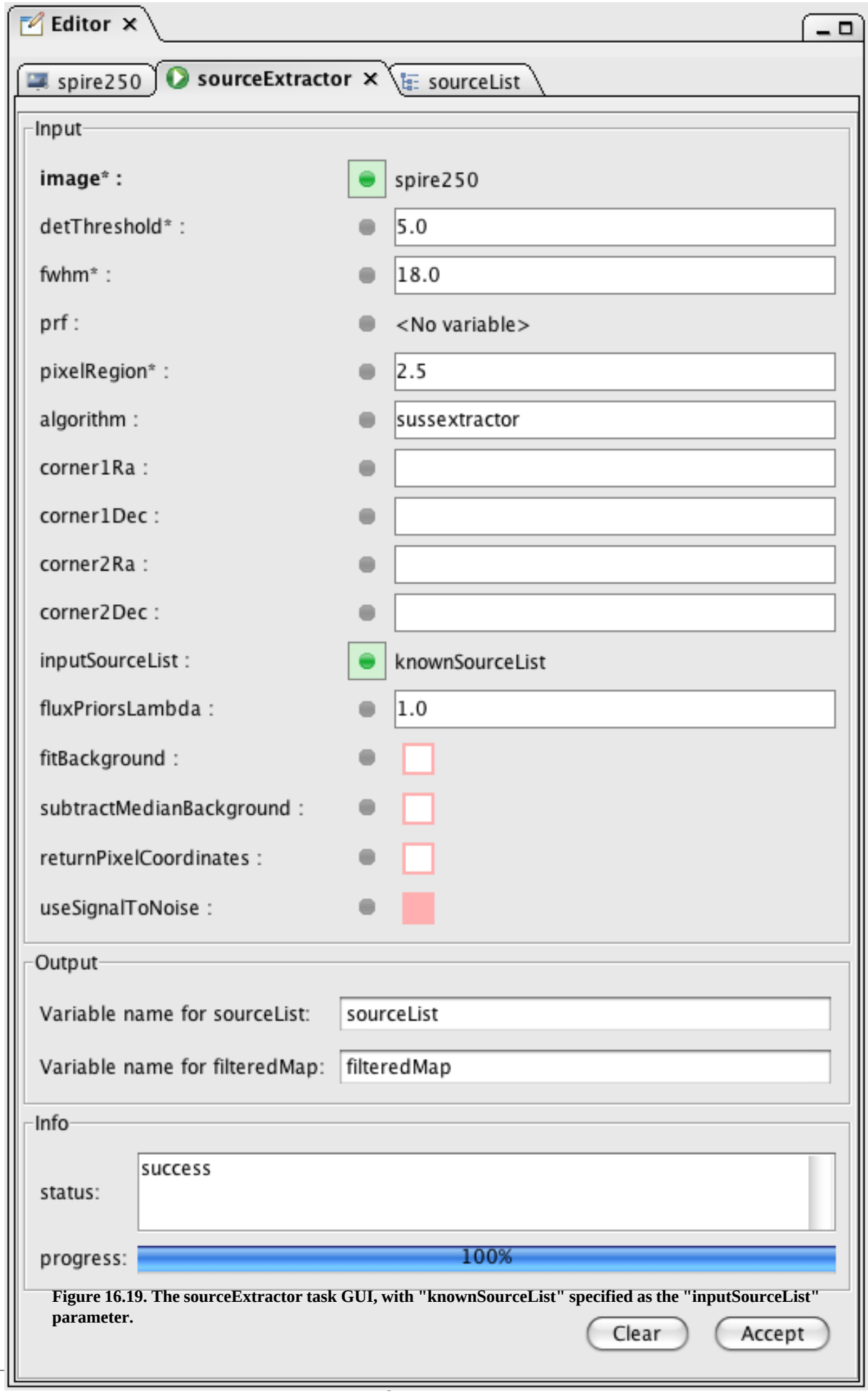


Figure 16.19. The sourceExtractor task GUI, with "knownSourceList" specified as the "inputSourceList" parameter.

16.4.4.5. Specifying a custom point response function

By default, the sourceExtractor task assumes a Gaussian point response function (PRF), with full-width-half-maximum (in arcsec) provided by the "fwhm" parameter. Alternatively, a custom PRF may be specified by the "prf" parameter. This should be a SimpleImage of odd dimension, with the peak at the centre, normalised such that the PRF image gives the (central pixels) of a point source of flux 1 Jy, in the units of the input map.

16.4.4.6. Working with source lists in ASCII files

One may wish to work with external ASCII files, either to export Herschel source lists, or to import other source lists, either to provide known positions for the HIPE source extractor, or for use in other analyses. This may easily be done using the asciiTableWriter and asciiTableReader tasks. To save a Herschel source list as an ASCII file, first the SourceListDataset must be retrieved from the SourceListProduct. A SourceListDataset is a kind of TableDataset, and consequently the asciiTableWriter will appear in the applicable tasks list. This is shown in Figure 16.20.

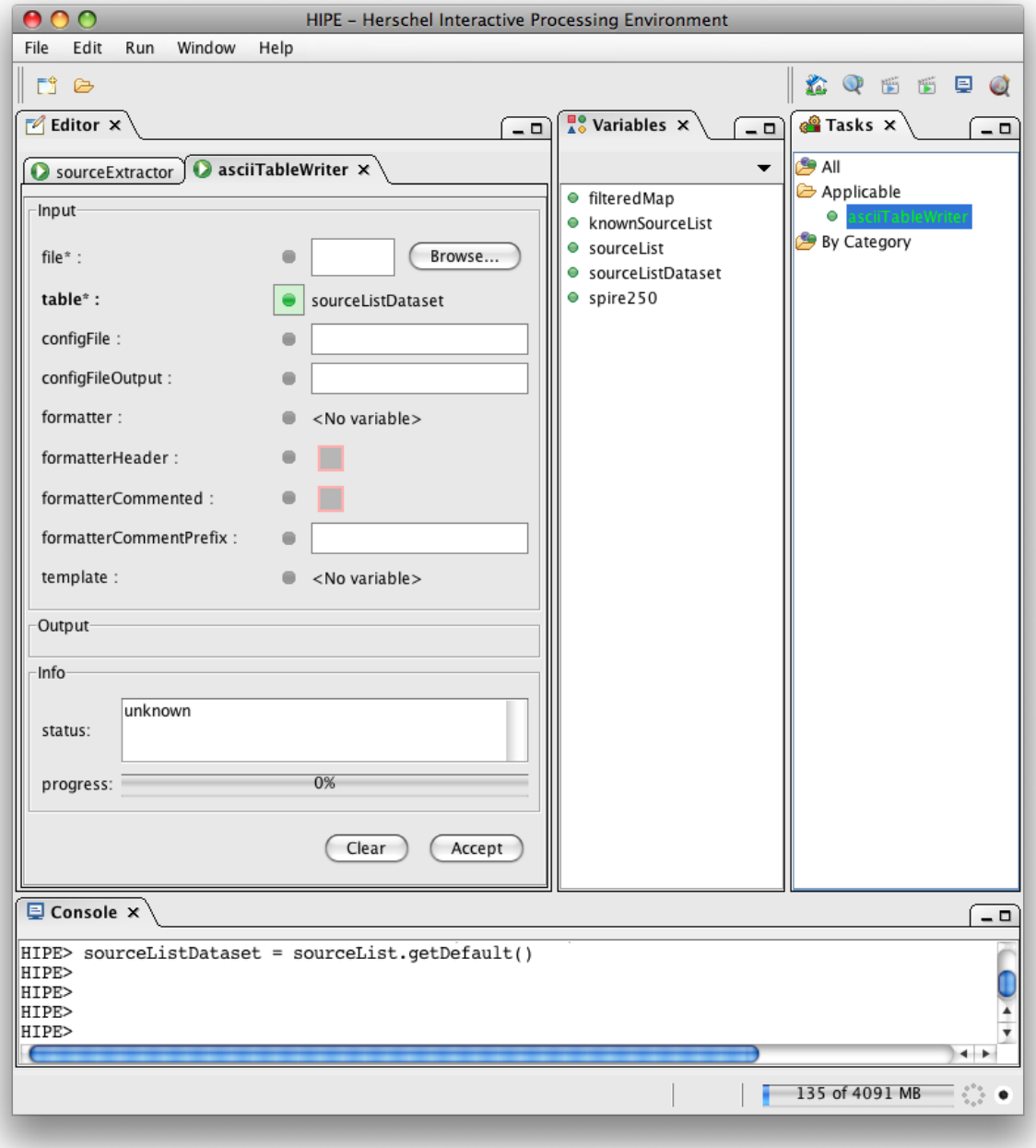


Figure 16.20. Exporting a SourceListProduct as an ASCII file. First, the SourceListDataset is retrieved using the command "sourceListDataset = sourceList.getDefault". Then "sourceListDataset" is highlighted in the Variables view. In the Tasks view, "asciiTableWriter" now appears as an applicable task. Double-clicking this opens the GUI for the asciiTableWriter, as shown in the Figure.

To import an ASCII file as a SourceListProduct, first the asciiTableReader should be launched. This is shown in Figure 16.21, for a source list contained in "sourceList.txt".

This creates a TableDataset, "table". This may be converted into a SourceListProduct called "importedSourceList" using the command "importedSourceList = SourceListProduct(table)". It is important to note that the column names in the imported source list must match the default column names in a SourceListDataset ("ra", "dec", "flux", etc.), although whether these names are upper, lower or mixed case does not matter.

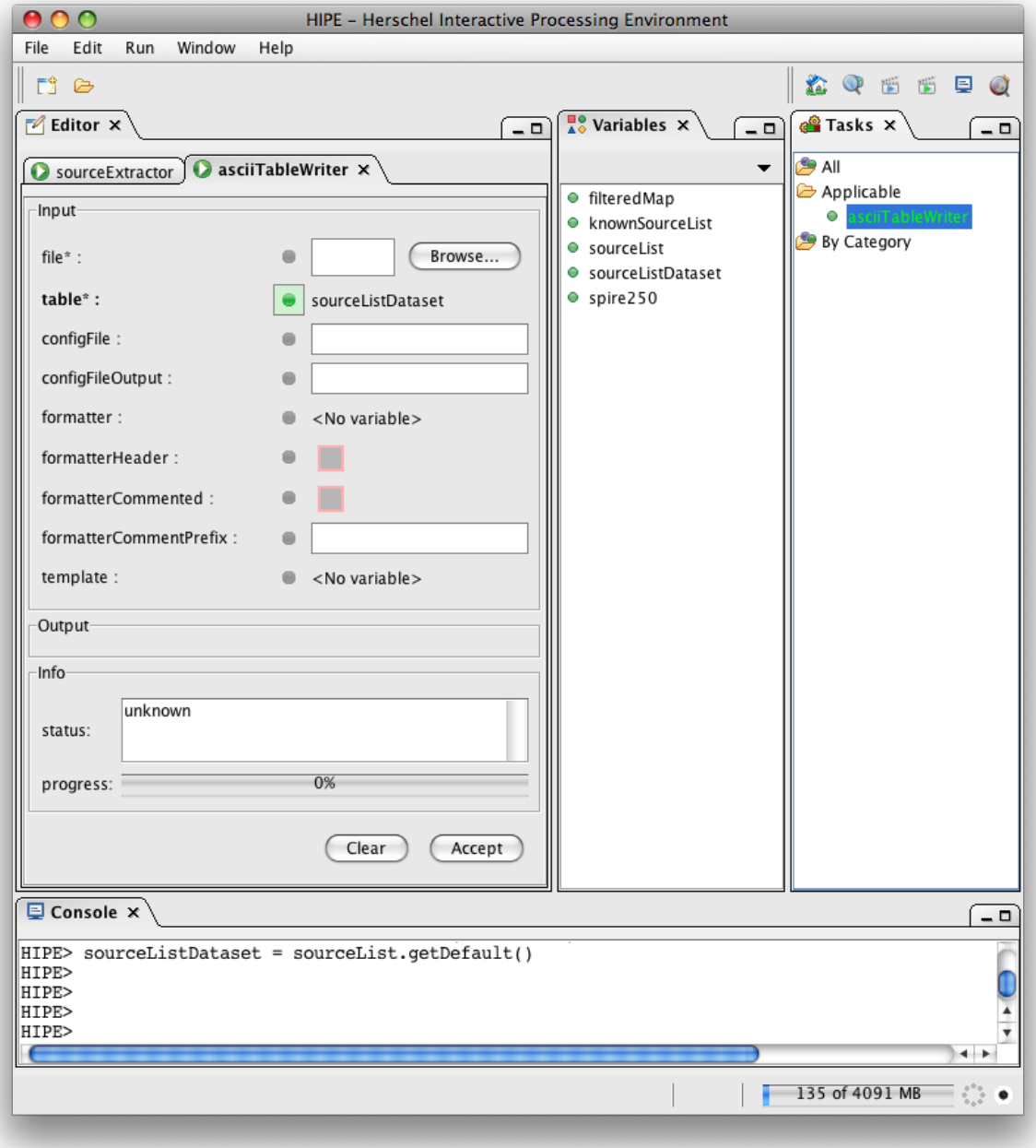


Figure 16.21. The asciiTableReader task for importing sourceList.txt.

16.4.4.7. Working with source lists in FITS files A SourceListProduct may be exported or imported as a FITS file.

This may be useful for working with applications such at TOPCAT. To export a SourceListProduct as a FITS file, simply select the name of the SourceListProduct in the Variables view, and the "simpleFitsWriter" task should appear in the list of applicable tasks. To import a FITS file as a SourceListProduct, simple select "Open" from the "File" menu, and select the FITS file. This launches the simpleFitsReader, and the FITS file will be imported as a Herschel Product. If the FITS file contained an exported Herschel SourceListProduct, then the imported Product will be of this type. Otherwise, the imported Product will be a generic Product, with the source list contained in a Dataset. For example, if the Dataset has the name "HDU_1", then a SourceListProduct may be created using the command

```
HIPE> importedSourceList = SourceListProduct(sourceList["HDU_1"])
```

This is shown in Figure 16.22.

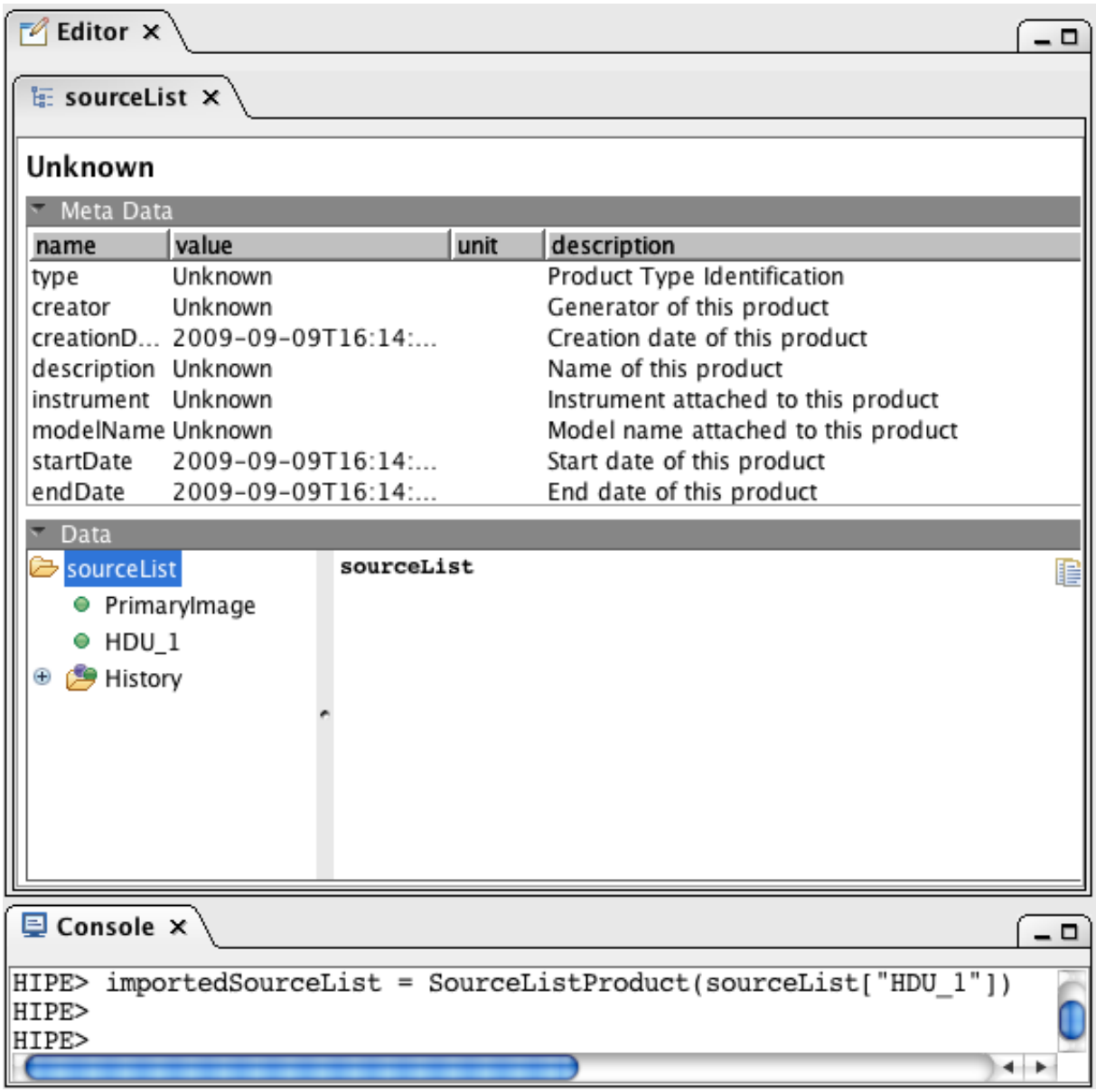


Figure 16.22. The asciiTableReader task for importing sourceList.txt.

16.4.5. Contour Plotting

Another functionality of the toolbox is contour plotting. A contour plot connects all points in the image with the same intensity, like isobars on a weather map.

First we create the contours. We then, later, overlay these contours on any image we wish.

There are two methods for providing a set of contours for display. The first is an `automaticContour` where the number of contour levels and a min and max contour level are selected and the intermediate levels are generated automatically with linear, ln or log intervals of intensity. The second is a `manualContour` where the values of each contour level are individually put in by the user.

In either case we, as usual, start by clicking the name of the image we want to be countoured from the "Variables" list. Then we choose either `automaticContour` or `manualContour` by double-clicking these items in the "Tasks" list (see Figure 16.23for example).

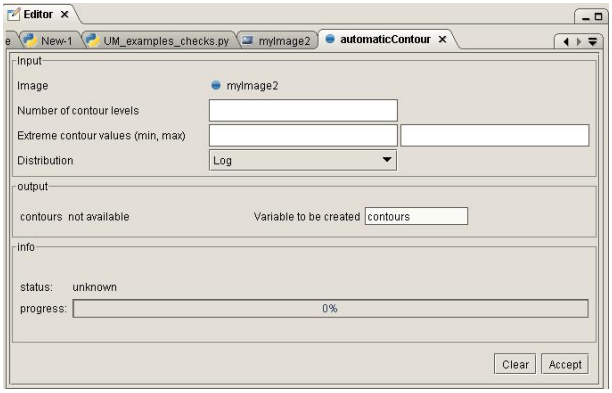


Figure 16.23. Dialog for `automaticContour`.

In either case, we create an output (editable value for user), e.g. "contours". When hitting the "Accept" button this is the variable that will store the contour results.

Chapter 17. How to Save/Play Back Scripts in HIPE

Herschel Editorial Board

17.1. Introduction

Hipe keeps a running record of all items typed or actiona taken using the graphical interface (Mouse points and clicks). The purpose of this article is to identify the steps you can take to save this information. The goal is to be able to keep a record of all actions and create a Jython script which can be resused or slightly modified.

17.2. How to save/ play back a script in HIPE

These are the steps to follow using to save all the commands which were given to HIPE during your session. In the tab bar with the Console view there is a tab called history. You can also bring this History view up from the Window--Show view pull down menu.

1. In the History view. Mouse left mouse click on the column called "Command". The entire column should be now selected.

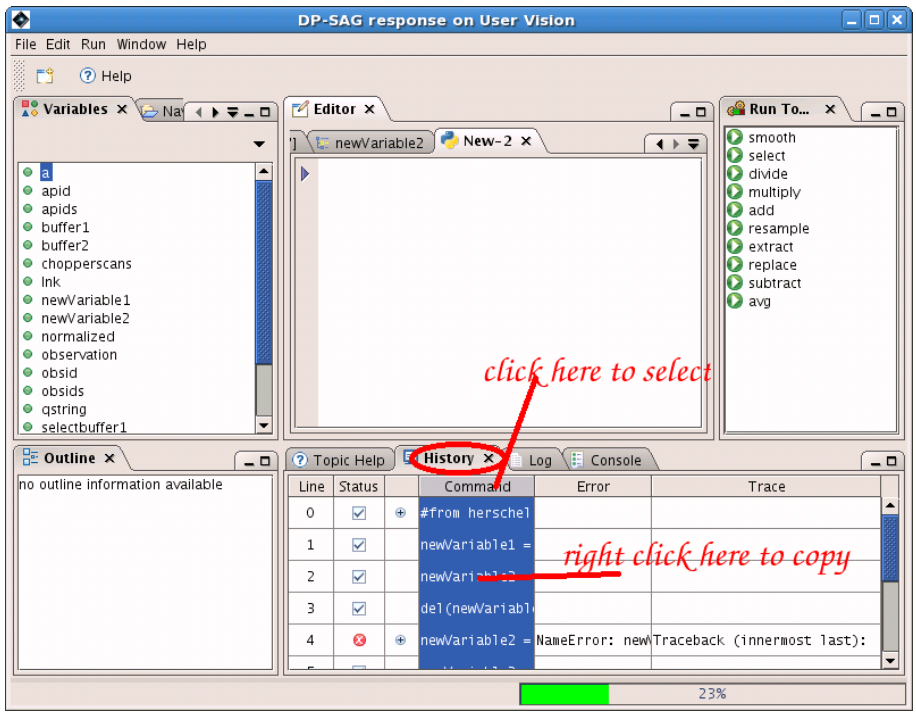
2. From the "File" button at the top left of the HIPE window, create a new (blank) Jython script.

3. Then right-mouse click on the selection and Choose copy.

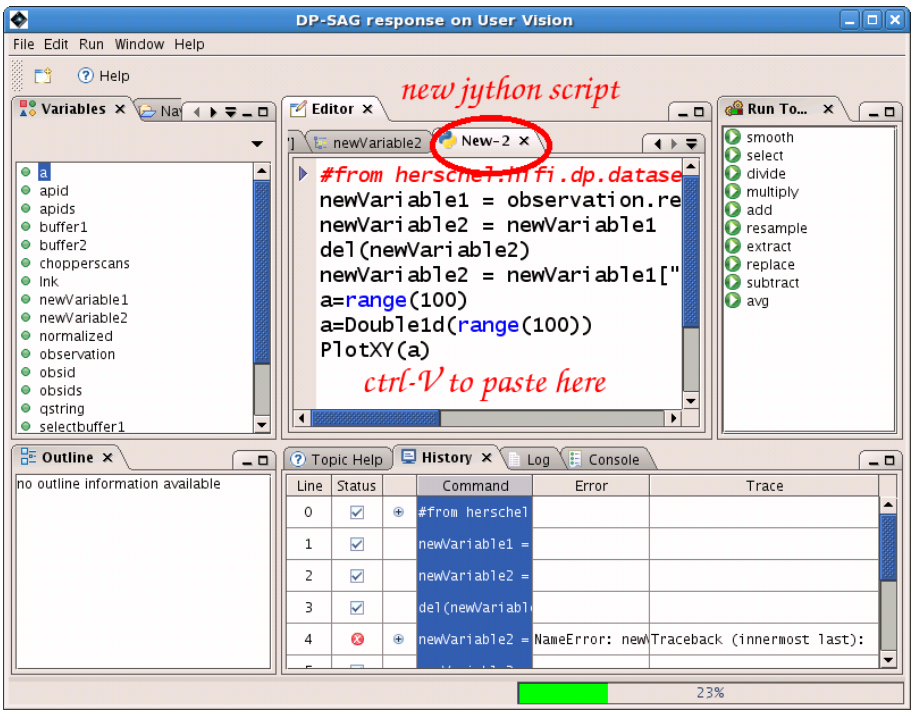
4. Move the cursor to the blank Jython script page and either select the "Paste" command in the Edit pull down menu or type Ctrl-V. The script will appear in the Editor window.

5. Your new Jython script can be saved for later importing (via the Navigator view).

The following screen shots show what is described above.



And...



To save the script to file (default extension .py), click on the Editor view tab that contains the script -- which brings the script to the foreground -- then hit CTRL-S. The file will be saved. If you have not already provided a directory and name for the script then you are prompted for one, otherwise the previous version is overwritten at the same place in your directory structure.

Alternatively -- click on the appropriate Editor tab (as above) and then go to the "File" pull-down menu at top left of HIPE. Go to "Save" or "Save As...".

17.3. How to Play Back a Script from the Command Line

The main way in which a script is developed and run in the HIPE environment is via the Editor screen, as described above. It can also then be saved -- as noted above. However, it is also possible to play back a saved script that is on the disk using the `execfile` command. Enter something similar to the following on the command-line of the Console view (do not forget the quotation marks).

```
execfile("<full path name><file name>")
```